A. M. Chamkin, B. A. Kurdikov
Saint-Petersburg state electrotechnical university «LETI»
V. Yu. Smirnov
OJSC «RPC "Karat"» (Saint-Petersburg)

THE PULSE AND THE FREQUENCY ANALYSIS METHODS FOR MONITORING THE WINDINGS CONDITION OF THE ELECTRICAL MACHINES

The article describes the main causes of internal damage to the electrical machine windings. Two fault diagnosis method windings: pulse method and frequency analysis method are reviewed. Method for converting signals from the time domain to the frequency domain using a discrete Fourier transform is shown. The results of experiments using these methods are presented. The main advantages and disadvantages of these diagnosis methods are identified.

Pulse analysis, frequency analysis, winding, transformer, diagnosis, interturn fault, short circuit

УДК 681.324

А. В. Митяков, Ю. С. Татаринов
Санкт-Петербургский государственный электротехнический университет «ЛЭТИ» им. В. И. Ульянова (Ленина)

Подход к ускорению итеративных алгоритмов на модели MapReduce

Рассматривается метод выборочной обработки данных при исполнении итеративных алгоритмов на модели MapReduce. Предлагается метод выборочной обработки данных, основанный на приоритетах. Приводятся данные эксперимента на примере алгоритма поиска кратчайших путей в графе.

MapReduce, итеративные алгоритмы, выборочная обработка данных

Итеративные алгоритмы представляют собой важный класс алгоритмов, лежащих в основе многих других: машинное обучение, графовые алгоритмы, ссылочный анализ и др. Суть итеративных алгоритмов заключает в том, что данные обрабатываются в цикле и выход одной итерации идет на вход следующей до тех пор, пока выполняется некоторое условие. С ростом объема данных в различных областях человеческой деятельности возникла необходимость в распределенных системах обработки данных. Наиболее популярными и известными системами такого рода являются различные реализации модели MapReduce, например ApacheHadoop.

Исполнение итеративных алгоритмов на модели MapReduce возможно, однако с точки зрения эффективности оно заметно проигрывает в сравнении с различными более специализированными системами. Недостатки модели MapReduce для исполнения итеративных алгоритмов связаны с одним из главных преимуществ данной модели, а именно – с отказоустойчивостью. Итеративный алгоритм в рамках MapReduce представляет собой последовательность MapReduce-задач (пары фаз Map и Reduce), решаемых в цикле до выполнения некоторого условия. При этом для обеспечения отказоустойчивости результаты каждой MapReduce-задачи сохраняются в распределенной файловой системе. Следующая итерация в качестве исходных данных вынуждена вновь загружать данные из распределенной файловой системы. Затраты на сетевой ввод-вывод являются основным моментом, отрицательно влияющим на производительность итеративных алгоритмов на модели MapReduce.

Существуют различные системы исполнения итеративных алгоритмов, лишенные данного недостатка [1]. Все решения по сути представляют собой различные варианты баланса между отказо-

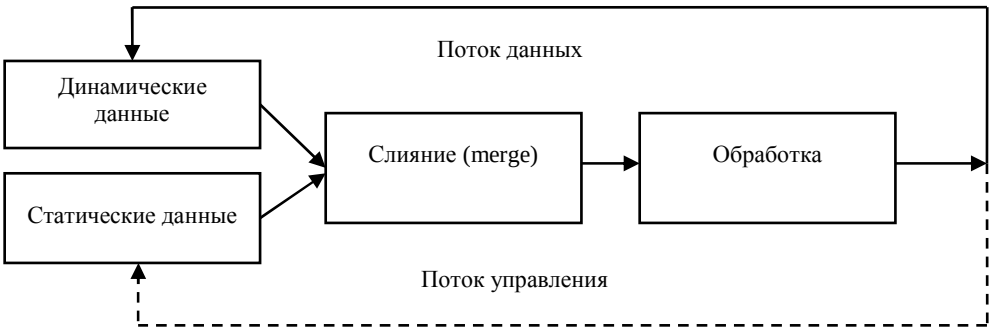

Рис. 1

устойчивостью и производительностью. При этом с точки зрения производительности в основном упор делается на снижение сетевого ввода-вывода.

В статье рассматривается возможность снижения как сетевого, так и дискового ввода-вывода за счет выборочной обработки данных на каждой итерации.

На рис. 1 представлена обобщенная схема работы существующих систем для исполнения итеративных алгоритмов (основные различия систем проявляются на более низком уровне конкретных реализаций).

Потоки данных в системе можно разделить на 2 типа: статические и динамические. Статическая часть данных (например, структура графа) не изменяется между итерациями и кешируется в узлах кластера. Статическая часть обычно наиболее объемная, и благодаря ее кешированию в узлах достигается значительная экономия сетевого трафика. Динамическая же часть является результатом работы алгоритма на конкретной итерации. Перед началом новой итерации происходит объединение статической и динамической частей для получения актуального с точки зрения алгоритма набора данных для текущей итерации.

В качестве расширения данной архитектуры предлагается ввести помимо потока данных еще и поток управления, который задавал бы множество данных (ключей) для обработки в следующей итерации.

Метод выборочной обработки данных. В модели MapReduce на фазе Map обрабатываются все доступные узлу данные. Таким образом, даже зная, какие данные в текущей итерации можно опустить, их все равно приходится обработать. Связано это с тем, что в момент выполнения фазы Map отсутствует информация о данных, обрабатываемых в параллельных Map-узлах. В связи с этим принять решение о целесообразности обработки конкретных данных можно только пост-фактум на этапе Reduce. Основная идея метода выборочной обработки заключается в том, чтобы уменьшить интервалы времени, между которыми можно принимать решения и корректировать выборку.

Предлагается модифицировать модель таким образом, чтобы предоставить разработчику алгоритма возможность задавать, какие именно данные следует обработать на следующей итерации.

В качестве примера рассмотрим алгоритм поиска кратчайших путей в графе. В [2] приводятся результаты работы данного алгоритма на двух реальных общедоступных графах (AmazonAds и YouTubeAds). Эксперимент проводился с целью сравнения количества обрабатываемых данных в немодифицированной версии MapReduce и при наличии обратной связи для корректировки выборки.

Результаты эксперимента показали, что возможность формирования выборки для обработки в следующей итерации позволяет снизить количество обрабатываемых данных на 50–90 %. В данной статье расширим этот подход введением понятия приоритета.

Приоритетное выполнение. Основной идеей, лежащей в основе предлагаемого подхода, является следующее предположение: «Порядок, в котором обрабатываются данные, влияет на скорость отработки алгоритма в целом».

Приоритетное выполнение означает следующее. На каждой итерации обрабатывается только K наиболее приоритетных по некоторому критерию данных. При этом K гораздо меньше общего количества доступных данных. Таким образом увеличивается общее количество итераций, но за счет небольших объемов данных проходят они гораздо быстрее. Далее на конкретном примере будет показано, что обработка данных в порядке приоритета может значительно снизить время выполнения алгоритма за счет использования различных эвристических подходов.

Вновь обратимся к алгоритму поиска кратчайшего пути в графе. Примером эвристики для данного алгоритма является аналогия с последовательной реализацией данного алгоритма – алгоритмом Дейкстры. Алгоритм Дейкстры является примером «жадного» алгоритма: каждый раз при движении по графу выбирается путь с наименьшим весом. В типичной реализации модели MapReduce воспользоваться данной эвристикой не представляется возможным, но ее вполне можно описать на уровне приоритезированной выборки.

Был проведен эксперимент, чтобы выяснить, позволит ли эта эвристика уменьшить общее количество передаваемых по сети данных в параллельной версии алгоритма. Модифицируем алгоритм таким образом, чтобы в каждой итерации обрабатывались *K* вершин с наименьшим текущим расстоянием.

Результаты эксперимента показывают, что использование приоритетов при обработке позволяет сократить объем передаваемых по сети данных до 38 % в сравнении с ручной выборкой данных и примерно в 7 раз по сравнению с классической реализацией модели MapReduce за счет двух дополнительных итераций.

Общая архитектура системы. Для того чтобы получить более точные временные оценки по эффективности предложенного метода и его применимости для других алгоритмов, необходимо реализовать фреймворк. Общая схема работы фреймворка для поддержки выборочной обработки данных представлена рис. 2.

По сравнению с базовой реализацией модели MapReduce предлагается ввести 3 дополнительных компонента:

1. Индексация.
2. Сортировка.
3. Выборка *K* верхних элементов.

Рассмотрим каждый из элементов более подробно.

Индексация. Базовая реализация модели MapReduce предполагает, что Map-узлы последовательно обрабатывают все доступные для них данные. Необходимо сузить объем доступных данных до некоторого подмножества и обрабатывать только его. Для этого, чтобы избежать последовательного чтения и проверки всех данных, необходимо знать, где конкретно на диске находятся нужные данные. Для построения индекса необходимо совершить один полный проход по всем данным. Так как на первой итерации любого итеративного алгоритма выборка еще не сформирована, используем первую итерацию в том числе для индексирования данных. В качестве алгоритма индексирования могут выступать различные алгоритмы в зависимости от того, насколько сильно ограничена доступная память. Рассматриваем ситуацию, на взгляд авторов наиболее соответствующую действительности, когда память дешевая и ее много, при этом действительно важным является время выборки данных по индексу. В связи с этим используем алгоритм индексирования на основе B-TREE.

Сортировка. Одним из основных, на взгляд авторов, факторов популярности модели MapReduce в современном мире является низкий порог входа к написанию распределенных алгоритмов. Разработчику достаточно реализовать всего две функции: Map и Reduce. Все остальные задачи по маршрутизации данных, обеспечению отказоустойчивости и контролю за ходом выполнения берет на себя фреймворк, реализующий модель MapReduce. При разработке архитектуры предлагаемой системы авторы старались руководствоваться этим же принципом.

Формировать выборку можно в двух режимах: ручном и автоматическом. В ручном режиме разработчик алгоритма при реализации функции Reduce сам указывает, какие именно данные следует обработать в следующей итерации. В автоматическом режиме от разработчика требуется вместе с указанием реализации функций Map и Reduce указать также реализацию функции SORT:

```
int function SORT( ReduceValue1, ReduceValue2 )
```

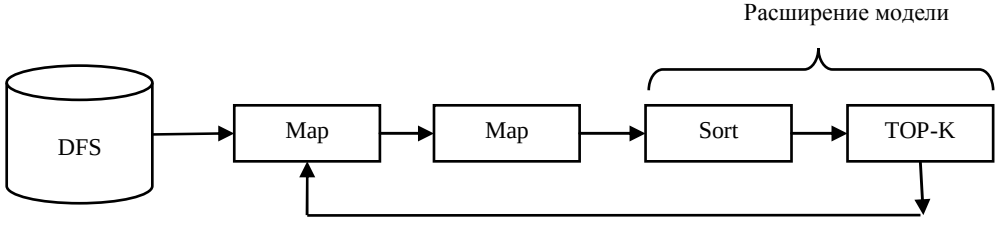

Рис. 2

В качестве параметров функции передаются 2 значения, полученные в результате выполнения функции Reduce. Результатом являются значения: –1 (rv1 < r v2), 0 (rv1 = = rv2) или 1 (rv1 = = rv2).

При автоматическом формировании выборки результаты работы Reduce-узлов сортируются согласно функции SORT и первые *K* элементов попадают в выборку данных для следующей итерации.

Алгоритм сортировки по сути представляет собой модифицированную версию алгоритма «сортировка слиянием». Модификация алгоритма связана с тем, что начиная со второй итерации в Reduce-узлах уже присутствуют отсортированные данные с прошлой итерации. Поэтому сортируются только те данные, значения которых изменились. Затем происходит их слияние с данными прошлой итерации.

Выборка K верхних элементов. Основной параметр, на который можно влиять, – максимальное количество элементов в выборке. Очередь может быть двух видов: локальная (для каждого Map-узла) и глобальная – общая для всех узлов. Общая очередь порождает проблему фрагментации объема работы между вычислительными узлами. Так одному узлу теоретически может достаться 99 % всех обрабатываемых данных. Наличие барьера [3] в модели MapReduce вынуждает Reduce-узлы до начала своей работы ожидать завершения всех Map-узлов. В связи с этим фрагментация выборки по узлам может быть серьезной проблемой производительности. Решение данной проблемы – введение локальных очередей для каждого узла. При этом размер очередей должен быть примерно одинаковым для повышения вероятности того, что все Map-узлы завершатся приблизительно в одно и то же время. Также стоит отметить, что одинаковый размер очереди далеко не всегда гарантирует равные временные затраты на обработку данных.

Значение параметра *K* является важным фактором, влияющим на производительность алгоритма. В частности, при *K*, близком к максимальному количеству существующих в узлах вершин, выборочная обработка ничем не отличается от полной, а при *K* близком к нулю затраты на старт новой итерации и постоянную передачу данных лишают этот подход выигрыша в производительности. В связи с этим оптимальное значение *K* является важным параметром в предлагаемом методе и требует дополнительного изучения.

Результаты. Контролирование данных, подлежащих обработке на следующей итерации, само по себе способно уменьшить время выполнения алгоритма за счет снижения количества операций дискового и сетевого ввода-вывода. При этом метод приоритетной обработки данных позволяет для ряда алгоритмов применить эвристические подходы, чтобы повысить их временную эффективность. Представленный эксперимент с алгоритмом поиска всех путей в графе при применении эвристического утверждения о том, что в первую очередь нужно идти по более коротким путям, позволяет увеличить эффективность алгоритма до 38 % в сравнении с простой выборочной обработкой и примерно в 7 раз по сравнению с реализацией на классической модели MapReduce.

Для изучения применимости данного подхода к другим алгоритмам реализуется программный фреймворк для поддержки выборочной обработки данных с приоритетами.

СПИСОК ЛИТЕРАТУРЫ

1. Митяков А. В., Татаринов Ю. С. Подходы к эффективному исполнению итеративных алгоритмов на модели MapReduce // Изв. СПбГЭТУ «ЛЭТИ». 2014. № 4. С. 8–15.

2. Митяков А. В., Татаринов Ю. С. MapReduce: Проблема полного перебора в итеративных алгоритмах и подход к решению // Тр. конф. «21 век: фундаментальная наука и технологии». М.: Spc Academic, 2014. Т. 2. С. 98–102.

3. Breaking the MapReduce stage barrier / A. Verma, N. Zea, B. Cho, I. Gupta, and R. Campbell // CLUSTER, Heraklion, Crete, Greece. 2010. Vol. 1, № 12. P. 235–244.

A. V. Mityakov, Yu. S. Tatarinov
Saint-Petersburg state electrotechnical university «LETI»

SAMPLING DATA PROCESSING – APPROACH TO SPEEDING UP ITERATIVE ALGORITHMS ON MAPREDUCE MODEL

In this paper we propose a method of sampling data to increase the performance of iterative algorithms on MapReduce model. A method proposed for selective data update based on priorities. Experiment data by the example of shortest paths search algorithm are presented.

MapReduce, iterative algorithms,sampling data