# MapReduce: внутри, снаружи или сбоку от параллельных СУБД?

*Кузнецов С.Д.*
*kuzloc@ispras.ru*

**Аннотация.** Обсуждаются подходы к использованию технологии MapReduce в аналитических СУБД. Рассматриваются подходы, при которых MapReduce реализуется внутри ядра параллельной СУБД, используется в качестве коммуникационной инфраструктуры новой параллельной СУБД и применяется автономно в симбиотическом единстве с параллельной СУБД. В качестве примера применения первого подхода анализируются особенности организации массивно-параллельных СУБД Greenplum Database и nCluster компаний Greenplum и Aster Data Systems соответственно. Второй подход применяется в проекте HadoopDB университетов Yale и Brown. Наконец, третий подход развивается в компании Vertica.
**Ключевые слова.** Массивно-параллельные аналитические СУБД, MapReduce

## *1. Введение*

Как отмечалось в Клермонтском отчете [1], "*... сбор, интеграция и анализ данных больше не считаются расходами на ведение бизнеса; данные – это ключ к достижению эффективности и прибыльности бизнеса. В результате быстро развивается индустрия, поддерживающая анализ данных*". Если к концу прошлого века программные средства, пригодные для организации хранилищ данных и выполнения над ними оперативного анализа, можно было пересчитать по пальцам одной руки (IBM DB2, Teradata, Sybase IQ, Oracle, частично Microsoft SQL Server, причем только в DB2 и Teradata поддерживалась массивно параллельная архитектура без общих ресурсов между узлами (sharing nothing) и только в Sybase IQ использовалось поколоночное хранение таблиц (column-based store)), то с начала нового тысячелетия активизировалось направление специализированных аппаратно-программных систем, полностью ориентированных на поддержку хранилищ данных и/или анализа данных (Data Warehouse Appliance или Analytic Appliance; в дальнейшем для соблюдения точности и для краткости я буду обозначать это направление и относящиеся к нему системы аббревиатурой DWAA). Основной целью этого направления являлось и является создание аппаратно-программных средств, которые были бы существенно дешевле средств поддержки хранилищ данных, предлагаемых поставщиками универсальных СУБД, но при этом обеспечивали бы не меньшую, а



желательно, б*о*льшую производительность и масштабируемость при работе со сверхбольшими хранилищами данных.

### 1.1. Аналитические параллельные СУБД сегодня

Как отмечается в [2], в действительности направление DWAA появилось еще в 1980-е гг., и соответствующие пионерские продукты были созданы в компании Britton Lee Inc. [3], которая в 1989 г. была сначала переименована в ShareBase Corporation, а затем поглощена компанией Teradata [4], которая к этому времени тоже придерживалась подхода DWAA. Аппаратно-программное решение, основанное на ассоциативной адресации элементов хранения данных, имелось у компании ICL (Content Addressable File Store [5]). Однако на рынке систем поддержки хранилищ данных на основе подхода DWAA с тех пор осталась только Teradata.

Возрождение направления DWAA в начале 2000-х, безусловно, связано с упомянутым выше ростом заинтересованности компаний в сравнительно недорогих и эффективных решениях, направленных исключительно на поддержку хранилищ данных и их анализа. Вокруг этого направления стали возникать софтверные стартапы, первым из которых стала компания Netezza [6], основавшая свое эффективное DWAA-решение на использовании программируемых вентильных матриц (Field Programmable Gate Array, FPGA) и процессоров PowerPC. Использование FPGA в контроллерах магнитных дисков позволяет осуществлять "на лету" первичную фильтрацию данных, а применение PowerPC вместо процессоров Intel (по утверждению компании) позволяет снизить энергопотребление и расходы на охлаждение.

С тех пор появилось еще около десяти новых компаний, ориентирующихся на разработку DWAA с применением (почти всегда) разновидностей массивно-параллельной архитектуры (MPP) "sharing-nothing":

- Vertica Systems [7] – MPP, поколоночное хранение таблиц;
- DATAllegro Inc. [8], недавно поглощенная Microsoft, которая основала на продукте этой компании проект Madison, ставший основой SQL Server 2008 R2 Parallel Data Warehouse [15], – MPP, система основана на использовании СУБД Ingres [16] (тем самым, таблицы хранятся по строкам);
- Greenplum [9] – MPP, система основана на использовании СУБД PostgreSQL [17] (тем самым, таблицы хранятся по строкам);
- Aster Data Systems [10] – MPP, таблицы хранятся по строкам;
- Kognitio [11] – MPP, таблицы хранятся по строкам;
- EXASOL AG [12] – MPP, поколоночное хранение таблиц;
- Calpont Corporation [13] – MPP, поколоночное хранение таблиц, система (InfiniDB) внешне схожа с MySQL;
- Dataupia Corporation [14] – MPP, таблицы хранятся по строкам;

- Infobright [15] – поколоночное хранение таблиц, система основана на MySQL, ориентирована на использование многоядерных процессоров, массивный параллелизм не используется;
- Kickfire [16] – поколоночное хранение таблиц, используется специальная аппаратура, ускоряющая выполнение SQL-запросов, система создана на основе MySQL и не основана на массивно-параллельной архитектуре.

Подход DWAA постепенно проникает и в продукты основных поставщиков SQL-ориентированных СУБД. Как отмечалось выше, разработка компании DATAllegro стала основой массивно-параллельного варианта Microsoft SQL Server (SQL Server 2008 R2 Parallel Data Warehouse), а компания Oracle обеспечивает специализированное массивно-параллельное хранилище табличных данных Oracle Exadata Storage Server [21], позволяющее значительно ускорить работу основной СУБД.

У разных решений категории DWAA имеются свои интересные технические особенности, заслуживающие более глубокого обсуждения, анализа и сравнения. Их можно классифицировать и сравнивать по разным критериям. Однако это не является целью данной статьи. Некоторую попытку такого анализа представляет собой обзор [22]. Значительный рост интереса к направлению DWAA, к специализированным СУБД вообще и к СУБД Vertica в частности вызвала статья [23].

## 1.2. При чем здесь MapReduce?

В этой своей статье я сосредоточусь на частном, но очень важном в настоящее время вопросе взаимоотношений технологий массивно-параллельных аналитических СУБД и MapReduce [24]. При рассмотрении этого вопроса контекст DWAA является вполне естественным, поскольку практически все СУБД, созданные на основе подхода DWAA, являются массивно-параллельными без использования общих ресурсов. Эти системы создавались в расчете на использование в кластерной аппаратной архитектуре, и они сравнительно легко могут быть перенесены в "облачную" среду динамически конфигурируемых кластеров.

Поэтому появление "родной" для "облачной" среды технологии MapReduce и в особенности энтузиазм по части ее использования, проявленный многими потенциальными пользователями параллельных СУБД, очень озаботили представителей направления DWAA. Сначала авторитетные представители сообщества баз данных и одновременно активные сторонники подхода DWAA Майкл Стоунбрейкер (Michael Stonebraker) и Дэвид Девитт (David J. DeWitt) старались убедить общественность в том, что MapReduce – это технология, уступающая технологии параллельных баз данных по всем статьям [25-26]. Потом была проведена серия экспериментов, продемонстрировавшая, что при решении типичных простых аналитических задач MapReduce уступает в производительности не только поколоночной СУБД Vertica, но и



традиционной массивно-параллельной СУБД с хранением таблиц по строкам [27].

Все приводимые доводы и результаты экспериментов были весьма солидными и убедительными, и вряд ли кто-нибудь из людей, знакомых с обеими технологиями, сомневается в том, что MapReduce не вытеснит параллельные СУБД, и что эти технологии будут благополучно сосуществовать в "облаках" и в среде кластерных архитектур вообще. Однако возникает другой вопрос: а нет ли в технологии MapReduce каких-либо положительных черт, которых не хватает параллельным СУБД? И можно ли каким-либо образом добавить эти черты в параллельные СУБД, сохранив их основные качества: декларативный доступ на языке SQL, оптимизацию запросов и т.д. (Кстати, понятно, что у параллельных СУБД имеется масса положительных черт, которыми не обладает MapReduce, но похоже, что добавление их к MapReduce изменило бы суть этой технологии, превратив ее в технологию параллельных СУБД.)

И на эти два вопроса удалось получить положительный ответ. В нескольких проектах, связанных с направлением DWAA, удалось воспользоваться такими преимуществами MapReduce, как масштабируемость до десятков тысяч узлов, отказоустойчивость, дешевизна загрузки данных, возможность использования явно написанного кода, который хорошо распараллеливается. Сразу следует заметить, что пока ни в одном проекте не удалось воспользоваться сразу всеми этими преимуществами, но даже то, чего уже достигли исследователи и разработчики, позволяет добавить в параллельные СУБД важные качества, которыми они до сих по не обладали.

Мы рассмотрим три подхода к интеграции технологий MapReduce и параллельных СУБД, предложенных и реализованных специалистами компаний Greenplum [28] и Aster Data [29], университетов Yale и Brown [30], а также компании Vertica [31] соответственно, которые можно было бы назвать:

- MapReduce внутри параллельной СУБД;
- СУБД внутри среды MapReduce и
- MapReduce сбоку от параллельной СУБД.

В общих словах, первый подход ориентирован на поддержку написания и выполнения хранимых на стороне сервера баз данных пользовательских функций, которые хорошо распараллеливаются в кластерной (в том числе, в "облачной") среде. Т.е. в данном случае используется преимущество MapReduce по применению явно написанного кода и его распараллеливанию.

Второй подход направлен на использование MapReduce в качестве инфраструктуры параллельной СУБД, в качестве базовых компонентов которой используются традиционные не параллельные СУБД. Применение MapReduce позволяет добиться неограниченной масштабируемости получаемой системы и ее отказоустойчивости на уровне выполнения запросов.

Наконец, при применении третьего подхода MapReduce используется для выполнения процедуры ETL (Extract, Tansform, Load) над исходными данными до их загрузки в систему параллельных баз данных. В этом случае используется преимущество MapReduce в отношении дешевой загрузки данных до их обработки.

Основные разделы статьи организованы следующим образом. В разд. 2 приводится общий обзор технологии MapReduce. Разд. 3 посвящается обсуждению деталей интеграции технологий MapReduce и параллельных СУБД путем встраивания средств MapReduce в СУБД. В разд. 4 описываются основные приемы, используемые для создания параллельной СУБД на основе MapReduce. В разд. 5 обсуждаются проблемы загрузки данных в аналитические базы данных и преимущества, которые можно получить при выполнении процедуры ETL на основе MapReduce. Разд. 6 содержит заключение.

## *2. MapReduce: модель и реализации*

Программная модель MapReduce была придумана несколько лет тому назад в компании Google [24], и там же была выполнена первая реализация этой модели на основе распределенной файловой системы той же компании GFS (Google File System) [32]. Эта реализация активно используется в программных продуктах самой Google, но является сугубо проприетарной и недоступна для использования вне Google.

Альтернативная, свободно доступная реализация Hadoop MapReduce [33] (с открытыми исходными текстами) была выполнена в проекте Hadoop [34] сообщества Apache [35]. Она основана на использовании распределенной файловой системы HDFS (Hadoop Distributed File System) [36], также разработанной в проекте Hadoop. Реальную популярность MapReduce принесла именно реализация Hadoop в силу своей доступности и открытости, а широкое использование Hadoop MapReduce в различных исследовательских и исследовательских проектах приносит несомненную пользу этой системе, стимулируя разработчиков к ее постоянному совершенствованию.

Однако реализация Hadoop MapReduce полностью основана на спецификациях Google, и поэтому каноническим описанием технологии была и остается статья [24]. Заметим, что при этом в документации Hadoop MapReduce [37] используется терминология, несколько отличная от [24]. В этом разделе из уважения к первенству Google я буду использовать термины из [24], а в следующих разделах там, где будет иметься конкретно реализация Hadoop MapReduce, будет использоваться терминология Hadoop (это не должно привести к путанице).



### 2.1. Общая модель программирования MapReduce

В этой модели вычисления производятся над множествами входных пар "ключ-значение", и в результате каждого вычисления также производится некоторое множество результирующих пар "ключ-значение". Для представления вычислений в среде MapReduce используются две основные функции: *Map* и *Reduce*. Обе функции явно кодируются разрабочиками приложений в среде MapReduce.

Функция *Map* в цикле обрабатывает каждую пару из множества входных пар и производит множество промежуточных пар "ключ-значение". Среда MapReduce групирует все промежуточные значения с одним и тем же ключом *I* и передает их функции *Reduce*.

Функция *Reduce* получает значение ключа *I* и множество значений, связанных с этим ключом. В типичных ситуациях каждая группа обрабатывается (в цикле) таким образом, что в результате одного вызова функции образуется не более одного результирующего значения.

### 2.2. Реализация в распределенной среде

Реализации MapReduce от Google и Hadoop ориентированы на использование в кластерной распределенной среде со следующими основными характеристиками:

- узлы среды выполнения MR-приложений обычно представляют собой компьютеры общего назначения с операционной системой Linux;
- используется стандартное сетевое оборудование с адаптерами, расчитанными на скорости передачи в 100 мегабит в секунду или 1 гигабит в секунду, но средняя пропускная способность существенно ниже;
- кластер состоит из сотен или тысяч машин, так что вполне вероятны отказы отдельных узлов;
- для хранения данных используются недорогие дисковые устройства, подключенные напрямую к отдельным машинам;
- для управления данными, хранящимися на этих дисках, используется распределенная файловая система;
- пользователи представляют свои задания в систему планирования; каждое задание состоит из некоторого набора задач, которые отображаются планировщиком на некоторый набор узлов кластера.

#### *2.2.1. Выполнение MR-приложения*

Вызовы *Map* распределяются по нескольким узлам кластера путем разделения входных данных на *M* непересекающихся групп (split). Входные группы могут параллельно обрабатываться на разных машинах. Вызовы *Reduce* распределяются путем разделения пространства ключей промежуточных ключей на *R* частей с использованием некоторой функции разделения

(например, функции хэширования). Число разделов *R* и функция разделения задаются пользователем.

Выполнение MR-программы происходит следующим образом. Сначала среда MapReduce расщепляет входной файл на *M* частей, размер которых может задаваться пользователем. Затем сразу в нескольких узлах кластера запускается основная программа MapReduce. Один из экземпляров этой программы играет специальную роль и называется *распорядителем (master)*. Остальные экземпляры являются *исполнителями (worker)*, которым распорядитель назначает работу. Распорядитель должен назначить исполнителям для выполнения *M* задач *Map* и *R* задач *Reduce*.

Исполнитель, которому назначена задача *Map*, читает содержимое соответствующей группы, разбирает пары "ключ-значение" входных данных и передает каждую пару в определенную пользователем функцию *Map*. Промежуточные пары "ключ-значение", производимые функцией *Map*, буферизуются в основной памяти. Периодически буферизованные пары, разделяемые на *R* областей на основе функции разделения, записываются в локальную дисковую память исполнителя. Координаты этих сохраненных на диске буферизованных пар отсылаются распорядителю, который, в свою очередь, передает эти координаты исполнителям задачи *Reduce*. *I*-ый *Reduce*-исполнитель снабжается координатами всех *i*-ых областей буферизованных пар, произведенных всеми *M Map*-исполнителями.

После получения этих координат от распорядителя исполнитель задачи *Reduce* с использованием механизма удаленных вызовов процедур переписывает данные с локальных дисков исполнителей задачи *Map* в свою память или на локальный диск (в зависимости от объема данных). После переписи всех промежуточных данных выполняется их сортировка по значениям промежуточного ключа для образования групп с одинаковым значением ключа. Если объем промежуточных данных слишком велик для выполнения сортировки в основной памяти, используется внешняя сортировка.

Далее *Reduce*-исполнитель организует цикл по отсортированным промежуточным данным и для каждого уникального значения ключа вызывает пользовательскую функцию *Reduce* с передачей ей в качестве аргумента значения ключа и соответствующее множество значений. Результирующие пары функции *Reduce* добавляются в окончательный результирующий файл данного *Reduce*-исполнителя. После завершения всех задач *Map* и *Reduce* распорядитель активизирует программу пользователя, вызывавшую MapReduce.

После успешного завершения выполнения задания MapReduce результаты размещаются в *R* файлах распределенной файловой системы (имена этих результирующих файлов задаются пользователем). Обычно не требуется объединять их в один файл, потому что часто полученные файлы используются в качестве входных для запуска следующего MR-задания или в



каком-либо другом распределенном приложении, которое может получать входные данные из нескольких файлов.

### *2.2.2. Отказоустойчивость*

Поскольку технология MapReduce предназначена для обработки громадных объемов данных с использованием сотен и тысяч машин, в ней обязательна должна присутствовать устойчивость к отказам отдельных машин.

#### *2.2.2.1. Отказ исполнителя*

Распорядитель периодически посылает каждому исполнителю контрольные сообщения. Если некоторый исполнитель не отвечает на такое сообщение в течение некоторого установленного времени, распорядитель считает его вышедшим из строя. В этом случае все задачи *Map*, уже выполненные и еще выполнявшиеся этим исполнителем, переводятся в свое исходное состояние, и можно заново планировать их выполнение другими исполнителями. Аналогично распорядитель поступает со всеми задачами *Reduce*, выполнявшимися отказавшим исполнителем к моменту отказа.

Завершившиеся задачи *Map* выполняются повторно по той причине, что их результирующие пары сохранялись на локальном диске отказавшего исполнителя и поэтому недоступны в других узлах. Завершившиеся задачи *Reduce* повторно выполнять не требуется, поскольку их результирующие пары сохраняются в глобальной распределенной файловой системе. Если некоторая задача *Map* выполнялась исполнителем *A*, а потом выполняется исполнителем *B*, то об этом факте оповещаются все исполнители, выполняющие задачи *Reduce*. Любая задача *Reduce*, которая не успела прочитать данные, произведенные исполнителем *A*, после этого будет читать данные от исполнителя *B*.

#### *2.2.2.2. Отказ распорядителя*

В реализациях MapReduce от Google и Hadoop какая-либо репликация распорядителя не производится. Считается, что поскольку распорядитель выполняется только в одном узле кластера, его отказ маловероятен, и если он случается, то аварийно завершается все выполнение MapReduce. Однако в [24] отмечается, что несложно организовать периодический сброс в распределенную файловую систему всего состояния распорядителя, чтобы в случае отказа можно было запустить его новый экземпляр в другом узле с данной контрольной точки.

#### *2.2.2.3. Семантика при наличии отказов*

Если обеспечиваемые пользователями функции *Map* и *Reduce* являются детерминированными (т.е. всегда выдают одни и те же результаты при одинаковых входных данных), то при их выполнении в среде распределенной реализации MapReduce при любых условиях обеспечивает тот же результат,

как при последовательном выполнении всей программы при отсутствии каких-либо сбоев.

Это свойство обеспечивается за счет атомарности фиксации результатов задач *Map* и *Reduce*. Каждая выполняемая задача записывает свои результаты в частные временные файлы. Задача *Reduce* производит один такой файл, а задача *Map* – *R* файлов, по одной на каждую задачу *Reduce*. По завершении задачи *Map* исполнитель посылает распорядителю сообщение, в котором указываются имена *R* временных файлов. При получении такого сообщения распорядитель запоминает эти имена файлов в своих структурах данных. Повторные сообщения о завершении одной и той же задачи *Map* игнорируются.

При завершении задачи *Reduce* ее исполнитель атомарным образом переименовывает временный файл результатов в окончательный файл. Если одна и та же задача *Reduce* выполняется несколькими исполнителями, то для одного и того же окончательного файла будет выполнено несколько операций переименования. Если в используемой распределенной файловой системе операция переименования является атомарной, то в результате в файловой системе соохранятся результаты только какого-либо одного выполнения задачи *Reduce*.

### *2.2.3. Резервные задачи*

Чаще всего к увеличению общего времени выполнения задания MapReduce приводит наличие "отстающих" ("straggler") – узлов кластера, в которых выполнение одной из последних задач *Map* или *Reduce* занимает необычно долгое время (например, из-за некритичной неисправности дискового устройства).

Для смягчения проблемы "остающих" в MapReduce применяется следующий общий механизм. Когда задание близится к завершению, для всех еще не завершившихся задач назначаются дополнительные, резервные исполнители. Задача считается выполненной, когда завершается ее первичное или резервное выполнение. Этот мезанизм настраивается таким образом, чтобы потребление вычислительных ресурсов возрастало не более чем на несколько процентов. В результате удается существенно сократить время выполнения крупных MR-заданий.

## 2.3. Расширенные средства

В большинстве приложений MapReduce оказывается достаточно просто написать свои функции *Map* и *Reduce*. Однако в ряде случаев оказываются полезными некоторые расширения базовых функциональных возможностей.

### *2.3.1. Функция разделения*

Пользователи MapReduce явно указывают требуемое число задач *Reduce R* (и соответствующих результирующих файлов). Данные распределяются между



этими задачами с использованием некоторой функции разделения от значений промежуточного ключа. По умолчанию используется функция хэширования (например, "*hash*(*key*) mod *R*"). Однако пользователи MapReduce могут специфицировать и собственные функции разделения.

### *2.3.2. Гарантии упорядоченности*

Внутри каждого раздела, передаваемого задаче *Reduce*, данные упорядочены по возрастанию значений промежуточного ключа. Это позволяет задачам *Reduce* производить отсортированные результирующие файлы.

### *2.3.3. Функция-комбинатор*

В некоторых случаях в результатах задачи *Map* содержится значительное число повторящихся значений промежуточного ключа, а определенная пользователем задача *Reduce* является коммутативной и ассоциативной. В таких случаях пользователь может определить дополнительную функцию-комбинатор (*Combiner*), выполняющую частичную агрегацию таких данных до их передачи по сети. Функция *Combiner* выполняется на той же машине, что и задача *Map*. Обычно для ее реализации используется тот же самый код, что и для реализации функции *Reduce*. Елинственное различие между функциями *Combiner* и *Reduce* состоит в способе работы с их результирующими данными. Результаты функции *Reduce* записываются в окончательный файл результатов. Результаты же функции *Combiner* помещаются в промежуточные файлы, которые впоследствии пересылаются в задачи *Reduce*.

### *2.3.4. Форматы входных и результирующих данных*

В библиотеке MapReduce поддерживается возможность чтения входных данных в нескольких разных форматах. Например, в режиме "text" каждая строка трактуется как пара "ключ-значение", где ключ – это смещение до данной строки от начала файла, а значение – содержимое строки. В другом распространенном формате входные данные представляются в виде пар "ключ-значение", отсортированных по значениям ключа. В каждой реализации формата входных данных известно, каким образом следует расшеплять данные на осмысленные части, которые обрабатываются отдельными задачами *Map* (например, данные формата "text" расщепляются только по границами строк).

Пользователи могут добавить к реализации собственные форматы входных данных, обеспечив новую реализацию интерфейса *reader* (в реализации Hadoop – *RecordReader*). *Reader* не обязательно должен читать данные из файла, можно легко определить *reader*, читающий данные из базы данных или из некоторой структуры в виртуальной памяти.

Аналогичным образом, поддерживаются возможности генерации данных в разных форматах, и имеется простая возможность определения новых форматов результирующих данных.

Думаю, что для общего ознакомления с технологией MapReduce и для понимания следующих разделов статьи этой информации достаточно. Кроме того, она позволяет понять, какие особенности модели и реализаций MapReduce обеспечивают масштабируемость технологии до десятков тысяч узлов, ее отказоустойчивость, дешевизну загрузки данных и возможность использования явно написанного кода, который хорошо распараллеливается.

## *3. MapReduce внутри параллельной СУБД*

Очевидны преимущества клиент-серверных организаций СУБД: в такой архитектуре сервер баз данных поддерживает крупную базу данных, которая сохраняется в одном экземпляре и доступна большому числу приложений, выполняемых прямо на стороне клиентов или в промежуточных серверах приложений. Однако даже при использовании реляционной (или, правильнее, SQL-ориентированной) организации баз данных, когда от клиентов на сервер баз данных отправляются высокоуровневые декларативные запросы, в обратную сторону, от сервера к клиенту, пересылаются результирующие данные, вообще говоря, произвольно большого объема.

Естественно, возникает вопрос: не окажется ли дешевле, чем пересылать данные с сервера на клиент для их дальнейшей обработки, переместить требуемую обработку данных на сервер, ближе к самим данным. Насколько мне известно, в явном виде идея перемещения вычислений на сторону сервера была высказана в статье Лоуренса Роува (Lawrence A. Rowe) и Майкла Стоунбрейкера (Michael R. Stonebraker) [38], хотя в более скрытой форме намеки на эту идею можно найти и в более ранних статьях М. Стоунбрейкера и др. [39-40], еще не имевших непосредственного отношения к СУБД Postgres.

С того времени, как поддержка определяемых пользователями хранимых процедур, функций и методов, типов данных и триггеров появилась во всех развитых SQL-ориентированных СУБД, соответствующие языковые средства специфицированы в стандарте языка SQL. Более того, возникла новая проблема выбора – одну и ту же функциональность приложения можно реализовать на стороне сервера, на сервере приложений и на клиенте. Однозначных методологий и рекомендаций, способствующих простому выбору, не существует. Например, очевидно, что если услугами одного сервера пользуется несколько приложений, то перегрузка сервера хранимыми процедурами и функциями, реализующими функциональность одного приложения, может нанести ущерб эффективности других приложений.

Тем не менее, во всех традиционных серверных организациях СУБД возможность переноса вычислений на сторону сервера существует и не очень сложно реализуется. Однако в параллельных СУБД (в особенности, категории



sharing-nothing) дела обстоят гораздо хуже. Выполнение SQL-запросов распараллеливается автоматически оптимизитором запросов. Но оптимизатор запросов не может распараллелить определенную пользователем процедуру или функцию, если она написана не на SQL, а на одном из традиционных языков программирования (обычно с включением вызовов операторов SQL).

Конечно, технически можно было бы такие процедуры и функции вообще не распараллеливать, а выполнять в каком-либо одном узле кластера. Но тогда (a) в этом узле пришлось бы собрать все данные, требуемые для выполнения процедуры или функции, для чего потребовалась бы массовая пересылка данных по сети, и (b) это свело бы на нет все преимущества параллельных СУБД, производительность которых основывается именно на параллельном выполнении.

С другой стороны, невозможно обязать распараллеливать свои программы самих пользователей, определяющих хранимые процедуры или функции (например, на основе библиотеки MPI [41], которую принято использовать для явного параллельного программирования на основе обмена сообщениями в массивно-параллельной среде). Во-первых, это слишком сложное занятие для разработчиков приложений баз данных, которые часто вообще не являются профессиональными программистами. Во-вторых, при таком явном параллельном программировании требовалось бы явным же образом управлять распределением данных по узлам кластера.

Несмотря на эти трудности, какая-то поддержка механизма распараллеливаемых определяемых пользователями процедур и функций в параллельных аналитических системам баз данных все-таки требуется, поскольку без этого аналитики вынуждены выполнять анализ данных на клиентских рабочих станциях, постоянно пересылая на них из центрального хранилища данных данные весьма большого объема. Другого способа работы у них просто нет. И как показывает опыт двух производственных разработок, для обеспечения возможностей серверного программирования в массивно-параллельной среде систем баз данных с пользой может быть применена модель MapReduce.

Речь идет о параллельных аналитических СУБД Greenplum Database компании Greenplum [9] и *n*Cluster компании Aster Data Systems [10]. Общим в подходах обеих компаний является то, что модель MapReduce реализуется внутри СУБД, и возможностями этих реализаций могут пользоваться разработчики аналитических приложений. Различие состоит в том, как можно пользоваться возможностями MapReduce: в Greenplum Database – наряду с SQL, а в *n*Cluster – из SQL. Рассмотрим эти подходы подробнее.

### 3.1. Greenplum – MapReduce наравне с SQL

Сначала немного поговорим об общей философии компании Greenplum, приведшей ее, в частности, к идее поддержки технологии MapReduce наряду с технологией SQL. По мнению идеологов Greenplum и основных архитекторов

Greenplum Database [28], возрастающий уровень востребованности хранилищ данных и оперативного анализа данных, возможность и целесообразность использования требуемых аппаратных средств в масштабах отдельных подразделений компаний приводят к потребности пересмотра "ортодоксального" подхода к организации хранилищ данных.

### *3.1.1. MAD Skills: новый подход к организации хранилищ данных и аналитике*

Предлагается и реализуется новый подход к анализу данных, который идеологи (и маркетологи!) компании связывают с аббревиатурой *MAD*. Здесь, конечно, имеется интересная игра слов, которую трудно выразить на русском языке. С одной стороны, *mad* применительно к технологии означает, что эта технология слегка безумна и уж во всяком случае не ортодоксальна. С другой стороны, *mad skills* означает *блестящие способности*, а значит, предлагаемая технология, по мнению ее творцов, обладает новыми полезнейшими качествами. Но в Greenplum *MAD* – это еще и аббревиатура от *magnetic*, *agile* и *deep*.

**Magnetic (магнетичность)** применительно к хранилищу данных означает, что оно должно быть "притягательным" по отношению к новым источникам данных, появляющимся в организации. Данные из новых источников должны легко и просто включаться в хранилище данных с пользой для аналитиков. В отличие от этого, при использовании традиционного ("ортодоксального") подхода к организации хранилища данных, для подключения нового источника данных требуется разработка и применение соответствующей процедуры ETL, а возможно, и изменение схемы хранилища данных, в результате чего подключение нового источника данных часто затягивается на месяца, а иногда и вовсе кончается ничем.

**Agile (гибкость)** – это предоставляемая аналитикам возможность простым образом и в быстром темпе воспринимать, классифицировать, производить и перерабатывать данные. Для этого требуется база данных, логическая и физическая структура и содержание которой могут постоянно и быстро изменяться. В отличие от этого, традиционным хранилищам данных свойственна жесткость, связанная с потребностью долгосрочного тщательного проектирования и планирования.

**Deep (основательность)** означает, что аналитикам должны предоставляться средства выполнения произвольно сложных статистических алгоритмов над всеми данными, находящимися в хранилище данных, без потребности во взятии образцов или выборок. Хранилище данных должно служить как основательным репозиторием данных, так и средой, поддерживающей выполнение сложных алгоритмов.

Я не буду больше распространяться здесь про MAD-аналитику (более развернутое обсуждение см. в [28], а остановлюсь только на том аспекте, который привел к реализации системы с поддержкой интерфейсов и SQL, и



MapReduce. Как считают разработчики Greenplum Database хозяевами будущего мира анализа данных должны стать аналитики. Фактически, на это направлены все аспекты MAD-аналитики. В частности, это означает всяческую поддержку написания и использования в среде хранилища данных разнообразных аналитических алгоритмов.

Как отмечалось в подразделе 1.1, параллельная СУБД Greenplum Database делалась на основе СУБД PostgreSQL, являющейся законной наследницей Postgres. Помимо своих прочих достоинств, Postgres была первой *расширяемой* СУБД (этот аспект системы впервые явно подчеркивался в [42]). Пользователи Postgres могли определять собственные процедуры и функции, типы данных и даже методы доступа к структурам внешней памяти. Эти возможности расширений системы были переняты и развиты в PostgreSQL. Наряду с традиционным в Postgres языком C, для программирования серверных расширений в PostgreSQL можно использовать, в частности, популярные скриптовые языки Perl и Python [43].

В Greenplum Database на основе этих возможностей расширений системы обеспечена расширенная среда, позволяющая на уровне языка SQL оперировать такими математическими объектами, как векторы, функции и функционалы. Пользователи могут определять собственные статистические алгоритмы и в полуавтоматическом режиме распараллеливать их выполнение по данным в массивно-параллельной среде (что часто является очень нетривиальной задачей). Однако в любом случае при использовании такого подхода к анализу данных пользователям-аналитикам приходится иметь дело с декларативным языком SQL, а как считают идеологи Greenplum, для многих аналитиков и статистиков SQL-программирование является обременительным и неудобным.

В качестве альтернативы аналитическому SQL-программированию в Greenplum Database обеспечивается полноправная реализация MapReduce, в которой обеспечивается доступ ко всем данным, сохраняемым в хранилище данных. При использовании MapReduce аналитики пишут собственный понятный для них процедурный код (можно использовать те же Perl и Python) и понимают, как будет выполняться их алгоритм в массивно-параллельной среде, поскольку это выполнение опирается на простую модель MapReduce.

### *3.1.2. Реализация MapReduce в Greenplum Database*

Для иллюстрации общей организации Greenplum Database воспользуемся
Рис. 1, позаимствованным из [44].

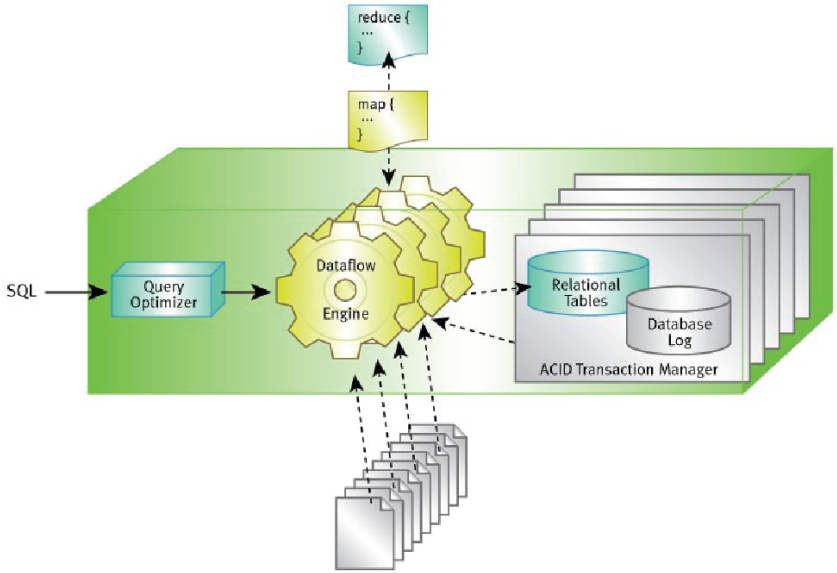


*Рис. 1. Общая организация Greenplum Database*

Как показывает этот рисунок, ядром системы является процессор потоков данных (Dataflow Engine). С его разработки началась реализация Greenplum Database, причем основные цели состояли в том, чтобы (a) заменить соответствующий компонент ядра PostgreSQL, чтобы обеспечить массивно-параллельное выполнение запросов и (b) обеспечить базовые функциональные возможности, требуемые для поддержки модели MapReduce. В результате SQL-ориентированная СУБД и MapReduce работают с общим ядром, поддерживающим массивно-параллельную обработку данных, и механизмы SQL и MapReduce обладают интероперабельностью.

Как отмечалось выше, функции *Map* и *Reduce* в среде Greenplum Database можно программировать на популярных скриптовых языках Python и Perl. В результате оказывается возможным использовать развитые программные средства с открытыми кодами, содержащиеся в репозиториях Python Package Index (PyPi) [45] и Comprehensive Perl Archive Network (CPAN) [46]. В составе этих репозиториев находятся средства анализа неструктурированного текста, статистические инструментальные средства, анализаторы форматов HTML и XML и многие другие программные средства, потенциально полезные аналитикам.

В среде Greenplum Database приложениям MapReduce обеспечивается доступ к данным, хранящимся в файлах, предоставляемым Web-сайтами и даже



генерируемым командами операционной системы. Доступ к таким данным не влечет накладных расходов, ассоциируемых с использованием СУБД: блокировок, журнализации, фиксации транзакций и т.д. С другой стороны, эффективный доступ к данным, хранимым в базе данных, поддерживается за счет выполнения MR-программ в ядре Greenplum Database. Это позволяет избежать расходов на пересылку данных.

Архитектура Greenplum Database с равноправной поддержкой SQL и MapReduce позволяет смешивать стили программирования, делать MR-программы видимыми для SQL-запросов и наоборот. Например, можно выполнять MR-программы над таблицами базы данных. Для этого всего лишь требуется указать MapReduce, что входные данные программы должны браться из таблицы. Поскольку таблицы баз данных Greenplum Database хранятся разделенными между несколькими узлами кластера, первая фаза *MAP* выполняется внутри ядра СУБД прямо над этими разделами.

Как и в автономных реализациях MapReduce, результаты выполнения MR-программ могут сохраняться в файловой системе. Но настолько же просто сохранить результирующие данные в базе данных с обеспечением транзакционной долговечности хранения этих данных (см. компонент ACID Transaction Manager на Рис. 1). В дальнейшем эти данные могут анализироваться, например, с применением SQL-запросов. Запись результирующих данных в таблицы происходит параллельным образом и не вызывает лишних накладных расходов.

Поскольку источником входных данных для MR-программы может служить любая таблица базы данных Greenplum, в частности, в качестве такой таблицы можно использовать представление базы данных, определенное средствами SQL. И наоборот, MR-программу можно зарегистрировать в базе данных как представление, к которому можно адресовать SQL-запросы. В этом случае MR-задание выполняется "на лету" во время обработки SQL-запроса, и результирующие данные в конвейерном режиме передаются прямо в план выполнения запроса.

## 3.2. Aster Data – MapReduce как основа нового механизма функций, определяемых пользователями

Как и у компании Greenplum с ее *MAD Skills*, у компании Aster Data имеется свой слоган *Big Data, Fast Insight*, который, по сути, означает то же самое – превращение массивно-параллельного хранилища данных в аналитическую платформу. И для этого тоже используется технология MapReduce, встроенная в СУБД. Однако, в отличие от Greenplum, эта технология применяется не для обеспечения альтернативного внешнего способа обработки данных, а для реализации нового механизма хорошо распараллеливаемых (по модели MapReduce), самоопределяемых и полиморфных табличных функций, определяемых пользователями и вызываемых из операторов выборки SQL [30].

### 3.2.1. Предпосылки и преимущества использования механизма SQL/MapReduce

По мнению основных разработчиков СУБД *n*Cluster, декларативный язык SQL во многом ограничивает использование аналитических СУБД. С одной стороны, несмотря на постоянное наращивание аналитических возможностей этого языка, для многих аналитиков их оказывается недостаточно. С другой стороны, эти возможности постепенно становятся такими сложными и непонятными, что зачастую становится проще написать процедурный код, решающий частную аналитическую задачу. Наконец, оптимизаторы запросов SQL-ориентированных СУБД постоянно отстают от развития языка, и планы сложных аналитических запросов могут быть весьма далеки от оптимальных, что приводит к их недопустимо долгому выполнению, а иногда и аварийному завершению.

Эти проблемы частично решаются за счет поддержки в SQL-ориентированных СУБД механизма функций, определяемых пользователями (User-Defined Function, UDF). Такие функции позволяют пользователям решать внутри сервера баз данных свои прикладные задачи путем написания соответствующего процедурного кода. Однако традиционные механизмы UDF разрабатывались в расчете на "одноузловые" СУБД, и по умолчанию предполагается чисто последовательное выполнение UDF. Как упоминалось в этом разделе ранее, автоматическое распараллеливание последовательного кода в массивно-параллельной среде с разделением данных является сложной нерешенной проблемой.

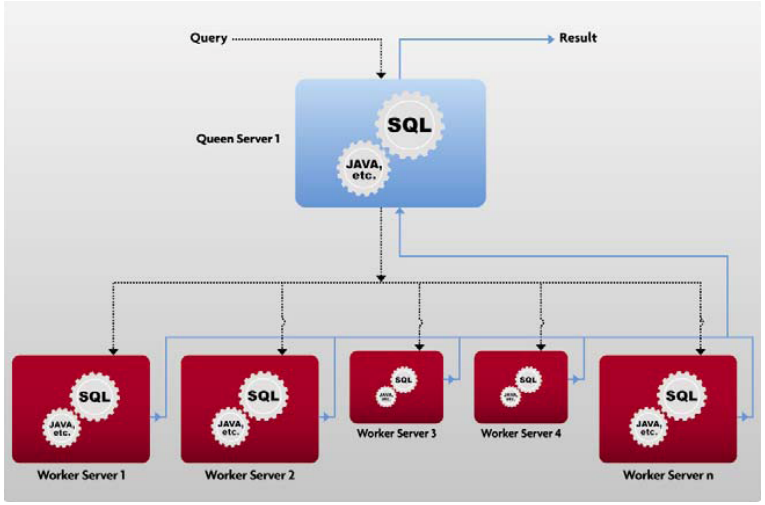


*Рис. 2. Совместное выполнение SQL-запросов и SQL/MapReduce-функций в nCluster*



В Aster Data для обеспечения механизма естественно распараллеливаемых UDF разработана инфраструктура SQL/MapReduce, поддерживаемая внутри SQL-ориентированной массивно-параллельной СУБД *n*Cluster (см. Рис. 2, позаимствованный из [47]). Организация среды SQL/MapReduce обеспечивает следующие возможности:

- можно эффективно выполнять в "реляционном" стиле операции над таблицами с использованием SQL, а "нереляционные" задачи и оптимизации – возлагать на явно программируемые процедурные функции;
- поскольку функции выполняются над согласованными данными из таблиц базы данных, обеспечивается согласованность вычислений;
- оценочный (cost-based) оптимизатор может принимать решения о способе выполнения SQL-запросов, содержащих вызовы SQL/MapReduce-функций, на основе достоверной статистики данных;
- пользователи *n*Cluster могут формулировать SQL-запросы с использованием высокоуровневых средств анализа данных, воплощенных в SQL/MapReduce-функциях.

SQL/MapReduce-функции можно программировать как на традиционных языках программирования (Java, C#, C++), так и скриптовых языках (Python, Ruby). При этом, независимо от используемого языка программирования, эти функции являются *самоописываемыми* и *полиморфными*. Поскольку здесь идет речь о *табличных* функциях (т.е. функциях, входными параметрами и результатами которых являются таблицы), то это означает, что одна и та же функция может принимать на вход таблицы с разными схемами (функция настраивается на конкретную схему входной таблицы на этапе формирования плана запроса, содержащего ее вызов) и выдавать таблицы также с разными схемами (функция сама сообщает планировщику запроса схему своего результата на этапе формирования плана запроса). Это свойство SQL/MapReduce-функций упрощает процедуру их регистрации в системе (в частности, не требуется выполнение специального оператора SQL `CREATE FUNCTION`, в котором описывались бы схемы входной и выходной таблиц) и способствует повторному использованию кода.

Синтаксические особенности определения SQL/MapReduce-функций и их семантика делают эти программные объекты естественным образом параллелизуемыми по данным: во время выполнения для каждой функции образуются ее экземпляры, параллельно выполняемые в узлах, которые содержат требуемые данные. Вызовы функций подобны подзапросам SQL, что обеспечивает возможность композиции функций, при которой при вызове функции вместо спецификации входной таблицы можно задавать вызов другой функции. Наконец, внешняя эквивалентность вызова SQL/MapReduce-функции подзапросу позволяет применять при формировании плана SQL-запроса с вызовами таких функций обычную оценочную оптимизацию на

основе статистики, а также динамически изменять порядок выполнения функций и вычисления настоящих SQL-подзапросов.

В следующем пункте без излишних деталей обсуждаются синтаксические конструкции, семантика и особенности реализации, обеспечивающие отмеченные свойства SQL/MapReduce-функций (более подробное описание см. в [30]).

### *3.2.2. Синтаксис, семантика и особенности реализации SQL/MapReduce*

Обсудим, как вызываются SQL/MapReduce-функции, как они определяются, и как обрабатываются SQL-запросы с вызовами таких функций.

#### *3.2.2.1. Синтаксис вызова SQL/MapReduce-функции*

Вызов SQL/MapReduce-функции может присутствовать только в качестве элемента списка ссылок на таблицы раздела `FROM` SQL-запроса. Синтаксис вызова показан на Рис. 3.

```
SELECT ...
FROM function_name(
  ON table-or-query
  [PARTITION BY expr, ...]
  [ORDER BY expr, ...]
  [clausename(arg, ...) ...]
  )
...
```

*Рис. 3. Синтаксис вызова SQL/MapReduce-функции*

В разделе `ON`, который является единственным обязательным разделом вызова, указывается любой допустимый SQL/MapReduce-запрос (SQL-запрос, вызов SQL/MapReduce-функции или просто имя таблицы). Во время формирования плана запроса, содержащего вызов SQL/MapReduce-функции, схемой входной таблицы этого вызова считается схема результата запроса, указанного в разделе `ON`.

Раздел `PARTITION BY` указывается только в вызовах SQL/MapReduce-функций над разделами (аналоге функции *Reduce* исходной модели MapReduce, см. ниже). В этом случае в разделе `PARTITION BY` указывается список выражений, на основе значений которых производится разделение таблицы, специфицированной в разделе `ON`. При наличии раздела `PARTITION BY` в вызове может содержаться и раздел `ORDER BY`, указывающий на потребность в сортировке входных данных до реального вызова функции. Наконец, вслед за разделом `ORDER BY` можно указать произвольное число дополнительных разделов со специальными аргументами. Имена этих



разделов и значения аргументов передаются в SQL/MapReduce-функцию при ее инициализации.

#### *3.2.2.2. Модель выполнения SQL/MapReduce-функций*

В среде SQL/MapReduce используется модель выполнения функций, являющаяся обобщением модели MapReduce. Функция SQL/MapReduce может быть либо *функцией над строками (row function)*, либо *функцией над разделами (partition function)*. Функции первого типа является аналогами функций *Map* классической модели MapReduce, а функции второго типа – аналогами функций *Reduce*. Поскольку, как отмечалось ранее, в разделе `ON` вызова SQL/MapReduce-функции может содержаться вызов другой SQL/MapReduce-функции, в среде SQL/MapReduce допускается любое число и любой порядок вызовов функций *Map* и *Reduce*, а не только жесткая последовательность *Map*-*Reduce*, допускаемая классической моделью.

При выполнении функции над строками каждая строка входной таблицы обрабатывается ровно одним экземпляром этой функции. С точки зрения семантики каждая строка обрабатывается независимо, поэтому входная таблица может разделяться по экземплярам функции произвольным образом, что обеспечивает возможности параллелизма и масштабирования. Для каждой строки входной таблицы функция над строками может не производить ни одной строки, а может произвести несколько строк.

При выполнении функции над разделами каждая группа строк, образованная на основе спецификации раздела `PARTITION BY` вызова функции, обрабатывается ровно одним экземпляром этой функции, и этот экземпляр получает все группу целиком. Если в вызове функции содержится раздел `ORDER BY`, то экземпляры функции получают разделы в уже упорядоченном виде. С точки зрения семантики каждая строка обрабатывается независимо, что обеспечивает возможности параллелизма на уровне разделов. Для каждого входного раздела функция над строками может не производить ни одной строки, а может произвести несколько строк.

#### *3.2.2.3. Реализация SQL/MapReduce-функций*

Как отмечалось в предыдущем пункте, для реализации SQL/MapReduce-функций можно использовать разные языки, но все они являются объектно-ориентированными. Каждая SQL/MapReduce-функция реализуется в виде отдельного класса, и при выработке плана выполнения SQL-запроса, содержащего вызовы таких функций, для каждого вызова образуется объект соответствующего класса с обращением к его методу-конструктору (инициализатору функции). Это обеспечивает настройку функции и получение требуемого описания ее результирующей таблицы.

Более точно, взаимодействие оптимизатора запросов с инициализатором функции производится через специальный объект, называемый *контрактом времени выполнения (Runtime Contract)*. Анализируя вызов функции,

оптимизатор выявляет имена и типы данных столбцов входной таблицы, а также имена и значения разделов дополнительных параметров и соответствующим образом заполняет некоторые поля объекта-контракта, который затем передается инициализатору функции. Инициализатор завершает подготовку контракта путем заполнения его дополнительных полей, содержащих, в частности, информацию о схеме результирующей таблицы, и обращается к методу `complete` объекта-контракта. На основе готового контракта продолжается выработка плана выполнения запроса, и этот контракт соблюдается при последующем выполнении SQL/MapReduce-функции всеми ее экземплярами.

Наиболее важными методами интерфейсов классов для функций над строками и разделами являются методы `OperateOnSomeRows` и `OperateOnPartition`. При обращении к этим методам (реальном выполнении соответствующей функции) в качестве аргументов передаются *итератор над строками*, для обработки которых вызывается функция, и объект *emitter*, с помощью вызовов которого возвращаются результирующие строки.

Чтобы можно было начать использовать некоторую SQL/MapReduce-функцию, ее нужно инсталлировать. Для этого используется общий механизм инсталляции файлов, реализованный в *n*Cluster. Этот механизм реплицирует файл во всех рабочих узлах системы. Далее проверяется, что этот файл содержит SQL/MapReduce-функцию, а также выясняются ее статические свойства: является ли она функцией на строками или же над разделами, содержит ли она вызовы комбинатора и т.д.

Таким образом в этих двух системах обеспечивается возможность развитого анализа данных поблизости от самих данных. Разработчики серверных аналитических приложений несколько ограничиваются моделью MapReduce (в большей степени в Greenplum Database, в меньшей – в *n*Cluster), но зато пользовательский процедурный код хорошо распараллеливается по данным в массивно-параллельной среде.

## *4. Параллельная СУБД на основе MapReduce*

Начну этот раздел с того, что в одной из первых серьезных статей, посвященных сравнению эффективности технологий MapReduce и массивно-параллельных СУБД при решении аналитических задач [27], утверждалось, что развитость и зрелость технологии параллельных СУБД категории sharing-nothing позволяет им обходиться стоузловыми кластерами для поддержки самых крупных сегодняшних аналитических баз данных петабайтного масштаба. Вместе с тем, особые качества масштабируемости и отказоустойчивости технологии MapReduce проявляются при использовании кластеров с тысячами узлов. Из этого делался вывод, что в обозримом



будущем эти качества параллельным СУБД не то чтобы не требуются, но, во всяком случае, не являются для них настоятельно необходимыми.

Однако спустя всего несколько месяцев появилась статья [30], в которой звучат уже совсем другие мотивы (и это при том, что авторские коллективы [27] и [30] значительно пересекаются). В [30] говорится, что в связи с ростом объема данных, которые требуется анализировать, возрастает и число приложений, для поддержки которых нужны кластеры с числом узлов, больше ста. В то же время, имеющиеся в настоящее время параллельные СУБД не масштабируются должным образом до сотен узлов. Это объясняется следующими причинами.

- При возрастании числа узлов кластера возрастает вероятность отказов отдельных узлов, а массивно-параллельные СУБД проектировались в расчете на редкие отказы.
- Современные параллельные СУБД расчитаны на однородную аппаратную среду (все узлы кластера обладают одной и той же производительностью), а при значительном масштабировании полной однородности среды добиться почти невозможно.
- До последнего времени имелось очень небольшое число систем аналитических баз данных, для достижения требуемой производительности которых требовались кластеры с более чем несколькими десятками узлов. Поэтому существующие параллельные СУБД просто не тестировались в более масштабной среде, и при их дальнейшем масштабировании могут встретиться непредвиденные технические проблемы.

Требуемые характеристики масштабируемости и отказоустойчивости может обеспечить технология MapReduce, поскольку она с самого начала разрабатывалась с расчетом на масштабирование до тысяч узлов, и ее реализация от Google эффективно используется для поддержки внутренних операций этой компании. Несмотря на то, что изначально технология MapReduce ориентировалась на обработку неструктурированных текстовых данных, известны показательные примеры ее использования и для обработки огромных объемов структурированных данных.

Однако объективно при обработке структурированных данных MapReduce не может конкурировать с параллельными СУБД по производительности, что объясняется отсутствием схемы у обрабатываемых данных, индексов, оптимизации запросов и т.д. В результате при выполнении многих типичных аналитических запросов MapReduce показывает производительность, более чем на порядок уступающую производительности параллельных СУБД [27, 31].

В проекте HadoopDB [48] специалисты из университетов Yale и Brown предпринимают попытку создать гибридную систему управления данными, сочетающую преимущества технологий и MapReduce, и параллельных СУБД. В их подходе MapReduce обеспечивает коммуникационную инфраструктуру,

объединяющую произвольное число узлов, в которых выполняются экземпляры традиционной СУБД. Запросы формулируются на языке SQL, транслируются в среду MapReduce, и значительная часть работы передается в экземпляры СУБД. Наличие MapReduce обеспечивается масштабируемость и отказоустойчивость, а использование в узлах кластера СУБД позволяет добиться высокой производительности.

## 4.1. Общая организация HadoopDB

Архитектура HadoopDB показана на Рис. 4, позаимствованном из [49]. Основой системы является реализация MapReduce, выполненная в проекте Hadoop [33]. К ней добавлены компоненты компиляции поступающих в систему SQL-запросов, загрузки и каталогизирования данных, связи с СУБД и самих СУБД. При реализации всех компонентов системы максимально использовались пригодные для этого программные средства с открытыми исходными текстами.

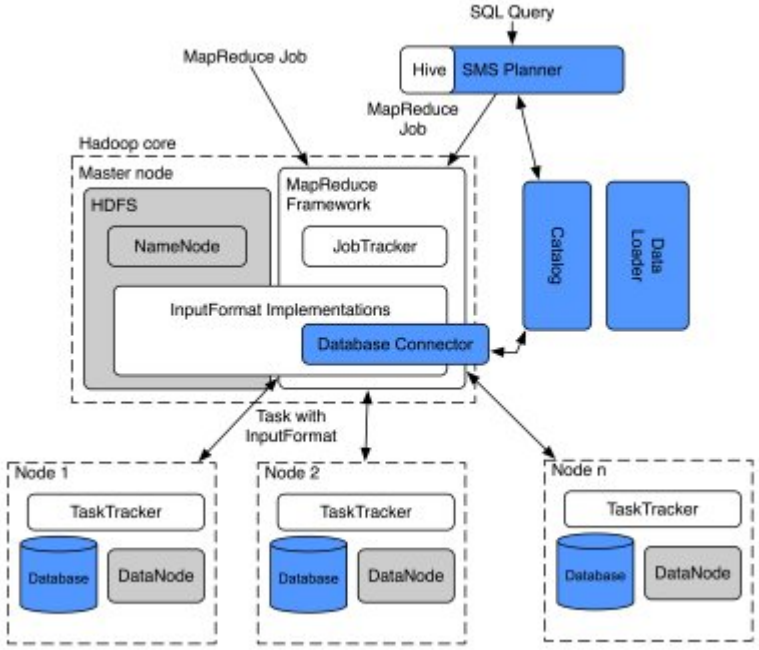


*Рис. 4. Архитектура HadoopDB*

### *4.1.1. Немного про Hadoop MapReduce*

Как отмечалось в разделе 2, реализация Hadoop MapReduce основана на спецификациях Google, содержащихся в [24]. Однако в этом проекте



используется собственная терминология, и для простоты описания особенностей HadoopDB в этом пункте кратко описывается организация Hadoop MapReduce в терминах Hadoop.

Hadoop MapReduce опирается на распределенную файловую систему HDFS (Hadoop Distributed File System) [36]. Файлы HDFS имеют блочную структуру, и блоки одного файла распределяются по узлам данных (DataNode). Файловая система работает под централизованным управлением выделенного узла имен (NameNode), в котором поддерживаются метаданные о файлах (в том числе, об их размерах, о размещении блоков и их реплик и т.д.).

В самой среде Hadoop MapReduce в соответствии с [24] поддерживаются один узел-распорядитель (в Hadoop он называется JobTracker) и много узлов-исполнителей (здесь TaskTracker). В узле JobTracker планируется выполнение MR-заданий, а также отслеживаются данные о загрузке узлов TaskTracker и доступных ресурсах. Каждое задание разбивается на задачи *Map* и *Reduce*, которые назначаются узлом JobTracker узлам TaskTracker с учетом требований локальности данных и балансировки нагрузки.

Требование локальности удовлетворяется за счет того, что JobTracker пытается назначать каждую задачу *Map* тому узлу TaskTracker, для которого данные, обрабатываемые этой задачей, являются локальными. Балансировка нагрузки достигается путем назначения задач всем доступным узлам TaskTracker. Узлы TaskTracker периодически посылают в узел JobTracker контрольные сообщения с информацией о своем состоянии.

Для обеспечения доступа к входным данным MR-задания поддерживается библиотека InputFormat. В Hadoop MapReduce имеется несколько реализаций этой библиотеки, одна из которых позволяет всем задачам одного MR-задания обращаться к JDBC-совместимой базе данных.

### *4.1.2. Собственные компоненты HadoopDB*

Как показывает рис. 4, в архитектуре HadoopDB присутствует ряд компонентов, расширяющих среду Hadoop MapReduce.

#### *4.1.2.1. Коннектор баз данных (Database Connector)*

Коннектор баз данных обеспечивает интерфейс между TaskTracker и независимыми СУБД, располагаемыми в узлах кластера. Этот компонент расширяет класс InputFormat и является частью соответствующей библиотеки. От каждого MR-задания в коннектор поступает SQL-запрос, а также параметры подключения к системе баз данных (указание драйвера JDBC, размер структуры выборки данных и т.д.).

Теоретически коннектор обеспечивает подключение к любой JDBC-совместимой СУБД. Однако в других компонентах HadoopDB приходится учитывать специфику конкретных СУБД, поскольку для них требуется по-разному оптимизировать запросы. В экспериментах, описываемых в [30],

использовалась реализация коннектора для PostgreSQL, а в [49] уже упоминается некоторая поколоночная система. В любом случае, для среды HadoopDB эта реализация обеспечивает естественное и прозрачное использование баз данных в качестве источника входных данных.

#### *4.1.2.2. Каталог*

В каталоге поддерживаются метаданные двух сортов: параметры подключения к базе данных (ее месторасположение, класс JDBC-драйвера, учетные данные) и описание наборов данных, содержащихся в кластере, расположение реплик и т.д. Каталог сохраняется в формате XML в HDFS. К нему обращаются JobTracker и TaskTracker для выборки данных, требуемых для планирования задач и обработки данных.

#### *4.1.2.3. Загрузчик данных (Data Loader)*

Обязанностями загрузчика данных являются:

- глобальное разделение данных по заданному ключу при их загрузке из HDFS;
- разбиение данных, хранимых в одном узле, на несколько более мелких разделов (*чанков, chunk*);
- массовая загрузка данных в базу данных каждого узла с использованием чанков.

Загрузчик данных состоит из компонентов GlobalHasher и LocalHasher. GlobalHasher запускает в Hadoop MapReduce специальное задание, в котором читаются файлы данных HDFS и производится их разделение на столько частей, сколько имеется рабочих узлов в кластере. Сортировка данных не производится. Затем LocalHasher в каждом узле копирует соответствующий раздел из HDFS в свою файловую систему, разделяя его на чанки в соответствии с установленным в системе максимальным размером чанка.

В GlobalHasher и LocalHasher используются разные хэш-функции, обеспечивающие примерно одинаковые размеры всех чанков. Эти хэш-функции отличаются от хэш-функции, используемой в Hadoop MapReduce для разделения данных по умолчанию. Это способствует улучшению балансировки нагрузки.

#### *4.1.2.4. Планирование SQL-запросов*

Внешний интерфейс HadoopDB позволяет выполнять SQL-запросы. Компиляцию и подготовку планов выполнения SQL-запросов производит планировщик SMS (SMS Planner на рис. 4), являющийся расширением планировщика Hive [50].

Планировщик Hive преобразует запросы, представленные на языке HiveQL (вариант SQL) в задания MapReduce, которые выполняются над таблицами, хранимыми в виде файлов HDFS. Эти задания представляются в виде



ориентированных ациклических графов (directed acyclic graph, DAG) реляционных операций фильтрации (ограничения), выборки (проекции), соединения, агрегации, каждая из которых выполняется в конвейере: после обработки каждого очередного кортежа результат каждой операции направляется на вход следующей операции.

Операции соединения, как правило, выполняются в задаче *Reduce* MR-задания, соответствующего SQL-запросу. Это связано с тем, что каждая обрабатываемая таблица сохраняется в отдельном файле HDFS, и невозможно предполагать совместного размещения соединяемых разделов таблиц в одном узле кластера. Для HadoopDB это не всегда так, поскольку соединяемые таблицы могут разделяться по атрибуту соединения, и тогда операцию соединения можно вытолкнуть на уровень СУБД.

Для пояснения того, как работает планировщик Hive, и каким образом его функциональность расширяется в SMS, невозможно обойтись без примера, и для простоты воспользуемся примером из [30]. Пусть задан следующий простой запрос с агрегацией, смысл которого состоит в получении ежегодных суммарных доходов от продаж товаров:

```
SELECT YEAR(saleDate), SUM(revenue)
  FROM sales
  GROUP BY YEAR(saleDate);
```

В Hive этот запрос обрабатывается следующим образом:

- Производится синтаксический разбор запроса, и образуется его абстрактное синтаксическое дерево.
- Далее работает семантический анализатор, который выбирает из внутреннего каталога Hive *MetaStore* информацию о схеме таблицы `sales`, а также инициализирует структуры данных, требуемые для сканирования этой таблицы и выборки нужных полей.
- Затем генератор логических планов запросов производит план запроса – DAG реляционных операций.
- Вслед за этим оптимизатор перестраивает этот план запроса, проталкивая, например, операции фильтрации ближе к операциям сканирования таблиц. Основной функцией оптимизатора является разбиение плана на фазы *Map* и *Reduce*. В частности, перед операциями соединения и группировки добавляется операция переразделения данных (Reduce Sink). Эти операции отделяют фазу *Map* от фазы *Reduce*. Оценочная (cost-based) оптимизация не используется, и поэтому получаемые планы не всегда эффективны.
- Генератор физических планов выполнения запросов преобразует логический план в физический, допускающий выполнение в виде одного или нескольких MR-заданий. Первая (и каждая аналогичная) операция Reduce Sink помечает переход от фазы Map к фазе Reduce некоторого задания MapReduce, а остальные операции Reduce Sink помечают начало следующего задания MapReduce. Физический план

выполнения приведенного выше запроса, сгенерированный планировщиком Hive, показан на рис. 5.

- Полученный DAG сериализуется в формате XML. Задания инициируются драйвером Hive, который руководствуется планом в формате SQL и создает все необходимые объекты, сканирующие данные в таблицах HDFS и покортежно обрабатывающие данные.

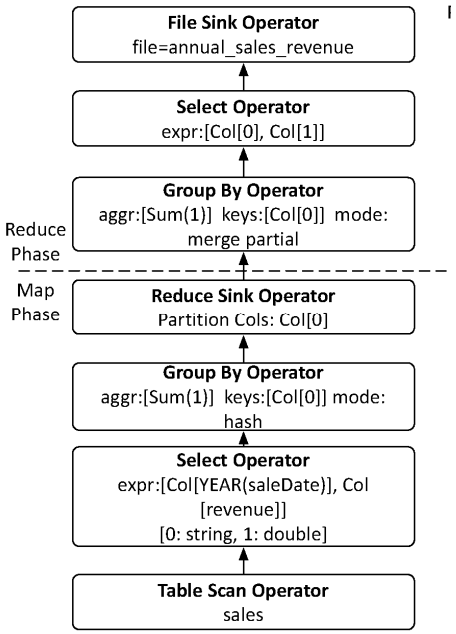


*Рис. 5. Задание MapReduce, генерируемое Hive*

В планировщике SMS функциональность планировщика Hive расширяется следующим образом. Во-первых, до обработки каждого запроса модифицируется MetaStore, куда помещается информация о таблицах базы данных. Для этого используется каталог HadoopDB (см. выше).

Далее, после генерации физического плана запроса и до выполнения MR-заданий выполняются два прохода по физическому плану. На первом проходе устанавливается, какие столбцы таблиц действительно обрабатываются запросом, и определяются ключи разделения, используемые в операциях Reduce Sink.

На втором проходе DAG запроса обходится снизу-вверх от операций сканирования таблиц до формирования результата или первой операции



Reduce Sink. Все операции этой части DAG преобразуются в один или несколько SQL-запросов, которые проталкиваются на уровень СУБД. Для повторного создания кода SQL используется специальный основанный на правилах генератор.

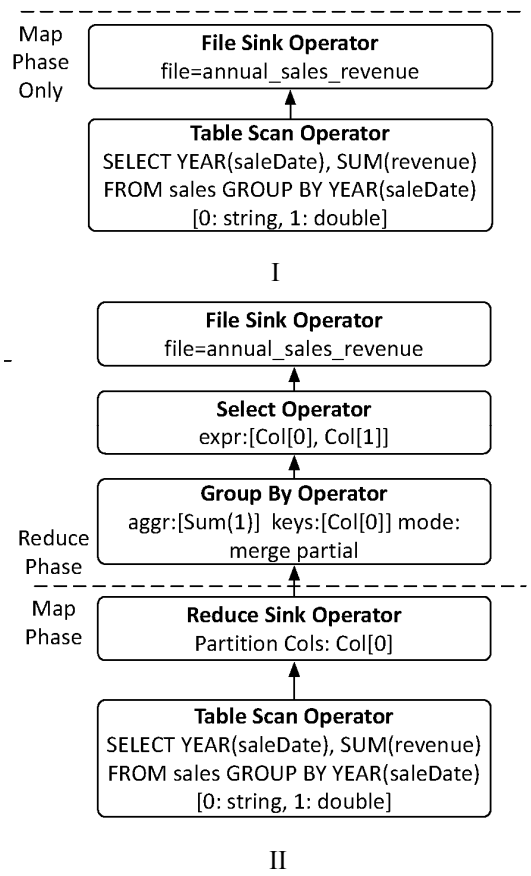


*Рис. 6. Варианты MR-заданий, генерируемые SMS*

На рис. 6 показаны два плана, которые производит SQL для приведенного выше запроса. План в левой части рисунка производится в том случае, если таблица `sales` является разделенной по `YEAR(saleDate)`. В этом случае вся логика выполнения запроса выталкивается в СУБД. Задача *Map* всего лишь записывает результаты запроса в файл HDFS.

В противном случае генерируется план, показанный в правой части рис. 6. При выполнении запроса по этому плану на уровне базы данных производится

частичная агрегация данных, а для окончательной агрегации требуется выполнение задачи *Reduce*, производящей слияние частичных результатов группировки, которые получены в каждом узле на фазе задачи *Map*.

## 4.2. Производительность, масштабируемость и устойчивость к отказам и падению производительности узлов кластера

В [30] описан ряд экспериментов, показывающих, что гибридное использование технологий MapReduce и баз данных в реализации HadoopDB позволяет добиться от этой системы производительности, соизмеримой с производительностью параллельных СУБД, и устойчивости к отказам и падению производительности узлов, свойственной MapReduce. Я не буду в этой статье описывать детали этих экспериментов, а лишь кратко отмечу их основные результаты.

### *4.2.1. Производительность и маштабируемость*

В большинстве экспериментов параллельные СУБД существенно превосходят HadoopDB по производительности, а HadoopDB оказывается значительно (иногда на порядок) производительнее связки Hive и Hadoop MapReduce. В экспериментах использовались поколоночная параллельная СУБД Vertica и некоторая коммерческая параллельная СУБД-X с хранением таблиц по строкам. Наибольшую производительность, естественно, демонстрировала Vertica, но в ряде случаев HadoopDB уступала ей значительно меньше, чем на десятичный порядок.

В [30] значительное отставание HadoopDB от параллельных СУБД объясняется тем, что в качестве базовой СУБД в HadoopDB использовалась PostgreSQL, в которой отсутствует возможность хранения таблиц по столбцам (как уже отмечалось, в [49] в HadoopDB уже используется поколоночная СУБД). Кроме того, в экспериментах с HadoopDB не использовалось сжатие данных. Наконец, в HadoopDB возникали значительные накладные расходы на взаимодействие Hadoop MapReduce и PostgreSQL, которые потенциально можно снизить. Так что в целом производительность HadoopDB не должна критически отставать от производительности параллельных СУБД.

Время загрузки данных в HadoopDB в десять раз больше соответствующего времени для Hadoop MapReduce. Однако это окупается десятикратным выигрышем в производительности при выполнении некоторых запросов.

Как и следовало ожидать, при возрастании числа узлов в кластере при одновременном увеличении объема данных HadoopDB (как и Hadoop) масштабируется почти линейно. Но следует заметить, что в этом диапазоне не хуже масштабируется и Vertica (с СУБД-X дела обстоят несколько хуже), а эксперименты на кластерах большего размера не производились. Так что объективных данных в этом отношении пока нет.



### *4.2.2. . Устойчивость к отказам и неоднородности среды*

В экспериментах с отказоустойчивостью и падением производительности некоторого узла сравнивались HadoopDB, Hadoop MapReduce с Hive и Vertica. В первом случае работа одного из узлов кластера искусственным образом прекращалась после выполнения 50% обработки запроса. Во втором случае работа одного узла замедлялась за счет выполнения фонового задания с большим объемом ввода-вывода с тем же диском, на котором сохранялись файлы соответствующей системы.

При продолжении работы после отказа узла СУБД Vertica приходилось выполнять запрос заново с использованием реплик данных, и время выполнения запроса возрастало почти вдвое. В HadoopDB и Hadoop MapReduce с Hive время выполнения увеличивалось примерно на 15-20% за счет того, что задачи, выполнявшиеся на отказавшем узле, перераспределялись между оставшимися узлами. При этом относительная производительность HadoopDB оказывается несколько выше, чем у Hadoop MapReduce с Hive, поскольку в первом случае обработка запроса проталкивалась на узлы, содержащие реплики баз данных, а во втором приходилось копировать данные, не являющиеся локальными для обрабатывающего узла.

При замедлении работы одного из узлов производительность Vertica определялась скоростью этого узла, и в экспериментах время выполнения запроса увеличивалось на 170%. При использовании HadoopDB и Hadoop MapReduce с Hive время выполнения запроса увеличивалось всего на 30% за счет образования резервных избыточных задач в недозагруженных узлах.

Проект HadoopDB представляется мне очень интересным и перспективным. В отличие от других систем, рассматриваемых в этой статье, HadoopDB – это проект с открытыми исходными текстами, так что потенциально участие в этой работе доступно для всех желающих. Помимо прочего, продукт HadoopDB открывает путь к созданию высокопроизводительных, масштабируемых и отказоустойчивых параллельных СУБД на основе имеющихся программных средств с открытыми кодами.

## *5. Использование MapReduce для подготовки данных параллельных СУБД*

Я не могу считать себя специалистом в области различных средств ETL (Extract-Transform-Load), которых существует множество, и которые применяются во многих компаниях, использующих хранилища данных (я не буду даже пытаться как-то их классифицировать и/или сравнивать). Но, в любом случае, важность этих средств трудно переоценить, поскольку в хранилище данных по определению поступают данные из самых разнообразных источников: транзакционных баз данных, сообщений, участвующих в организации бизнес-процессов, электронной почты, журналов

Web-серверов и т.д. Все эти данные нужно очистить, привести к единому формату, согласовать и загрузить в хранилище данных для последующего анализа.

Наверное, можно согласиться с идеологами MAD-аналитики из Greenplum (см. п. 3.1.1), что при использовании ортодоксального подхода к организации хранилищ данных подключение нового источника к хранилищу данных может занять недопустимо много времени во многом как раз из-за потребности в создании соответствующей процедуры ETL. Наверное, можно согласиться и с тем, что для аналитиков гораздо важнее получить новые данные, чем быть вынужденными ждать неопределенное время их в согласованной форме. Но совершенно очевидно, что если данные в хранилище данных не очищать никогда, то со временем в них не разберется никакой, даже самый передовой аналитик.

Итак, что мы имеем. Число источников данных, пригодных для анализа в составе хранилища данных, все время растет. Их разнородность тоже все время возрастает. Все меньший процент составляют структурированные базы данных, данные поступают из частично структурированных файлов и совсем неструктурированных текстовых документов. Для каждой разновидности источников данных нужна своя разновидность процедуры ETL, и по причине роста объемов исходных данных для обеспечения умеренного времени их загрузки в хранилище данных эти процедуры должны выполняться в массивно-параллельной среде. И в этом может помочь технология MapReduce.

## 5.1. MapReduce и ETL

Как отмечается в [31], для канонического способа использования технологии MapReduce характерно применение следующих операций:

- чтение журнальных данных из нескольких разных файлов-журналов;
- разбор и очистка журнальных данных;
- преобразования этих данных, в том числе их частичная агрегация;
- принятие решения о схеме результирующих данных;
- загрузка данных в хранилище данных или другую систему хранения.

В точности такие же шаги выполняются в системах ETL при извлечении, преобразовании и загрузке данных. По сути дела, MapReduce производит из исходных "сырых" данных некоторую полезную информацию, которую потребляет другая система хранения. В некотором смысле можно считать любую реализацию MapReduce параллельной инфраструктурой выполнения процедур ETL.

Имелись попытки реализации процедур ETL внутри сервера баз данных средствами языка SQL. Разработчики параллельных СУБД с поддержкой MapReduce Greenplum Database и *n*Cluster компании Aster Data тоже намекают, что их встроенный MapReduce можно использовать и для поддержки ETL. Но исторически системы ETL промышленного уровня



существуют отдельно от СУБД. Обычно СУБД не пытаются выполнять ETL, а системы ETL не поддерживают функции СУБД.

## 5.2. Hadoop и Vertica

Эти рассуждения наводят на мысль, что если иметь в виду поддержку именно ETL, то наиболее грамотное (и самое простое) решение по интеграции технологий MapReduce и параллельных баз данных применяется в Vertica (см. рис. 7, позаимстованный из [51]).

В Vertica реализован свой вариант интерфейса DBInputFormat компании Cloudera [51] для Hadoop MapReduce, позволяющий разработчикам MapReduce выбирать данные из баз данных Vertica и направлять результирующие данные в эти базы данных. При этом подходе технологии MapReduce и параллельных баз данных тесно не интегрируются, но каждая из них может использовать возможности другой технологии.

Скорее всего, мы еще многое услышим о системах ETL, основанных на использовании технологии MapReduce, и, скорее всего, предводителем этого направления будет Vertica.

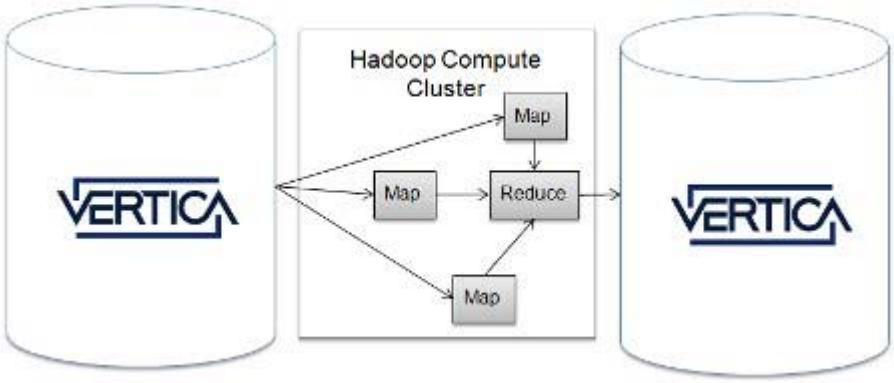


*Рис. 7. MapReduce*

# *6. Заключение*

Еще пару лет назад было непонятно, каким образом можно с пользой применять возникающие "облачные" среды для высокоуровневого управления данными. Многие люди считали, что в "облаках" системы управления базами данных будут просто вытеснены технологий MapReduce. Это вызывало естественное недовольство сообщества баз данных, авторитетные представители которого старались доказать, что пытаться заменить СУБД

какой-либо реализацией MapReduce если не безнравственно, то, по крайней мере, неэффективно.

Однако вскоре стало понятно, что технология MapReduce может быть полезна для самих параллельных СУБД. Во многом становлению и реализации этой идеи способствовали компании-стартапы, выводящие на рынок новые аналитические массивно-параллельные СУБД и добивающиеся конкурентных преимуществ. Свою лепту вносили и продолжают вносить и университетские исследовательские коллективы, находящиеся в тесном сотрудничестве с этими начинающими компаниями.

На сегодняшний день уже понятно, что технология MapReduce может эффективно применяться внутри параллельной аналитической СУБД, служить инфраструктурой отказоустойчивой параллельной СУБД, а также сохранять свою автономность в симбиотическом союзе с параллельной СУБД. Все это не только мешает развитию технологии параллельных СУБД, а наоборот, способствует ее совершенствованию и распространению.

Интересные работы ведутся и в направлении использования "облачных" сред для создания нового поколения транзакционных средств управления данными. Но это уже, как говорили братья Стругацкие, совсем другая история.

**Литература**


[1] Rakesh Agrawal, Anastasia Ailamaki, Philip A. Bernstein, Eric A. Brewer, Michael J. Carey, Surajit Chaudhuri, AnHai Doan, Daniela Florescu, Michael J. Franklin, Hector Garcia Molina, Johannes Gehrke, Le Gruenwald, Laura M. Haas, Alon Y. Halevy, Joseph M. Hellerstein, Yannis E. Ioannidis, Hank F. Korth, Donald Kossmann, Samuel Madden, Roger Magoulas, Beng Chin Ooi, Tim O'Reilly, Raghu Ramakrishnan, Sunita Sarawagi, Michael Stonebraker, Alexander S. Szalay, Gerhard Weikum. The Claremont Report on Database Research, http://db.cs.berkeley.edu/claremont/claremontreport08.pdf, 2008 г.
Перевод на русский язык: Ракеш Агравал, Анастасия Айламаки, Филипп Бернштейн, Эрик Брювер, Майкл Кери, Сураджит Чаудхари, Анхай Доан, Даниэла Флореску, Майкл Франклин, Гектор Гарсиа Молина, Йоханнес Герке, Ле Грюнвальд, Лаура Хаас, Элон Хэлеви, Джозеф Хелерстейн, Яннис Иоаннидис, Хэнк Корт, Дональд Коссман, Сэмюэль Мэдден, Роджер Магулас, Бенг Чин Ой, Тим О'Рейли, Раджу Рамакришнан, Суннита Сарагави, Майкл Стоунбрейкер, Александер Залай, Герхард Вейкум. Клермонтский отчет об исследованиях в области баз данных, http://citforum.ru/database/articles/claremont_report/, 2008 г.

[2] Curt Monash. Data warehouse appliances – fact and fiction, http://www.dbms2.com/2007/12/03/data-warehouse-appliances-%e2%80%93-fact-and-fiction/, December 3, 2007

[3] Википедия. Britton Lee, Inc., http://en.wikipedia.org/wiki/Britton_Lee,_Inc., 2010

[4] Teradata Home Page, http://www.teradata.com/t/, 2010

[5] C.H.C. Lemg and K.S. Wong. File Processing Efficiency on the Content Addressable File Store, http://www.vldb.org/conf/1985/P282.PDF, Proceedings of the VLDB Conference, Stockholm, 1985, 282-291





[6] Netezza Home Page, http://www.netezza.com/, 2010

[7] Vertica Systems Home Page, http://www.vertica.com/, 2010

[8] DATAllegro Home Page, http://www.datallegro.com/, 2010

[9] Greenplum Home Page, http://www.greenplum.com/, 2010

[10] Aster Data Systems Home Page, http://www.asterdata.com/, 2010

[11] Kognitio Home Page, http://www.kognitio.com/, 2010

[12] EXASOL AG Home Page, http://www.exasol.com/, 2010

[13] Calpont Corporation Home Page, http://www.calpont.com/, 2010

[14] Dataupia Corporation Home Page, http://www.dataupia.com/, 2010

[15] Infobright Home Page, http://www.infobright.com/, 2010

[16] Kickfire Home Page, http://www.kickfire.com/, 2010

[17] SQL Server 2008 R2 Parallel Data Warehouse, http://www.microsoft.com/sqlserver/2008/en/us/parallel-data-warehouse.aspx, 2010

[18] Ingres Corporation Home Page, http://www.ingres.com/, 2010

[19] PostgreSQL Home Page, http://www.postgresql.org/, 2010

[20] MySQL Home Page, http://www.mysql.com/, 2010

[21] Oracle Exadata, http://www.oracle.com/us/products/database/exadata/index.html, 2010

[22] Richard Hackathorn, Colin White. Data Warehouse Appliances: Evolution or Revolution?, http://www.beyeresearch.com/study/4639, June 26, 2007

[23] M. Stonebraker and U. Cetintemel. One Size Fits All: An Idea whose Time has Come and Gone, http://www.cs.brown.edu/%7Eugur/fits_all.pdf // Proc. ICDE, 2005, 2-11.
Перевод на русский язык: Майкл Стоунбрейкер, Угур Кетинтемел. Один размер пригоден для всех: идея, время которой пришло и ушло, http://citforum.ru/database/articles/one_size_fits_all/, 2007

[24] Jeffrey Dean and Sanjay Ghemawat. MapReduce: Simplified Data Processing on Large Clusters, http://labs.google.com/papers/mapreduce.html // Proceedings of the Sixth Symposium on Operating System Design and Implementation, San Francisco, CA, December, 2004, 137–150

[25] Michael Stonebraker, David J. DeWitt. MapReduce: A major step backwards, http://databasecolumn.vertica.com/database-innovation/mapreduce-a-major-step-backwards/, January 17, 2008

[26] Michael Stonebraker, David J. DeWitt. MapReduce II, http://databasecolumn.vertica.com/database-innovation/mapreduce-ii/, January 25, 2008

[27] Andrew Pavlo, Erik Paulson, Alexander Rasin, Daniel J. Abadi, David J. DeWitt, Samuel Madden, Michael Stonebraker. A Comparison of Approaches to Large-Scale Data Analysis, http://cs-www.cs.yale.edu/homes/dna/papers/benchmarks-sigmod09.pdf // Proceedings of the 35th SIGMOD International Conference on Management of Data, 2009, Providence, Rhode Island, USA, 165-178.
Перевод на русский язык: Эндрю Павло, Эрик Паулсон, Александр Разин, Дэниэль Абади, Дэвид Девитт, Сэмюэль Мэдден, Майкл Стоунбрейкер. Сравнение подходов к крупномасштабному анализу данных, http://citforum.ru/database/articles/mr_vs_dbms/, 2009

[28] Jeffrey Cohen, Brian Dolan, Mark Dunlap, Joseph M. Hellerstein, Caleb Welton. MAD Skills: New Analysis Practices for Big Data, http://db.cs.berkeley.edu/jmh/papers/madskills-032009.pdf // Proceedings of the VLDB'09 Conference, Lyon, France, August 24-28, 2009, 1481-1492.
Перевод на русский язык: Джеффри Коэн, Брайен Долэн, Марк Данлэп, Джозеф Хеллерстейн, Кейлэб Велтон. МОГучие способности: новые приемы анализа больших данных, http://citforum.ru/database/articles/mad_skills/, 2009

[29] Eric Friedman, Peter Pawlowski, John Cieslewicz. SQL/MapReduce: A practical approach to self-describing, polymorphic, and parallelizable userdefined functions, http://www.asterdata.com/resources/downloads/whitepapers/sqlmr.pdf // Proceedings of the 35th VLDB Conference, August 24-28, 2009, Lyon, France, 1402-1413.
Перевод на русский язык: Эрик Фридман, Питер Павловски и Джон Кислевич. SQL/MapReduce: практический подход к поддержке самоописываемых, полиморфных и параллелизуемых функций, определяемых пользователями, http://citforum.ru/database/articles/asterdata_sql_mr/, 2010

[30] Azza Abouzeid, Kamil Bajda-Pawlikowski, Daniel Abadi, Avi Silberschatz, Alexander Rasin. HadoopDB: An Architectural Hybrid of MapReduce and DBMS Technologies for Analytical Workloads, http://www.vldb.org/pvldb/2/vldb09-861.pdf // Proceedings of the 35th VLDB Conference, August 24-28, 2009, Lyon, France, 922-933.
Перевод на русский язык: Азза Абузейд, Камил Байда-Павликовски, Дэниэль Абади, Ави Зильбершац, Александр Разин. HadoopDB: архитектурный гибрид технологий MapReduce и СУБД для аналитических рабочих нагрузок, http://citforum.ru/database/articles/hadoopdb/, 2010

[31] Michael Stonebraker, Daniel Abadi, David J. Dawitt, Sam Madden, Erik Paulson, Andrew Pavlo and Alexander Rasin. MapReduce and Parallel DBMSs: Friends or Foes?, http://database.cs.brown.edu/papers/stonebraker-cacm2010.pdf. // Communications of the ACM, vol. 53, no. 1, January 2010, 64-71.
Перевод на русский язык: Майкл Стоунбрейкер, Дэниэль Абади, Дэвит Девитт, Сэм Мэдден, Эрик Паулсон, Эндрю Павло и Александр Разин. MapReduce и параллельные СУБД: друзья или враги?, http://citforum.ru/database/articles/mr_vs_dbms-2/, 2010

[32] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. The Google File System, http://static.googleusercontent.com/external_content/untrusted_dlcp/labs.google.com/ru/papers/gfs-sosp2003.pdf // Proceedings of the ACM Symposium on Operating Systems Principles, Bolton Landing, New York, USA, October 19–22, 2003, 29 - 43

[33] Hadoop MapReduce Home Page, http://hadoop.apache.org/mapreduce/, 2010

[34] Apache Hadoop Home Page, http://hadoop.apache.org/, 2010

[35] The Apache Software Foundation Home Pagem, http://www.apache.org/, 2010

[36] Hadoop Distributed File System Home Page, http://hadoop.apache.org/hdfs/, 2010

[37] Map/Reduce Tutorial, http://hadoop.apache.org/common/docs/current/mapred_tutorial.htm, 2010

[38] Lawrence A. Rowe, Michael R. Stonebraker. The POSTGRES Data Model, http://www.vldb.org/conf/1987/P083.PDF // Proceedings of the 13th VLDB Conference, Brighton, 1987, 83-96

[39] Michael Stonebraker, Erlka Anderson, Eric Hanson and Brad Ruben. Quel as a Data Type. ACM SIGMOD Record, Volume 14 , Issue 2 (June 1984), 208-214



[40] Michael Stonebraker, Jeff Anton, Eric N. Hanson. Extending a Database System with Procedures, http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.47.4497&rep=rep1&type=pdf // ACM Trans. Database Syst. 12(3), 1987, 350-376

[41] MPI: The Message Passing Interface, http://parallel.ru/tech/tech_dev/mpi.html, 2010

[42] Michael Stonebraker, JeffAnton, and Michael Hirohama. Extensibility in Postgres, http://sites.computer.org/debull/87JUN-CD.pdf // IEEE Database Engineering Bulletin 10(2), June 1987, 16-24

[43] PostgreSQL 8.4.3 Documentation. V. Server Programming, http://www.postgresql.org/docs/8.4/static/server-programming.html, 2010

[44] A Unified Engine for RDBMS and MapReduce, http://www.greenplum.com/download.php?alias=register-map-reduce&file=Greenplum-MapReduce-Whitepaper.pdf, Greenplum Whitepaper, 2009

[45] Python Package Index (PyPi) Home Page, http://pypi.python.org/pypi. 2010

[46] Comprehensive Perl Archive Network (CPAN) Home Page, http://www.cpan.org/, 2010

[47] Ajeet Singh. Aster Data's SQL-MapReduce: Deriving Deep Insights from Large Datasets, http://www.asterdata.com/resources/assets/wp_Aster_Data_4.0_MapReduce_Technical_Whitepaper.pdf, Aster Data Whitepaper, 2009

[48] HadoopDB Home Page, http://db.cs.yale.edu/hadoopdb/hadoopdb.html, 2010

[49] Azza Abouzied, Kamil Bajda-Pawlikowski, Jiewen Huang, Daniel J. Abadi, Avi Silberschatz. HadoopDB in Action: Building Real World Applications, http://cs-www.cs.yale.edu/homes/dna/papers/hadoopdb-demo.pdf // Proceedings of the 36th SIGMOD International Conference on Management of Data, 2010, Indianapolis, Indiana, USA.

[50] Hive Home Page, http://hadoop.apache.org/hive/, 2010

[51] Aaron Kimball. Database Access with Hadoop, http://www.cloudera.com/blog/2009/03/database-access-with-hadoop/, March 06, 2009