О. Д. Гобрик, А. Б. Гуринович

ОПТИМИЗАЦИЯ ОБРАБОТКИ ДАННЫХ В ПАРАЛЛЕЛЬНЫХ СИСТЕМАХ

В работе исследуется инструмент параллельной обработки данных MapReduce и метод принятия решений во время выполнения, чтобы улучшить пропускную способность соединения. Рассмотренный метод использует балансировку нагрузки на вычислительные узлы и узлы хранения.

Введение

С ростом объема данных и множеством источников потребности в обработке данных меняются. Хотя реляционные СУБД остаются основной технологией управления данными для обработки структурированных данных, но столкнувшись с резким ростом объема данных, реляционные базы данных, достигают своих пределов.

Системы параллельной пакетной обработки данных (Hadoop MapReduce и Spark), предназначены для обработки данных на большом кластере машин. Такие системы, как HBase и Cassandra, часто используются в качестве внутренних хранилищ данных и поддерживают индексированный доступ по первичным ключам.

Однако объемные данные, и множество источников, которые привели к разнообразию структур, изменяют и потребности обработки данных. С этими изменениями технологии баз данных развивались, и были представлены новые модели. MapReduce - одна из таких платформ, которая стала успешной для приложений, обрабатывающих большие объемы данных.

I. MapReduce

MapReduce - это модель программирования, разработанная Google. Она создана для обработки больших наборов данных с помощью параллельного распределенного алгоритма в кластере. MapReduce создан для упрощения параллельной обработки и распределения данных на большом количестве машин с абстракцией, которая скрывает детали аппаратного уровня для программистов: она скрывает детали распараллеливания, отказоустойчивости, оптимизации локальности и балансировки нагрузки.

Чаще всего фреймворки реализующие модель MapReduce осуществляют три операции (шага)[1]:

1. Map: входные данные решаемой задачи представляют большой список значений, и на шаге Map происходит его предварительная обработка. Для этого главный узел кластера (master node) получает этот список, делит его на части и передает рабочим узлам (worker node). Каждый рабочий узел применяет функцию Map к локальным данным и записывает результат в формате «ключ-значение» во временное хранилище.

2. Shuffle: рабочие узлы перераспределяют данные на основе ключей (созданных функцией Map) так, что все данные принадлежащие одному ключу лежат на одном рабочем узле.

3. Reduce: рабочие узлы обрабатывают каждую группу результатов по порядку следования ключей. Главный узел получает промежуточные ответы от рабочих узлов и передаёт их на свободные узлы для выполнения следующего шага. Получившийся после прохождения всех необходимых шагов результат – это решение задачи, которая изначально формулировалась.

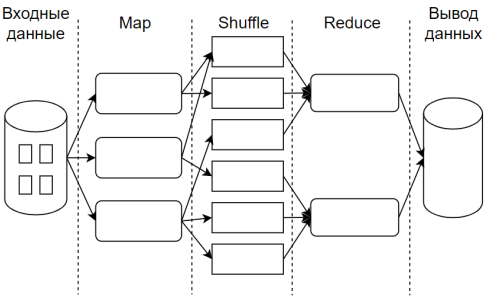

Рис. 1 – Архитектура MapReduce

Описание связанных функций высшего порядка:

1. Функция map - принимает на вход список и некую функцию, затем применяет данную функцию к каждому элементу списка и возвращает новый список.

2. Функция reduce (свёртка) - преобразует список к единственному атомарному значению при помощи заданной функции, которой на каждой итерации передаются новый элемент списка и промежуточный результат.

Для обработки данных пользователь библиотеки MapReduce должен только определить две эти функции, а также указать имена входных и выходных файлов и параметры обработки.

II. Хранение данных

Подход заключается в размещении хранимых данных, проиндексированных по ключу соединения, в параллельном хранилище данных. Чтобы обработать входящий элемент данных, ищутся значения из параллельного хранилища данных с помощью ключа соединения и выполняются вычисления, если таковые имеются, на основе полученного значения. Если удаленные

данные еще не проиндексированы ключом соединения в параллельном хранилище данных, то они могут быть перераспределены, и проиндексированы.[2]

Многие параллельные хранилища данных поддерживают выполнение определенных пользователем функций для определенных элементов данных в узлах данных, например, конечные точки в HBase. Возможность выполнять определяемые пользователем функции на узлах данных позволяет подтолкнуть выполнение функции к узлам данных.

III. Оптимизация обращений в хранилище данных

Запросы к хранилищу данных обычно блокируются. Эти блокирующие вызовы приведят к снижению пропускной способности, поскольку каждый процесс/поток будет иметь только один активный запрос за раз. Отправка одного запроса за раз приводит к плохому использованию ресурсов.

Однако независимо от того, предоставляет ли хранилище данных вызовы асинхронного доступа к хранилищу данных, если приложение обрабатывает один входной кортеж (например, Hadoop) или один пакет кортежей за раз (например, Spark Streaming), оно будет заблокировано для обработки кортежа (или пакета) для завершения. Чтобы эффективно использовать асинхронные вызовы, необходимо существенно изменить приложение. Более чистый подход - отправлять запросы предварительной выборки, желательно в отдельной функции.

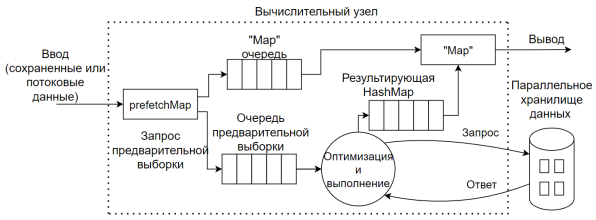

Рис. 2 – Вызов функций prefetchMap и Map

1. Предварительная загрузка. Расширяя MapReduce API Hadoop, добавив функцию prefetchMap, которую можно использовать для отправки запросов предварительной выборки. Пользователь может предоставить реализацию функции prefetchMap для выдачи запросов предварительной выборки.

Запрос предварительной выборки возвращается сразу после принятия мер по инициированию выборки результатов в фоновом режиме. Функция драйвера (функция, которая вызывает функцию Map на вычислительном узле) модифицируется так, что функции prefetchMap и Map выполняются как отдельные потоки. Функция prefetchMap извлекает элементы данных из источника, выполняет предварительную выборку, а затем добавляет элементы данных из ввода в очередь Map, как показано на Рис.2. Вычислительный узел решает, как выполнять функцию (на вычислительных узлах или на узле данных) и отправляет соответствующие запросы к узлам данных. Как только функция для кортежа вычислена, она добавляется к результирующей HashMap, откуда функция Map может считывать вычисленное значение, соответствующее кортежу, считанному из очереди отображения.

В случае объединений несколько таких блоков, как показано на Рис.2, можно строить параллельную систему вычисления. Вместо одной пары функций <prefetchMap, Map> создается серия пар <prefetchMap, Map>, каждая для вычисления одного соединения.

2. Распределение данных. Отправка данных или запросов на вычисления по отдельности может привести к снижению производительности. Чтобы повысить производительность, необходимо объединить несколько запросов на выборку / предварительную выборку или вычисление данных в один пакетный вызов.

Размер пакета определяется статически, при этом пакеты сохраняются достаточно большими, чтобы гарантировать, что накладные расходы на запрос невелики по сравнению с фактической стоимостью запроса. Для потоковых систем большой размер пакета полезен для повышения пропускной способности. По прошествии заданного времени с момента добавления первого элемента данных в очередь отправляется пакет независимо от того, заполнен пакет или нет.

IV. Заключение

Проведенные исследования показали, что модель распределённых вычислений MapReduce с использованием расширений представляет собой эффективный алгоритм оптимизации предварительной выборки и пакетной обработки для приложений на параллельных платформах данных.

1. Dean J., Ghemawat S. MapReduce: Simplified data processing on large clusters//Proceedings of Operating System Design and Implementation(OSDI), 2004,pp.137-150.
2. Hadoop. Подробное руководство / Том Уайт. – СПб.: Питер, 2013. – 672 с.

Гобрик Олег Дмитриевич, магистрант кафедры информационных технологий автоматизированных систем БГУИР, oleg.gobrik@gmail.com.

Научный руководитель: Гуринович Алевтина Борисовна, заместитель декана по научно-методической работе БГУИР, кандидат физико-математических наук, доцент, gurinovich@bsuir.by.