Теорема. Пусть задана структура $\Im = \langle E, \to \rangle$ и $S \subseteq E$ – подмножество априорных событий, полученных из внешней среды. Тогда минимальное множество причин событий S MinCause (S) должно обладать следующими свойствами:

1) *корректностью*: $\mathrm{MinCause}(S) \Rightarrow S$;

2) *оптимальностью*: для любого $S' : S' \Rightarrow S$ имеем $\mathrm{Rank}(S') \le \mathrm{Rank}(\mathrm{MinCause}(S))$.

Метод логической фильтрации на основе графов зависимостей, таких, как возможность формальной верификации, возможность компактного представления графа в виде булевых функций, а также данный метод, имеет хорошую теоретическую обоснованность. Недостатком данного метода является отсутствие адаптивности к изменениям структуры, но данный недостаток может быть устранен комбинированием данного метода с модельным подходом. Таким образом, метод логической фильтрации на основе графов зависимостей наиболее предпочтителен для разработки подсистемы логической фильтрации событий.

СПИСОК ЛИТЕРАТУРЫ

1. Dinesh Chandra Verma. Principles of Computer Systems and Network Management, Springer Science. New York: Business Media, 2009.

2. Masum Hasan, Binay Sugla, Ramesh Viswanathan. A Conceptual Framework for Network Managment Event Correlation and Filtering Systems //Proc. of the sixth IFIP/IEEE Intern. Symposium, Boston, 1999. MA. P. 5–9.

3. Rozemeijer E., Van Bon J., Verheijen T. Frameworks for IT Management: A Pocket Guide. Zaltbommel: Van Haren Publishing, 2007.

4. Kenneth R. Sheers. HP OpenView Event Correlation Services // Hewlett-Packard J. 1996. Oct. P. 6–7.

R. A. Nechitaylenko

DEMENSIONALITY REDUCTIONS METHODS FOR FILRETING INFORMATION IN GEOGRAPHICALLY DISTRIBUTED INFORMATION SYSTEM

The possibility of applying the method of logical filtering based on dependency graphs. We propose an algorithm logic filter events in geographically distributed control system based on dependency graphs.

Anomaly detection system, dependency graphs, logical filtering information, geographically distributed information system

УДК: 20.53.19, 28.23.13

И. В. Петухов

Распределенное выполнение алгоритмов построения деревьев решений с использованием библиотеки для анализа данных и концепции Map-Reduce

Описывается вариант распараллеливания алгоритмов построения деревьев решений с использованием библиотеки анализа данных на основе блочной структуры. Кроме того описывается способ взаимосвязи библиотеки и вычислительного кластера, основанного на технологии Map-Reduce. Описываются результаты экспериментального запуска алгоритма в разных средах.

Деревья решений, параллельные алгоритмы, распределенные вычисления

Современную жизнь невозможно представить без алгоритмов интеллектуального анализа данных, которые используются даже в обычных магазинах. Однако применение данных алгоритмов требует, как правило, большой вычислительной мощности в связи с тем, что для выявления закономерностей нужны большие объемы информации. Такие мощности не может предоставить локальная нераспределенная система. Даже с учетом темпов научно-технического прогресса такие задачи не решаются одним компьютером в одиночку. Для обработки огромного количества данных требуется параллельное выполнение алгоритмов data mining.

В настоящее время на рынке распределенных систем существует множество вариантов. Все они имеют свои достоинства и недостатки. Однако среди множества выделяется парадигма MapReduce. В настоящее время она является фактически стандартом де факто. Закрытый вариант реализации использует Google, которому принадлежат авторские права на эту технологию. Открытый вариант Apache Hadoop, на разработку которого было дано разрешение владельца авторских прав, используют такие компании IT-гиганты, как Facebook, Yahoo. Заказчиков привлекает стабильность и колоссальная масштабируемость системы, построенной на этой парадигме. Кроме того, это еще и выгодно заказчику, потому что можно построить кластер из недорогих отказонеустойчивых серверов. Даже при выходе из строя одного из узлов кластера кластер не теряет информацию и продолжает свою работу.

В данной статье будет рассмотрен алгоритм построения деревьев решений, промежуточный слой для соединения библиотеки анализа данных и кластера для распределенных вычислений, основанного на Map-Reduce, и результаты экспериментов по запуску этого алгоритма на одной машине и на кластере.

В качестве алгоритма для проведения экспериментов был выбран алгоритм C4.5 [1]. Это усовершенствованная версия алгоритма ID3 (Iterative Dichotomizer), использующая теоретико-информационный подход. Для выбора наиболее подходящего атрибута предлагается следующий критерий:

$$\mathrm{Gain}(\mathrm{X}) = \mathrm{Info}(\mathrm{T}) - \mathrm{Info}_{\mathrm{X}}(\mathrm{T}),$$

где Info(T) – энтропия множества T, а энтропия множества после разбиения:

$$\mathrm{Info}_{\mathrm{X}}(\mathrm{T}) = \sum_{i=1}^{n} \frac{|\mathrm{T}_i|}{|\mathrm{T}|} \mathrm{Info}(\mathrm{T}_i)$$

Алгоритм построения дерева, представленный в псевдокоде, выглядит следующим образом:

1. *корень = создать_узел (множество_векторов)*
2. *обработать_узел (корень)*
3.
4. *функция обработать_узел (узел)*
5. *если узел пуст*
6. *узел.выбрать_частый_класс_предка ()*
7. *узел.сделать_листом ()*
8. *иначе*
9. *узел.выбрать_критерий_рзабиения () // здесь применяется формула расчета критерия Gain*
10. *если разбиение возможно*
11. *подмножества = узел.разбить_на подмножества ()*
12. *цикл по подмножествам*
13. *потомок = узел.добавить_потомка (подмножество)*
14. *обработать_узел(потомок) // рекурсивный вызов*
15. *иначе*
16. *узел.сделать_листом ()*
17. *конец функции*

При проведении экспериментов использовалась библиотека для построения алгоритмов анализа данных DXelopes [2]. Для того чтобы запустить алгоритм с использованием библиотеки, он был декомпозирован на отдельные блоки-шаги. Кроме того, алгоритм был преобразован из рекурсивного в итеративный, так как параллелизация данного алгоритма в рекурсивном виде невозможна. В итеративном алгоритме для обработки узлов использовалась очередь, в которой хранились узлы дерева, ожидающие обработки. Итеративный алгоритм в псевдокоде выглядит следующим образом:

1. *корень = создать_узел (множество_векторов)*
2. *добавить_в_очередь (корень)*
3. *цикл, пока очередь не пуста*
4. *узел = взять_узел_из_очереди ()*
5. *если узел пуст*
6. *узел.выбрать_частый_класс_предка ()*
7. *узел.сделать_листом ()*
8. *иначе*
9. *узел. выбрать_критерий_разбиения ()*
10. *если разбиение возможно*
11. *подмножества = узел.разбить_на подмножества ()*
12. *цикл по подмножествам*
13. *потомок = узел.добавить_потомка (подмножество)*
14. *добавить_узел_в_очередь (потомок)*
15. *иначе*
16. *узел.сделать_листом ()*
17. *конец цикла*

Для декомпозированного алгоритма были рассмотрены 2 способа параллелизации вычислений:

1) по узлам;

2) по векторам.

Декомпозированный вариант алгоритма с параллелизацией *по узлам* представлен на рис. 1. При параллельной обработке узлов основной цикл алгоритма, в котором узлы берутся из очереди, выполняется параллельно. В этом блоке узлы берутся из очереди пока они там есть и обрабатываются. Сложность реализации данного алгоритма заключается в том, как дать понять алгоритму на этапе проверки очереди, что в ней больше нет узлов и работу можно завершить. Кроме того, при расчете вычислительной сложности оказалось, что этот алгоритм в худшем случае будет работать с такой же скоростью, как последовательный. Из-за сложности реализации такого типа параллелизации данный вариант не был реализован.

Рис. 1

Декомпозированный вариант алгоритма с параллелизацией *по векторам* представлен на рис. 2. При параллельной обработке векторов алгоритм работает как обычный последовательный алгоритм до тех пор, пока не дойдет до шага, где необходимо рассчитать критерий Gain(X) для каждого из атрибутов. Так как этот критерий рассчитывается с использованием всех векторов, принадлежащих текущему узлу, то в этом месте алгоритм параллельно берет по одному или несколько векторов и вычисляет критерий частично, а затем соединяет полученные части воедино. Затем алгоритм продолжает свою работу по обычному сценарию.

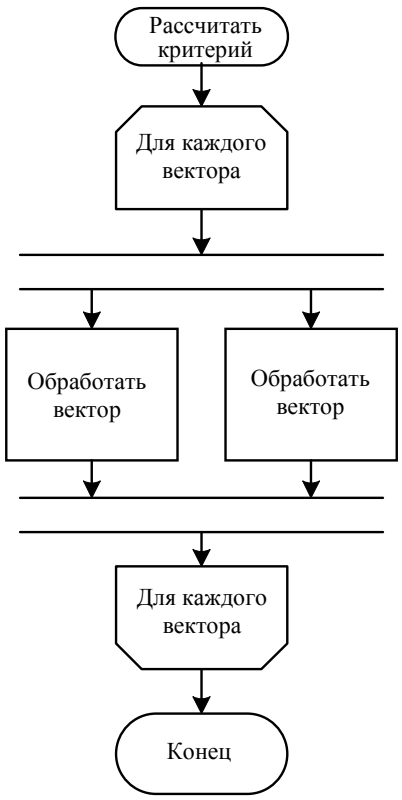

Рис. 2

Для связи воедино среды выполнения Map-Reduce [3] и библиотеки анализа данных таким образом, чтобы уже созданные и новые алгоритмы минимально зависели от среды выполнения и «не знали», на чем они выполняются, необходим промежуточный адаптер, который будет управлять выполнением алгоритмов в определенной среде. Адаптер связывает библиотеку и среду выполнения Map-Reduce. Он принимает от библиотеки необходимые команды и параметры алгоритма, а затем настраивает среду и передает ей на выполнение принятые команды. После чего забирает полученный результат и возвращает его для дальнейшей работы в библиотеку. Схема работы адаптера в псевдокоде выглядит таким образом:

1. *сохранить все векторы и настройки алгоритма в распределенной файловой системе*
2. *запустить параллельные шаги на кластере*
3. *дождаться завершения работы шагов*
4. *модель = модель после параллельных шагов*
5. *продолжить работу алгоритма*

При этом выполнение на кластере выглядит следующим образом:

1. *наборы_векторов* =

```
    подготовить_векторы ()
2.  шаги = загрузить_параллельные_шаги ()
3.  цикл по наборам
4.          модели += map (набор, шаги)
5.  модель = reduce e(модели)
6.  вернуть модель
7.
8.  функция map (векторы, шаги)
9.  шаги.выполнить (векторы)
10. вернуть измененную модель из шагов
11.
12. функция reduce (модели)
13. модель = модели.взять ()
14. модель.объединить (модели)
15. вернуть модель
```

Тесты проводились на кластере, состоящем из четырех узлов: один управляющий узел и 3 вычислительных. Сначала запускался последовательный алгоритм на одном узле, затем параллельный с использованием потоков (также на одном узле), и наконец, параллельный с использованием Map-Reduce – на всех узлах. Результаты запуска последовательного алгоритма отображены на рис. 3.

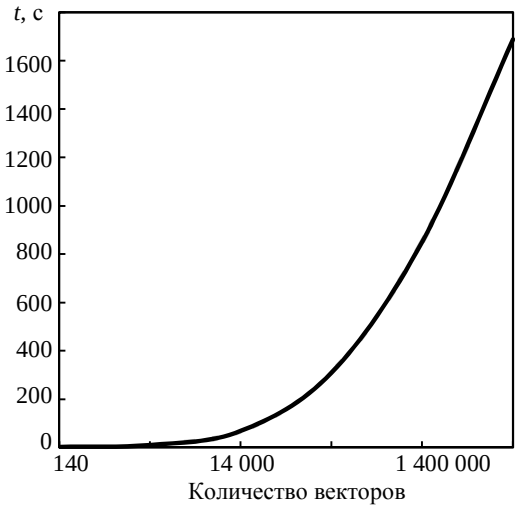

Рис. 3

В сравнении с последовательным алгоритмом параллельный алгоритм на потоках работает быстрее, и его скорость растет с увеличением количества векторов. На рис. 4 представлены 3 графика. Линией *1* отмечено время выполнения последовательного алгоритма в зависимости от количества векторов, линией *2* – параллельного алгоритма на двух потоках, линией *3* – на четырех потоках.

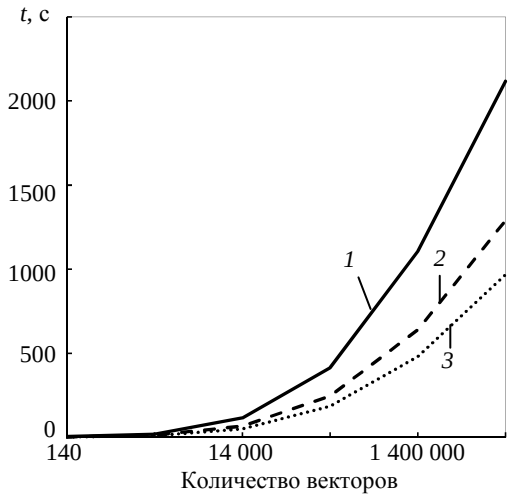

Рис. 4

При запуске параллельного алгоритма на кластере на малом количестве векторов отмечалось очень большое время выполнения по сравнению с последовательным алгоритмом, но при увеличении количества векторов время обработки сначала становится близким к последовательному, а затем уменьшается относительно него. Исключением является только работа алгоритма на одном узле кластера: из-за высоких накладных расходов время выполнения алгоритма проигрывает даже последовательному алгоритму. На рис. 5 представлены графики: *1* – работа алгоритма на кластере с одним вычислительным узлом; *2* – с двумя узлами; *3* – с тремя узлами; *4* – последовательный алгоритм.

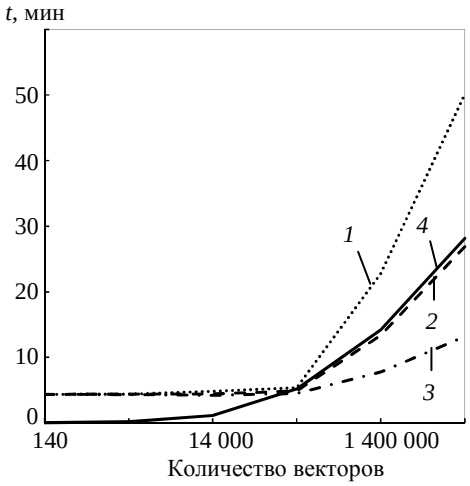

Рис. 5

В результате экспериментальных запусков было выяснено, что параллелизация по векторам для алгоритма построения деревьев решений при запуске на кластере с использованием Map-Reduce дает преимущество только при достаточ-

но большом количестве векторов (в данном случае – это миллионы векторов). В то же время, при малом количестве векторов из-за накладных расходов на пересылку данных между узлами видно значительное отставание от алгоритмов, выполнявшихся на одной машине.

СПИСОК ЛИТЕРАТУРЫ

1. Quinlan J. Ross. C4.5: Programs for Machine learning. Morgan Kaufmann Publishers, 1993.

2. Интеллектуальный анализ данных в распределенных системах / М. С. Куприянов, И. И. Холод, З. А. Каршиев, И. А. Голубев. СПб.: Изд-во СПбГЭТУ «ЛЭТИ», 2012.

3. White T. Hadoop: The Definitive Guide. USA: O'Reilly Media, 2009.

I. V. Petukhov

DISTRIBUTED EXECUTION OF DECISION TREE BUILDING ALGORITHMS BASED ON A DATA MINING LIBRARY WITH MAP-REDUCE

This article describes the method of paralleling decision tree building algorithms based on a data mining library that uses units structure for algorithm building and execution. Also it describes the method of interconnection of a library and a map-reduce computing cluster. Finally it describes results of execution of that algorithm with different environments.

Decision trees, parallel algorithms, distributed computations

УДК 681.32

А. В. Бессонов, К. А. Кноп, Ю. Т. Лячек, Ю. И. Попов

Определение минимальной ширины канала между парой компонентов при топологической трассировке

Предложен метод расчета минимальной ширины канала между парой многоконтактных компонентов при трассировке соединений в произвольных направлениях.

Печатный монтаж, топологическая трассировка, размещение компонентов

Гибкая топологическая трассировка в произвольных направлениях [1] имеет целый ряд преимуществ по сравнению с традиционной (под 90 и 45°).

Уже только отказ от преимущественных направлений трассировки позволяет [2]:

– уменьшить суммарную длину проводников;

– сократить площадь, занимаемую проводниками;

– понизить уровень перекрестных электромагнитных помех как за счет уменьшения длины проводников, так и за счет снижения уровня их параллельности;

– уменьшить риск рассогласования задержек в группе сигналов или в дифференциальном сигнале, обусловленный неоднородностью материала печатной платы;

– снизить риск коробления платы при воздействии тепловых нагрузок.

При гибкой трассировке фиксируется только относительное расположение проводников, межслойные переходы с помощью специальной процедуры перемещаются в оптимальные положения, а форма проводников вычисляется автоматически уже по окончании трассировки.

Автоматический расчет оптимальной формы проводников и автоматическая подвижка объектов (межслойных переходов, точек ветвления проводников и, при необходимости, компонентов) позволяют решать трудноразрешимые оптимизационные задачи, например построение деревьев Штейнера, сжатие топологического рисунка.

Задача выравнивания задержек сигналов при нефиксированном положении проводников значительно упрощается. Главное, что ее можно решать не последовательно, а параллельно [3].