РОССИЙСКАЯ ФЕДЕРАЦИЯ

(19) **RU** (11) **2 704 533**(13) **C1**

(51) МПК
G06F 8/00 (2018.01)
G06F 17/00 (2006.01)

ФЕДЕРАЛЬНАЯ СЛУЖБА
ПО ИНТЕЛЛЕКТУАЛЬНОЙ СОБСТВЕННОСТИ

(12) ОПИСАНИЕ ИЗОБРЕТЕНИЯ К ПАТЕНТУ

(52) СПК
G06F 8/31 (2019.08)

(21)(22) Заявка: **2019102187, 28.01.2019**

(24) Дата начала отсчета срока действия патента:
28.01.2019

Дата регистрации:
29.10.2019

Приоритет(ы):
(22) Дата подачи заявки: **28.01.2019**

(45) Опубликовано: **29.10.2019** Бюл. № **31**

Адрес для переписки:
127287, Москва, Старый Петровско-Разумовский пр-д, 1/23, стр. 1, Открытое акционерное общество "Информационные технологии и коммуникационные системы"

(72) Автор(ы):
Малов Алексей Викторович (RU)

(73) Патентообладатель(и):
Открытое акционерное общество "Информационные технологии и коммуникационные системы" (RU)

(56) Список документов, цитированных в отчете о поиске: **US 9612883 B2, 04.04.2017. US 2011/0208947 A1, 25.08.2011. US 20170046420 A1, 16.02.2017. US 2007/0255929 A1, 01.11.2007. RU 2411569 C2, 10.02.2011.**

(54) Способ распараллеливания программ в среде агентно-ориентированного программирования в вычислительной системе

(57) Реферат:

Изобретение относится к вычислительной технике. Технический результат заключается в расширении класса решаемых задач, включая задачи, которые не обладают списочным гомоморфизмом. В способе распараллеливания программ в среде агентно-ориентированного программирования в вычислительной системе получают входные данные и приложение для их обработки; формируют функции отображения, преобразующие исходные данные D из файлов или баз данных в набор списочных структур данных, содержащих списки атрибутов и списки векторов; осуществляют декомпозицию приложения в последовательность шагов; формируют специализированную приложением функцию разбиения исходных данных на части для работы каждого из множества программных агентов; формируют специализированную приложением функцию объединения результатов работы множества программных агентов, реализующих шаг приложения, в единые промежуточные модели знаний в виде набора списочных структур данных; формируют специализированную приложением функцию для формирования финальной выходной модели знаний из промежуточной модели знаний, полученной на последнем этапе.

RU 2704533 C1

RU 2704533 C1

RUSSIAN FEDERATION

FEDERAL SERVICE
FOR INTELLECTUAL PROPERTY

(19) **RU** (11) **2 704 533**(13) **C1**

(51) Int. Cl.
G06F 8/00 (2018.01)
G06F 17/00 (2006.01)

(12) ABSTRACT OF INVENTION

(52) CPC
G06F 8/31 (2019.08)

(21)(22) Application: **2019102187, 28.01.2019**

(24) Effective date for property rights:
28.01.2019

Registration date:
29.10.2019

Priority:
(22) Date of filing: **28.01.2019**

(45) Date of publication: **29.10.2019** Bull. № **31**

Mail address:
127287, Moskva, Staryj Petrovsko-Razumovskij pr-d, 1/23, str. 1, Otkrytoe aktsionernoe obshchestvo "Informatsionnye tekhnologii i kommunikatsionnye sistemy"

(72) Inventor(s):
Malov Aleksej Viktorovich (RU)

(73) Proprietor(s):
Otkrytoe aktsionernoe obshchestvo "Informatsionnye tekhnologii i kommunikatsionnye sistemy" (RU)

RU 2704533 C1

(54) METHOD FOR PARALLELING PROGRAMS IN AN AGENT-BASED PROGRAMMING ENVIRONMENT IN A COMPUTER SYSTEM

(57) Abstract:

FIELD: computer equipment.

SUBSTANCE: invention relates to the computer equipment. In the method for paralleling programs in an agent-based programming environment in a computer system, input data and an application for processing thereof are obtained; forming display functions converting original data D from files or databases into a set of list data structures, containing lists of attributes and lists of vectors; performing decomposition of application into sequence of steps; generating application specialized partitioning of source data into parts for operation of each of plurality of program agents; generating application-specific function of combining operation results of plurality of program agents implementing step of application into common intermediate knowledge models in form of set of list data structures; application-specific function is generated to generate a final output knowledge model from an intermediate knowledge model obtained at the last step.

EFFECT: technical result consists in the solved tasks class expansion, including tasks that do not have the list-based homomorphism.

1 cl

RU 2704533 C1

Область техники, к которой относится изобретение

Предполагаемое изобретение относится к вычислительной технике и, в частности, к области параллельных вычислений и может быть использовано для решения задач интеллектуального анализа данных с использованием среды агентно-ориентированного программирования.

Уровень техники

Реализация параллельного выполнения вычислений в общем случае является сложной задачей. В большинстве случаев распараллеливание выполнения задач осуществляется применительно к конкретному приложению и конкретной задаче. Такой подход является весьма трудоемким, и при этом может быть малоэффективным, по причине ошибок, которые может допустить разработчик при осуществлении.

Особенно это характерно для задач интеллектуального анализа данных (ИАД). Интеллектуальный анализ данных связан с анализом больших объемов данных, что, соответственно, требует использования значительных вычислительных ресурсов для обеспечения приемлемого времени выполнения анализа. Поэтому актуальным направлением в разработке и реализации алгоритмов ИАД является ориентация на выполнение алгоритмов анализа в распределенной вычислительной среде. В качестве отдельного направления можно выделить реализацию и распараллеливание алгоритмов ИАД в средах агентно-ориентированного программирования. Выделение отдельного направления обусловлено значительными отличиями использования агентно-ориентированного программирования и сред от традиционного императивного подхода. Поэтому данное направление должно поддерживаться специальными методами реализации алгоритмов ИАД с целью их распараллеливания.

Известен способ для упрощения передачи данных в распределенной и параллельной вычислительной системе (патент США №20110208947, приоритет от 25.08.2011 г.).

Способ, реализуемый на вычислительной машине, для упрощения передачи в распределенной параллельной вычислительной системе, который предполагает, что указанная распределенная параллельная вычислительная система включает в себя, по меньшей мере, один параллельный вычислительный блок для генерации промежуточных результатов, причем способ предполагает:

- определение, по меньшей мере, одного элемента из входных данных для параллельного вычислительного блока;
- создание отношения соответствия между, по меньшей мере, одним элементом и его индексами в соответствии с алгоритмом упрощения кодирования, в котором средний размер индексов меньше, чем средний размер, по меньшей мере, одного элемента;
- замену, по меньшей мере, одного элемента соответствующими индексами в соответствии с отношением соответствия;
- генерирование упрощенных промежуточных результатов параллельным вычислительным блоком на основе указанных индексов;
- передачу упрощенных промежуточных результатов.

При этом распределенная параллельная вычислительная система может представлять собой вычислительную систему, использующую модель MapReduce, в которой параллельный вычислительный блок является блоком отображения.

Способ может дополнительно содержать этап, на котором регистрируют отношение соответствия в центральном репозитории словаря.

Способ может также дополнительно содержать этап, на котором сохраняют отношение соответствия локально в параллельном вычислительном блоке.

Алгоритм упрощения кодирования может включать в себя алгоритм хеширования.

Этап замены в способе может включать замену, по меньшей мере, одного элемента соответствующими индексами, прежде чем блок параллельных вычислений выполнит параллельные вычисления.

Способ дополнительно содержит этапы, на которых:

- получают отношение соответствия между, по меньшей мере, одним элементом и индексами;
- заменяют индексы в обработанных результатах, которые получены на основании упрощенных промежуточных результатов, соответствующими элементами, по меньшей мере, одному элементу, в соответствии с указанным отношением соответствия.

При этом этап получения отношения соответствия предполагает получение отношения соответствия путем поиска записей в центральном репозитории словаря.

На этапе получения отношения соответствия может быть предусмотрено

- в ответ на запрос пользователя на часть обработанных результатов на основе части, по меньшей мере, одного элемента, участвующего в части обработанных результатов, получают часть указанного отношения соответствия, относящуюся к части, по меньшей мере, одного элемента,
- получают часть индексов, соответствующих части, по меньшей мере, одного элемента.

На этапе замены индексов в обработанных результатах, которые получены на основе упрощенных промежуточных результатов, на соответствующий, по меньшей мере, один элемент в соответствии с отношением соответствия:

- получают часть данных, относящуюся к запрашиваемой части обработанных результатов из обработанных результатов согласно части индексов;
- заменяют часть индексов в части соответствующих данных соответствующей частью, по меньшей мере, одного элемента в соответствии с частью отношения соответствия.

Однако, известный способ имеет недостатки. Известный способ позволяет увеличить быстродействие программ на вычислительной системе, использующую модель MapReduce. При этом сохраняются ограничения модели MapReduce по классам алгоритмов, которые можно распараллелить.

Известен также способ выполнения обработки данных в среде распределенной и параллельной обработки (патент США №9612883, приоритет от 06.12.2013 г.), который также иногда называют MapReduce.

Способ выполнения крупномасштабной обработки данных в среде распределенной и параллельной обработки, включающий:

- в совокупности соединенных между собой вычислительных систем, каждая из которых имеет один или несколько процессоров и память:
  - выполнение множества рабочих процессов;
  - выполнение не зависящего от приложения контрольного процесса в совокупности соединенных между собой вычислительных систем, для:
    - определение, для входных файлов, множества задач обработки данных, включая множество данных, специфицирующих задачи отображения из входных файлов, которые будут обработаны [для получения] промежуточных значений данных, и множества задач сокращения, определяющие промежуточные значения данных, которые будут переработаны в финальные выходные данные;
    - присвоение задачам обработки данных статуса незанятых в рабочих процессах:
      - использование совокупности не зависящих от приложения функций отображения, выполняемых 1-м подмножеством из множества рабочих процессов, чтобы считывать

части входных файлов, содержащих данные, и сформировать промежуточные значения данных посредством применения, по крайней мере, одной, определенной пользователем, специализированной приложением операции отображения к данным;

• сохранение промежуточных значений данных в совокупности промежуточных структур данных, распределенных среди множества соединенных между собой вычислительных систем; и

• использование совокупности не зависящих от приложения функций сокращения, отличных от совокупности не зависящих от приложения функций отображения, чтобы сформировать финальные выходные данные посредством применения, по крайней мере, одной, определенной пользователем, специализированной приложением операции сокращения с промежуточными значениями данных, причем совокупность не зависящих от приложения функций сокращения выполняется 2-м подмножеством из множества рабочих процессов;

причем совокупность не зависящих от приложения функций отображения и совокупность не зависящих от приложения функций сокращения независимы от специализированных приложением операторов и операций, включая, по крайней мере, одну определенную пользователем, специализированную приложением операцию отображения и, по крайней мере, одну, определенную пользователем, специализированную приложением операцию сокращения.

Способ может также включать применение операции разделения, по крайней мере, к подмножеству промежуточных значений данных, причем для каждого соответствующего промежуточного значения данных, по крайней мере, в подмножестве промежуточных значений данных, операция разделения определяет соответствующую промежуточную структуру данных множества промежуточных структур данных, в которых сохраняются соответствующее промежуточное значение данных.

Способ может также предусматривать, что соответствующая специализированная приложением операция отображения включает специализированную приложением операцию объединения для объединения начальных значений, сформированных посредством выполнения соответствующей специализированной приложением операцией отображения, так чтобы сформировать промежуточные значения данных.

В ходе выполнения способа количество задач отображения превышает значение множества процессов, которым контролирующий процесс может назначить задачи отображения; и контролирующий процесс поддерживает информацию о состоянии относительно задач отображения, ждущих назначения в рабочий процесс.

Описанный способ принят за прототип.

Основным недостатками известного способа являются то, что известный способ подходит только для некоторых задач ИАД. При этом существуют алгоритмы ИАД, например, Apriori, PageRank и др., для которых данный способ не походит, так как алгоритмы не обладают списочным гомоморфизмом. Также данный способ не раскрывает подходов к реализации алгоритмов ИАД в средах агентно-ориентированного программирования, в том числе как следует реализовать отдельные агенты для организации параллельного вычисления с применением сред агентно-ориентированного программирования при решении задачи (Gorlatch S. Extracting and Implementing List Homomorphisms in Parallel Program Development // Science of Computer Programming, 1999, v. 33, pp.1-27;

Dean J., Ghemawat S. MapReduce: Simplified Data Processing on Large Clusters // In Proceedings of the USENIX Symposium on Operating SystemsDesign & Implementation (OSDI), 2004, pp.137-147;

Harmelen F., Kotoulas S., Oren E., Urbani J. Scalable Distributed Reasoning using Map Reduce // Proceedings of the International Semantic Web Conference, 2009. v. 5823, pp.293-309).

Алгоритмами не обладающие списочным гомоморфизмом являются алгоритмы, которые дают разные результаты, в случае, если применять их к полному набору данных, и в случае, если применять алгоритм к частям исходных данных с последующим объединением результатов. Если разделить исходные данные алгоритма ИАД, не обладающего списочным гомоморфизмом, на части, и далее применить алгоритм отдельно к полученным частям, а после получения результатов для отдельных частей исходных данных объединить полученные результаты, то финальные результаты будут некорректными (Gibbons J. The Third Homomorphism Theorem // Journal of Functional Programming, 1996, v. 6, pp.657-665).

Раскрытие сущности изобретения

Техническим результатом является расширение класса решаемых задач, включая задачи (алгоритмы), которые не обладают списочным гомоморфизмом.

Для этого предлагается способ распараллеливания программ в среде агентно-ориентированного программирования в вычислительной системе, включающей совокупность соединенных между собой систем, каждая из которых имеет один или несколько процессоров и память, причем каждая система содержит установленное и сконфигурированное программное обеспечение, выполненное с возможностью

- передачи сообщений с данными из множества программных агентов главному программному агенту,
- формирования множества программных агентов с помощью главного программного агента, и последующего запуска множества программных агентов в системах для выполнения,
- приема данных в главном программном агенте из множества программных агентов,
- запуска главного программного агента в системе для выполнения,
- выполнения множества программных агентов, в том числе и главного программного агента, на любом из процессоров системы,
- считывания из входных файлов или баз данных множества исходных данных, специфицирующих задачу;

способ заключается в том, что

- получают входные данные и приложение для их обработки;
- формируют функции отображения, преобразующие исходные данные D из файлов или баз данных в набор списочных структур данных, содержащих списки атрибутов и списки векторов;
- осуществляют декомпозицию приложения в последовательность шагов;
- формируют специализированную приложением функцию разбиения исходных данных на части для работы каждого из множества программных агентов;
- формируют специализированную приложением функцию объединения результатов работы множества программных агентов, реализующих шаг приложения, в единые промежуточные модели знаний в виде набора списочных структур данных;
- формируют специализированную приложением функцию для формирования финальной выходной модель знаний из промежуточной модели знаний, полученной на последнем этапе;
- формируют функцию приема данных главного программного агента, выполненную с возможностью принимать данные из множества программных агентов;
- формируют с использованием среды агентно-ориентированного программирования главный программный агент, выполненный с возможностью выполнения

○ формирования множества программных агентов, реализующих определенный шаг приложения, в главном программном агенте, и последующего запуска множества программных агентов в системах для выполнения;

○ функции разбиения исходных данных на части для работы одного из множества программных агентов;

○ передачи данных во множество программных агентов;

○ функции объединения результатов работы множества программных агентов в единую промежуточную модель знаний;

○ приема данных главного программного агента;

● формируют программные агенты, выполненные с возможностью

○ выполнять шаг приложения;

○ выполнять функцию передачи результатов работы в главный программный агент;

● запускают главный программный агент и с помощью него выполняют следующие действия

○ последовательно выполняют шаги приложения, начиная с первого, на множестве программных агентов, причем на каждом шаге

■ преобразуют с помощью функции отображения исходные данные D из файлов или баз данных в набор списочных структур данных, содержащих списки атрибутов и списки векторов;

■ разделяют данные на части для работы каждого из множества программных агентов;

■ запускают множество программных агентов;

■ выполняют шаг приложения в программных агентах, реализующих шаг приложения;

■ передают после завершения шага промежуточные структуры данных из множества программных агентов в главный программный агент;

■ принимают данные из множества программных агентов в главном программном агенте;

■ объединяют с помощью функций объединения результаты работы множества программных агентов в единую промежуточную модель знаний;

○ после выполнения всех шагов приложения объединяют полученные на последнем шаге единые промежуточные модели знаний в финальную выходную модель знаний;

○ получают финальную выходную модель знаний.

Предлагаемый способ подразумевает подход к распараллеливанию программ, реализующих алгоритмы ИАД в средах агентно-ориентированного программирования. Способ подразумевает использование среды агентно-ориентированного программирования.

Агентно-ориентированное программирование - это подход к программированию, который предполагает проектирование программных систем, базирующихся на программных агентах.

В объектно-ориентированном программировании объекты классов взаимодействуют друг с другом через интерфейс. Интерфейс объекта класса определен в соответствующем классе. При этом обращение к объекту, в свою очередь, может приводить к обращению к другому объекту.

Агентно-ориентированное программирование предусматривает взаимодействие программного агента с внешней средой, которая может быстро изменяться. Под агентом подразумевается программный или аппаратный компонент, который может выполнять поставленные перед ним задачи. Предполагается, что агент способен воспринимать изменения внешней среды, взаимодействовать с нею, выполнять действия, изменяющие

внешнюю среду. Пользователи, другие программные агенты также входят в понятие внешней среды агента.

Таким образом, программный агент является значительно более сложным элементом по сравнению с объектом класса в объектно-ориентированном программировании. Поэтому агентно-ориентированный подход позволяет описывать сложные программно-аппаратные системы на новом уровне.

Для решения вычислительных задач программные агенты можно представить в виде вычислительных акторов (Burgin M. Systems, Actors and Agents: Operation in a multicomponent environment [Режим доступа: https://pdfs.semanticscholar.org/97bb/93c8a1850a9fcdd3e1ef9a441 c491e0976da.pdf, своб., язык: английский). В рамках данного похода программный агент имеет возможность создавать новые программные агенты в среде агентно-ориентированного программирования, посылать сообщение другим программным агентам по известному адресу, получать сообщения от других программных агентов.

Способ предполагает разделение программ, реализующих алгоритмы ИАД, на отдельные шаги, каждый из которых может реализовывать программный агент, который соответствует определенному шагу. При этом на каждом шаге программные агенты могут работать в параллельном режиме. Выполнение в параллельном режиме каждого отдельного шага, а не всего алгоритма сразу, позволяет, в отличие от MapReduce, применять данный способ для распараллеливания алгоритмов, которые не обладают списочным гомоморфизмом.

Алгоритмы ИАД на входе принимают набор входных данных, а результатом их работы является финальная выходная модель знаний. Таким образом, каждый алгоритм ИАД задает функцию, так как получает на вход набор входных данных и выдает результат применения алгоритма к входным данным в виде финальной выходной модели знаний. Следовательно, алгоритм ИАД можно представить в виде функции, которой в качестве аргумента передается набор входных данных D, а возвращаемым значением является построенная финальная выходная модель знаний M. Таким образом, алгоритм ИАД можно представить в виде последовательности функций, представляющих шаг алгоритма.

В ходе работы алгоритмы ИАД анализируют набор данных и формируют промежуточную модель знаний на каждом шаге. Построенная промежуточная модель знаний передается на следующий этап. Каждый шаг алгоритма ИАД должен принимать набор данных и промежуточную модель знаний в качестве входных аргументов. Результатом работы шага является новая промежуточная модель знаний.

Таким образом, алгоритм ИАД можно представить в виде последовательной работы программных агентов, реализующих шаг алгоритма.

$$D \rightarrow A1 \rightarrow \ldots \rightarrow Ai \rightarrow \ldots \rightarrow An, \quad (1)$$

где Ai - программный агент, реализующий i-й шаг алгоритма ИАД, принимает на вход набор исходных данных (D) алгоритма ИАД и промежуточную модель знаний. В (1) стрелками условно обозначена передача исходных данных (D) алгоритма ИАД и промежуточной модели знаний от одного программного агента к следующему.

При этом программный агент A_1, реализующий 1-й шаг алгоритма ИАД, принимает на вход только набор исходных данных (D) алгоритма ИАД в качестве входных аргументов.

Результатом работы каждого программного агента является промежуточная модель знаний, которая используется на следующем шаге, а результатом работы программного агента An является финальная выходная модель знаний.

При этом при реализации алгоритма ИАД один и тот же шаг или группа шагов могут выполняться рекурсивно на соответствующих им агентах до достижения определенного условия, в зависимости от алгоритма.

Здесь и далее для примеров будет использоваться нотация языка программирования Java [https://docs.oracle.com/javase/tutorial/]. Также будут использоваться концепции, подходы и готовые классы среды агентно-ориентированого программирования JADE [http://jade.tilab.com].

Набор исходных данных (D) алгоритма ИАД удобно представить в виде списочных структур данных. На языке Java набор исходных данных (D) можно представить в виде класса, содержащего в себе в качестве членов список атрибутов (at-tributes_list), список векторов (vectors_list) и список всех возможных классов целевого атрибута (class_list):

```
public class D
{
        public ArrayList<ArrayList> attributes_list;
        public ArrayList<ArrayList> vectors_list;                    (2)
        public ArrayList class_list;
}
```

Список из K атрибутов attributes_list можно задать следующим образом

```
ArrayList<ArrayList> attributes_list =
{{
        add(attr0);
        …
        add(attrK);
}};
```

где

attr<i> - список возможных значений i-го атрибута. Ниже приведен пример для атрибута 0, который может иметь пять значений:

ArrayList attr0=new ArrayList(Arrays.asList(0.5, 3.0, 4.5, 5.4, 2.5));

Список из L векторов можно определить следующим образом

```
ArrayList<ArrayList> vectors_list =
{{
        add(vector0);
        …
        add(vectorL);
}};
```

где vector <i> - список конкретных значений каждого атрибута и значения целевого атрибута.

Значение i-го атрибута располагается на i-м месте в списке. Если значение целевого атрибута неизвестно, то в языке Java оно может быть задано символом null. Значение

целевого атрибута можно расположить в конце списка. Ниже приведен пример для вектора 0, в котором первые четыре значения соответствуют значениям атрибутов, а последнее значение соответствует значению целевого атрибута:

ArrayList vector0=new ArrayList(Arrays.asList(1.5, 4.0, 4.2, 5.5, 8.5));

Структура промежуточной (MP) и финальной выходной модели знаний (M) зависит от алгоритма и функции, реализующей ИАД. Промежуточную (MP) и финальную выходную модель знаний (M) удобно представлять в виде списочных структур данных.

Например, распространенной среди алгоритмов ИАД является модель знаний в виде деревьев решений. Для деревьев решений узлы дерева и соответствующие им множества, можно представить в виде списков. В общем виде промежуточную модель знаний можно представить в виде структуры, содержащей параметры текущего шага step_context и списка правил rules_list:

```
public class MP
{
    public StepContexClass step_context;                    (3)
    public ArrayList<ArrayList> rules_list;
}
```

Параметры текущего шага step_context могут содержать, например, список атрибутов, которые еще не использовались для построения модели знаний, и, следовательно, должны быть обработаны на следующих шагах.

Приложения ИАД в основном характеризуются параллелизмом данных. В этом случае для реализации параллельных вычислений требуется начальное разделение данных и последующее объединение результатов, полученных в параллельно работающих программных агентах, реализующих шаг алгоритма ИАД. Выполнение в параллельном режиме каждого отдельного шага, а не всего алгоритма сразу, позволяет, в отличие от MapReduce, применять данный способ для распараллеливания алгоритмов, которые не обладают списочным гомоморфизмом.

Для параллельного выполнения приложения подход предполагает разделение данных и объединения результатов, полученных в параллельно работающих программных агентах, в главном программном агенте. Эти операции позволят распараллеливать приложение в среде агентно-ориентированного программирования и многоагентной системе JADE. При этом правильно сконфигурированная JADE система, установленная на совокупность соединенных между собой систем, каждая из которых имеет один или несколько процессоров и память, обладает возможностью

- передачи сообщений с данными из множества программных агентов главному программному агенту,
- формирования множества программных агентов, в главном программном агенте, и последующего запуска множества программных агентов в системах для выполнения,
- приема данных в главном программном агенте из множества программных агентов,
- запуска главного программного агента в системе для выполнения,
- выполнения множества программных агентов, в том числе и главного программного агента, на любом из процессоров системы
- считывания из входных файлов или баз данных множества исходных данных, специфицирующих задачу;

В состав задач и функций, который нужно реализовать для параллельного выполнения приложения входят:

● специализированная приложением функция разбиения исходных данных на части для работы одного из множества программных агентов;

● специализированная приложением функция объединения результатов работы множества программных агентов в единые промежуточные модели знаний в виде набора списочных структур данных;

● специализированная приложением функция для формирования финальной выходной модели знаний из промежуточной модели знаний, полученной на последнем этапе;

● функция приема данных главного программного агента, выполненная с возможностью принимать данные из множества программных агентов;

● формирование главного программного агента, с использованием среды агентно-ориентированного программирования, выполненного с возможностью выполнения

○ формирования множества программных агентов, реализующих определенный шаг приложения, в главном программном агенте, и последующего запуска множества программных агентов в системах для выполнения;

○ функции разбиения исходных данных на части для работы одного из множества программных агентов;

○ передачи данных во множество программных агентов;

○ функции объединения результатов работы множества программных агентов в единую промежуточную модель знаний; о приема данных главного программного агента.

● формирование программных агентов, выполненных с возможностью

○ выполнять шаг приложения;

○ выполнять функцию передачи результатов работы в главный программный агент.

Исходные данные D из файлов или баз данных должны быть считаны главным программным агентом в набор списочных структур данных, содержащих списки атрибутов и списки векторов. Считывание из входных файлов или баз данных множества исходных данных, специфицирующих задачу, должна обеспечивать сконфигурированная JADE система. Так как главный программный агент может быть запущен на любом процессоре, входящим в систему, то доступ к файлам или базам данных должен быть у каждой системы из совокупности соединенных между собой систем.

С целью упрощения реализации для некоторых случаев каждый программный агент, реализующий шаг приложения, может считать самостоятельно входные данные из файлов или баз данных. Главный агент перед запуском множества программных агентов определяет количество процессоров в системе. На основе полученного числа, вычисляет диапазон данных, который должен обработать каждый из множества программных агентов, реализующих шаг приложения, и использует полученные данные при выполнении шага. При этом программный агент, реализующий шаг приложения, может считать не все данные, а только ту часть данных, которая предназначена ему для обработки. Диапазон данных для обработки программный агент может получить от главного программного агента, а также может вычислить его с помощью функции разделения исходных данных на части, в зависимости от удобства реализации.

При запуске любого программного агента в системе ему присваивается уникальный адрес. На одной системе, входящую в совокупность соединенных между собою систем, может быть запущено несколько программных агентов, в зависимости от числа процессорных ядер на машине.

Осуществление изобретения

Реализация предложенного способа может быть осуществлена в вычислительной системе, работающей, например, под управлением операционной системы Linux.

Для реализации способа достаточно реализовать выполнение предложенных операций на поддерживаемом средой языке программирования, для среды агентно-ориентированного программирования JADE, данным языком является язык программирования Java.

Реализовать действия предложенного способа в составе программы или функции может специалист в области программирования (программист), зная выполняемые действия.

В качестве примера рассмотрен алгоритм C4.5 (Hssina B. et al. A comparative study of decision tree ID3 and C4.5, статья по адресу: http://saiconference.com/Downloads/SpecialIssueNo10/Paper_3-A_comparative_study_of_decision_tree_ID3_and_C4.5.pdf;

Деревья решений - C4.5 математический аппарат. Часть 1, статья по адресу: https://basegroup.ru/community/articles/math-c45-part1).

Для реализации использовалась ОС Linux, среда агентно-ориентированного программирования JADE, язык Java. Алгоритм базируется на теоретико-информационном подходе и является усовершенствованной версией алгоритма ID3 (Iterative Dichotomizer). Алгоритм включает четыре основных шага:

- формируется корневой узел дерева, с которым ассоциируется все множество векторов V из набора исходных данных D; в (2) множество векторов V задает vectors_list;
- для каждого формируемого узла дерева решений и ассоциированного с ним множества векторов Vi выбирается атрибут A для разбиения,
- пусть выбранный атрибут A принимает n значений A1, A2 … An, тогда выбранный атрибут A разделяет множество векторов Vi на подмножества Vi1, Vi2 … Vin, при A равном, соответственно, A1, A2 … An,
- шаги повторяются до тех пор, пока в каждом формирующемся узле дерева не окажутся векторы, соответствующие одному классу, либо пустое множество векторов.

На втором шаге для выбора наиболее подходящего атрибута используется следующий критерий:

$$Gain(A) = Info(V_i) - Info_A(V_i)$$

$$Info_A(V_i) = \sum_{r=1}^{n} \frac{|V_{ir}|}{|V_i|} Info(V_{ir}) \qquad (4)$$

$$Info(V_i) = -\sum_{j=1}^{t} \left(\frac{freq(c_j, V_i)}{|V_i|} \right) \log \left(\frac{freq(c_j, V_i)}{|V_i|} \right),$$

где A - атрибут по которому производится разбиение на текущем шаге;

n - количество значений независимого атрибута A;

t - количество значений зависимого атрибута - классов;

c_j - класс, для которого производится вычисление;

$freq(c_j, V_i)$ - количество векторов из множества V_i, относящихся к классу c_j;

V_{ir} - подмножество данных V_i, у которых атрибут A имеет r-е значение;

V_i - подмножество векторов ассоциированных с текущим узлом дерева.

Здесь и далее для примеров будет использоваться нотация языка программирования Java (https://docs.oracle.com/javase/tutorial/).

Приложение, реализующее алгоритм C4.5 может быть представлено в виде:

```
public void Algorithm (D d, M m)                                    (5)
{
    MP mp = new MP();
    while( ! is_finished(d, mp) )  //достигнут конец алгоритма?
    {
        for ( ArrayList cur_attr : get_not_handled_attr_list(d, mp) )
        {
            calculate_attr_gain(cur_attr, d, mp);
        }
        set_best_attr(mp);
    }
    build_final_model(mp, m);   //построить выходную модель знаний
}
```

Функция Algorithm строит корневой узел дерева, с которым ассоциируется все множество векторов V из набора исходных данных D и выставляет корневой узел в качестве обрабатываемого узла. Данные действия осуществляются в конструкторе класса MP. Далее построенная промежуточная модель знаний MP передается в качестве параметра функциям в теле цикла while для выполнения шага.

Шаги приложения реализуются в теле цикла while. После выполнения тела цикла while получает, промежуточную модель знаний MP. Далее функция set_best_attr из step_context, определенном в (3) получает список вычисленных критериев (4) для каждого атрибута, выбирает наиболее подходящий атрибут A, добавляет соответствующие значениям атрибута A узлы дерева решений к промежуточной модели MP. После этого функция Algorithm проверяет, существуют ли узлы дерева решений в промежуточной модели, для которых требуется раскрытие с помощью функции is_finished(d, mp). Если существую, то производится новый шаг в теле цикла while. Если такие узлы отсутствуют, то возвращает в качестве результата финальную выходную модель знаний M.

get_not_handled_attr_list - функция, которая возвращает список атрибутов, которые еще не добавлены к дереву решений.

calculate_attr_gain - функция, которая вычисляет $Gain(A)$ из (4) для обрабатываемого атрибута A. Выполняет итерацию по возможным значениям атрибута A для вычисления $Info_A(V_i)$ из (4), а также, выполняет итерацию по всем классам для вычисления $Info(V_i)$ из (4), добавляет полученные результаты в промежуточную модель знаний.

build_final_model - специализированная приложением функция для формирования финальной выходной модели знаний из промежуточной модели знаний, полученной на последнем этапе.

Входные данные (D) для алгоритма C4.5 можно представить в виде таблицы, каждая строка которой содержит значения входных атрибутов и соответствующий им класс целевого атрибута.

Для реализации алгоритма C4.5 введем следующее представление для входных данных

```
public class D
{
        public ArrayList<ArrayList> attributes_list;
        public ArrayList<ArrayList> vectors_list;
        public ArrayList class_list;
}
```

где attributes_list - список атрибутов,
vectors_list - список векторов,
class_list - список всех возможных классов целевого атрибута.

Финальная выходная модель знаний (M) для алгоритма C4.5 представляется в виде дерева решений. Промежуточная модель знаний (MP) для алгоритма C4.5 представляется в виде неполного дерева решений rules_list в (3). Также промежуточная модель знаний включает список обработанных на данный момент атрибутов, список вычисленных на последнем шаге критериев (4) для атрибутов, обрабатываемый на данном шаге узел дерева решений. Данная часть промежуточной модели обозначена как step_context в (3).

Дерево решений можно представить в виде структуры вложенных списков.

```
public class TreeNode
{
        public TreeNode(Integer a, HashMap<AttrVal, TreeNode> s_n)
        {
                attr_number = a;
                sub_nodes = s_n;
        }
        public Integer attr_number;
        public HashMap<AttrVal, TreeNode> sub_nodes;
}
```

В классе TreeNode член класса attr_number содержит номер атрибута, которому соответствует данный узел. Член класса sub nodes содержит список всех дочерних узлов для данного узла. Таким образом, финальную выходную модель знаний M можно определить

```
TreeNode M = node_root;                    (6)
```

где node_root - корневой узел дерева.

Корневой узел дерева в свою очередь можно представить в виде:

```
TreeNode node_root=new TreeNode(attr_N,new HashMap<AttrVal,TreeNode>()
```

```
{{
        put(attr_N_value_1, node_1);
        ...
        put(attr_N_value_K, node_K);
}});
```

где attr_N - номер атрибута, который был выбран для построения корневого узла дерева решений, данный атрибут имеет K значений

attr_N_value_1 - значения атрибута, соответствующее следующему узлу дерева node_1

attr N_value_K - значения атрибута, соответствующее следующему узлу дерева node_K.

Узлы второго яруса дерева решений node_1… node_K определяются аналогично узлу node_root. Таким же образом определяются узлы всех последующих ярусов дерева решений.

Для выполнения приложения в среде агентно-ориентированного программирования, реализацию алгоритма C4.5 в (5) требуется трансформировать и представить в виде реализации программных агентов, реализующих шаги приложения и главного программного агента. Для реализации главного программного агента требуется добавление функций разделения данных и объединения результатов, полученных в разных программных агентах, реализующих шаг приложения.

Рассмотрим вариант параллельного приложения, выполняющего параллельную обработку по атрибутам, реализованного с применением предложенного подхода.

Реализация класса главного агента с использованием базовых классов среды агентно-ориентированного программирования JADE будет иметь следующий вид:

```
import jade.core.Agent;
import jade.core.behaviours.OneShortBehaviour;
/*импорт других требуемых библиотек*/


public class AgentMainBehaviour extends OneShotBehaviour
```

```
{
    public void action()
    {
        D d;
        load_data(d);
        MP mp = new MP();
        while( ! is_finished(d, mp ) )  //достигнут конец алгоритма?
        {
            ArrayList<ArrayList> attr_list_for_handle =
                                        get_not_handled_attr_list(d, mp);
            //Инициализация программных агентов, реализующих
            //шаг приложения и отсылка списка
            //атрибутов, требующих обработки для вычислений
            divide_attr_parall (attr_list_for_handle);
            merge_attr_parall( mp);
            set_best_attr(mp);
        }
        M m;
        build_final_model(mp, m);   //построить выходную модель знаний
        //Действия с финальной выходной моделью
        //M – сохранение в БД, файлы и т.п.
        myAgent.doDelete(); //Завершение работы агента
        return;
    }
    .....
    //Вспомогательные методы класса
}

public class AgentMain extends Agent
{
    public void setup()
    {
```

```
addBehaviour(new AgentMainBehaviour (this));
        }
    }
```

Класс AgentMain является реализацией главного программного агента для параллельного выполнения алгоритма с распараллеливанием по атрибутам в среде JADE. Класс наследуется от стандартного класса Agent среды JADE. В методе setup() данного класса происходит установка класса поведения AgentMainBehaviour, реализующего в терминах среды JADE поведение программного агента.

Класс AgentMainBehaviour наследуется от стандартного класса OneShotBehaviour среды JADE. AgentMainBehaviour в методе action() реализует задачу, которую выполняет главный программный агент. Метод action() вызывается один раз после старта агента, при окончании работы метода агент прекращает выполнение.

Функции is_finished, get_not_handled_attr_list, set_best_attr, build_final_model рассмотрены ранее. Функция load_data загружает исходные данных из файлов или баз данных в экземпляр d класса D. divide_attr_parall - специализированная приложением функция разбиения исходных данных на части для работы одного из множества программных агентов. Функция является членом класса AgentMainBehaviour. В функции перед запуском множества программных агентов определяется количество машин в системе, на которых могут выполняться агенты. На основе полученного числа, вычисляет диапазон данных, который должен обработать каждый из множества программных агентов, реализующих шаг приложения. Далее в процессе ее работы запускается множество программных агентов в среде JADE и сохраняется информация о запущенных агентах в переменных членах класса. Также передает каждому из множества запущенных программных агентов список атрибутов, предназначенный для обработки программным агентом. В примере для упрощения отсутствует реализация. Следует отметить, что реализация зависит от среды агентно-ориентированного программирования.

merge_attr_parall - специализированная приложением функция объединения результатов работы множества программных агентов в единую промежуточную модели знаний (mp) в виде структуры (3). В процессе ее работы происходит считывание результатов работы каждого из множества программных агентов, реализующих шаг приложения. В промежуточную модель (MP) записывается значение $Gain(A)$ из (4) для каждого обрабатываемого атрибута.

Если достигнут конец алгоритма, то функция build_final_model формирует финальную выходную модель знаний (m) в виде структуры (6).

Каждый из множества программных агентов, реализующих шаг приложения, работает с предназначенной ему частью исходных данных. В конце своей работы передает результаты работы, представленные списком значений $Gain(A)$ из (4) для каждого обработанного агентом атрибута, в главный программный агент.

Реализация класса программного агента, реализующего шаг приложения, с использованием базовых классов среды агентно-ориентированного программирования JADE может иметь следующий вид:

```
import jade.core.Agent;
import jade.core.behaviours.OneShortBehaviour;
/*импорт других требуемых библиотек*/

public class AgentStepBehaviour extends OneShotBehaviour
{
	public void action()
	{
		D d;
		load_data(d);
		MP mp = new MP();

		for ( ArrayList cur_attr : receive_attr_list_for_handle() )
		{
			calculate_attr_gain(cur_attr, d, mp);
		}
		send_results_to_main_agent(mp);

		myAgent.doDelete(); //Завершение работы агента
		return;
	}
	.....
	//Вспомогательные методы класса
}

public class AgentStep extends Agent
{
	public void setup()
	{
		addBehaviour(new AgentStepBehaviour (this));
	}
}
```

Стандартные классы Agent, OneShotBehaviour среды JADE, использованные для реализации класса программного агента, а также другие используемые концепции среды, функция calculate_attr_gain и предназначение функции load_data были рассмотрено выше. Метод receive_attr_list_for_handle класса AgentStepBehaviour получает от главного программного агента список атрибутов, предназначенный для обработки данным программным агентом, и возвращает его в качестве результата.

Функция send_results_to_main_agent служит для передачи результатов работы программного агента в главный программный агент. Ее реализация в примере для упрощения отсутствует.

Таким образом, рассмотрена реализация распараллеливания по атрибутам алгоритма C4.5 на основе предложенного способа.

Если применить концепцию MapReduce без доработок к последовательному варианту реализации (5) алгоритма C4.5 для распараллеливания по атрибутам, то на фазе Map исходный набор атрибутов будет разделен на N частей, где N это количество рабочих процессов в системе. Каждая из N частей предназначена для обработки одним процессом на фазе Reduce. Таким образом, после завершения Reduce фазы будет получено N деревьев решений, при этом отсутствует возможность получить из этих N деревьев решений финальную выходную модель знаний, так как алгоритма C4.5 не обладает списочным гомоморфизмом. Отсюда следует, что применение концепции MapReduce к данному алгоритму требует дополнительных методов. Таким образом, предложенный способ позволяет получить корректный результат, а применение концепции MapReduce к данному алгоритму без дополнительных методов не позволяет.

(57) Формула изобретения

Способ распараллеливания программ в среде агентно-ориентированного программирования в вычислительной системе, включающей совокупность соединенных между собой систем, каждая из которых имеет один или несколько процессоров и память, причем каждая система содержит установленное и сконфигурированное программное обеспечение, выполненное с возможностью

передачи сообщений с данными из множества программных агентов главному программному агенту,

формирования множества программных агентов с помощью главного программного агента, и последующего запуска множества программных агентов в системах для выполнения,

приема данных в главном программном агенте из множества программных агентов,

запуска главного программного агента в системе для выполнения,

выполнения множества программных агентов, в том числе и главного программного агента, на любом из процессоров системы,

считывания из входных файлов или баз данных множества исходных данных, специфицирующих задачу;

способ заключается в том, что

получают входные данные и приложение для их обработки;

формируют функции отображения, преобразующие исходные данные D из файлов или баз данных в набор списочных структур данных, содержащих списки атрибутов и списки векторов;

осуществляют декомпозицию приложения в последовательность шагов;

формируют специализированную приложением функцию разбиения исходных данных на части для работы каждого из множества программных агентов;

формируют специализированную приложением функцию объединения результатов работы множества программных агентов, реализующих шаг приложения, в единые промежуточные модели знаний в виде набора списочных структур данных;

формируют специализированную приложением функцию для формирования финальной выходной модели знаний из промежуточной модели знаний, полученной на последнем этапе;

формируют функцию приема данных главного программного агента, выполненную с возможностью принимать данные из множества программных агентов;

формируют с использованием среды агентно-ориентированного программирования главный программный агент, выполненный с возможностью выполнения

формирования множества программных агентов, реализующих определенный шаг приложения, в главном программном агенте, и последующего запуска множества программных агентов в системах для выполнения;

функции разбиения исходных данных на части для работы одного из множества программных агентов;

передачи данных во множество программных агентов;

функции объединения результатов работы множества программных агентов в единую промежуточную модель знаний;

приема данных главного программного агента;

формируют программные агенты, выполненные с возможностью выполнять шаг приложения;

выполнять функцию передачи результатов работы в главный программный агент;

запускают главный программный агент и с помощью него выполняют следующие действия:

последовательно выполняют шаги приложения, начиная с первого, на множестве программных агентов, причем на каждом шаге

преобразуют с помощью функции отображения исходные данные D из файлов или баз данных в набор списочных структур данных, содержащих списки атрибутов и списки векторов;

разделяют данные на части для работы каждого из множества программных агентов;

запускают множество программных агентов;

выполняют шаг приложения в программных агентах, реализующих шаг приложения;

передают после завершения шага промежуточные структуры данных из множества программных агентов в главный программный агент;

принимают данные из множества программных агентов в главном программном агенте;

объединяют с помощью функций объединения результаты работы множества программных агентов в единую промежуточную модель знаний;

после выполнения всех шагов приложения объединяют полученные на последнем шаге единые промежуточные модели знаний в финальную выходную модель знаний;

получают финальную выходную модель знаний.