На правах рукописи

Каршиев Зайнидин Абдувалиевич

СРЕДСТВА СОЗДАНИЯ ПАРАЛЛЕЛЬНЫХ АЛГОРИТМОВ ИНТЕЛЛЕКТУАЛЬНОГО АНАЛИЗА ДАННЫХ

05.13.11 – Математическое и программное обеспечение вычислительных машин, комплексов и компьютерных сетей

АВТОРЕФЕРАТ
диссертации на соискание ученой степени
кандидата технических наук

Санкт-Петербург – 2013

Работа выполнена в федеральном государственном бюджетном образовательном учреждении высшего профессионального образования «Санкт-Петербургский государственный электротехнический университет «ЛЭТИ» им. В.И. Ульянова (Ленина)» (СПбГЭТУ) на кафедре вычислительной техники

Научный руководитель – доктор технических наук, профессор
Куприянов Михаил Степанович

Официальные оппоненты:
доктор технических наук, профессор Яковлев Сергей Алексеевич, Санкт-Петербургский государственный электротехнический университет «ЛЭТИ», профессор кафедры «Автоматизированных систем обработки информации и управления»

кандидат технических наук, Смирнов Александр Николаевич, ЗАО «Моторола Солюшнз», ведущий инженер

Ведущая организация – Федеральное государственное автономное образовательное учреждение высшего профессионального образования «Санкт-Петербургский государственный университет аэрокосмического приборостроения»

Защита состоится «____»_____2013 г. в _____час. на заседании диссертационного совета Д 212.238.01 Санкт-Петербургского государственного электротехнического университета «ЛЭТИ» им. В.И. Ульянова (Ленина) по адресу: 197376, Санкт-Петербург, ул. проф. Попова, 5.

С диссертацией можно ознакомиться в библиотеке университета.

Автореферат разослан «_____»_____2013 г.

Ученый секретарь
Диссертационного совета
к.т.н. Щеголева Н.Л.

Общая характеристика работы

Актуальность работы

Наибольшая ценность и получаемых знаний при использовании алгоритмов интеллектуального анализа (ИАД) возможна при анализе значительных объемов данных. При этом возникают следующие основные проблемы анализа:

• производительность – анализ больших объемов (измеряемых терабайтами) требует больших вычислительных ресурсов и может выполняться за неприемлемое для аналитика время;

• распределенность – в связи с большим объемом данных хранение информации может быть реализовано в распределенном хранилище, кроме того в силу природы данных они могут храниться в разных источниках.

Обе проблемы могут решаться за счет параллельного и/или распределенного выполнения интеллектуального анализа данных.

В последние несколько лет повышение производительности вычислительной техники связано, как с развитие многоядерных процессоров, так и с все большим распространение кластерных систем, в том числе и «облачных» систем. Однако современное программное обеспечение значительно отстает от аппаратной части и часто неэффективно использует предоставляемые вычислительные ресурсы. Данная проблема в первую очередь связана с большой трудоемкостью решения задачи распараллеливания вычислительных алгоритмов.

Не являются исключением и алгоритмы интеллектуального анализа данных. В настоящее время проводится достаточно большое количество исследований в этой области. Выделены отдельные направления в области ИАД (в зарубежной литературе данная область имеет название Data Mining): параллельный ИАД (Parallel Data Mining) и распределенный ИАД (Distributed Data Mining). Большинство усилий исследователей в области параллельных алгоритмов ИАД направлено на распараллеливание отдельных алгоритмов анализа и их дальнейшую оптимизацию. Как правило, получаемы решения ориентированы на определенную среду вычисления и при переносе такого решения в другие условия оно становится не эффективным. В связи с этим, исследование в области общих подходов к распараллеливанию существующих алгоритмов интеллектуального анализа является актуальной задачей.

Целью диссертационной работы является разработка средств построения параллельных алгоритмов ИАД для выполнения в распределенной среде. Для достижения заявленной цели в работе решаются следующие **задачи:**

– анализ существующих подходов к созданию параллельных алгоритмов ИАД;

– разработка формальной модели алгоритма ИАД;

– разработка метода создания параллельных алгоритмов ИАД на основе потокобезопасных функциональных блоков;

– разработка методики построения параллельных алгоритмов ИАД для выполнения в распределенной среде;

– разработка программных шаблонов для реализации последовательных и параллельных алгоритмов ИАД из потокобезопасных функциональных блоков;

– проведение экспериментов по выполнению алгоритмов, построенных в соответствии с предложенной методикой.

Объектом исследования являются алгоритмы ИАД.

Предметом исследования являются методы распараллеливания алгоритмов ИАД.

Методы исследования. Методы теории множеств, методы распараллеливания алгоритмов, методы проектирования программного обеспечения.

Научная новизна работы заключается в следующем:

1. Предложена формальная модель алгоритма ИАД, отличающаяся представлением алгоритма в виде набора независимых операций, выполняющих изменение состояния модели знаний и структурных блоков, позволяющих модифицировать структуру алгоритма, в том числе и для параллельного выполнения.

2. Предложен метод создания параллельных алгоритмов ИАД, в отличие от существующих, использующий декомпозицию алгоритма на потокобезопасные функциональные блоки и позволяющий выполнять распараллеливание, как за счет изменения структуры параллельного алгоритма, так и за счет настройки его выполнения.

3. Предложена методика распараллеливания алгоритмов ИАД, которая отличается от известных тем, что к последовательным алгоритмам анализа применяется предложенный метод создания параллельных алгоритмов ИАД с учетом характеристик распределенной среды.

Практическая значимость:

1. Для создания параллельных алгоритмов ИАД предложены программные шаблоны, построенные на основе формальной модели и отделяющие реализацию алгоритма от средств распределенного выполнения.

2. Разработана библиотека параллельных алгоритмов ИАД для выполнения в распределенной среде, включающая в себя предложенные шаблоны.

Положения, выносимые на защиту:

1. Формальная модель алгоритмов ИАД.

2. Метод создания параллельных алгоритмов ИАД из потокобезопасных функциональных блоков.

4. Методика распараллеливания алгоритмов ИАД для выполнения в распределенной среде.

Апробация работы. Основные положения и результаты диссертационной работы докладывались и обсуждались на международных конференциях по мягким вычислениям и измерениям SCM’2010, SCM’2011 и SCM’2012, Санкт-Петербург, 2010-2012 гг, конференциях профессорско-преподавательского состава СПбГЭТУ «ЛЭТИ», Санкт-Петербург, 2011-2013 гг.

Внедрение результатов работы. Результаты работы были использованы при выполнении НИР и в учебном процессе СПбГЭТУ на кафедре вычислительной техники.

Достоверность результатов исследования. Достоверность результатов диссертационной работы подтверждается корректным применением математического аппарата и результатами машинного моделирования на гетерогенном кластере в ресурсном центре СПбГЭТУ.

Публикации. Основные теоретические и практические результаты диссертации опубликованы в 12 работах, среди которых 4 работы в ведущих рецензируемых изданиях, рекомендуемых в действующем перечне ВАК, 2 раздела в 2-х монографиях, 4 работы – в материалах международных научно-технических конференций, 2 свидетельства о регистрации программ для ЭВМ.

Структура и объем диссертации. Диссертация состоит из введения, четырех глав с выводами по каждой из них, заключения, списка литературы, включающего 77 наименований. Основная часть работы изложена на 178 страницах машинописного текста. Работа содержит 55 рисунка, 8 таблиц и 1 приложение общим объемом 5 страниц.

Основное содержание работы

Во введении дано обоснование актуальности темы диссертационного исследования, сформулированы цели и задачи работы, ее научная новизна и практическая значимость, представлены положения, выносимые на защиту.

В первой главе дается общий обзор существующих подходов к параллельному выполнению алгоритмов ИАД. В результате анализа существующих направлений исследований в области разработки параллельных алгоритмов ИАД можно выделить два основных подхода:

– декомпозиция алгоритма для возможной параллелизации – обобщенный подход к алгоритму ИАД, предполагающий разделения его на некоторые части, которые могут выполняться параллельно;

– индивидуальное распараллеливание алгоритмов – индивидуальный подход к каждому алгоритму ИАД и выбор наиболее эффективной параллельной структуры для заданных условий.

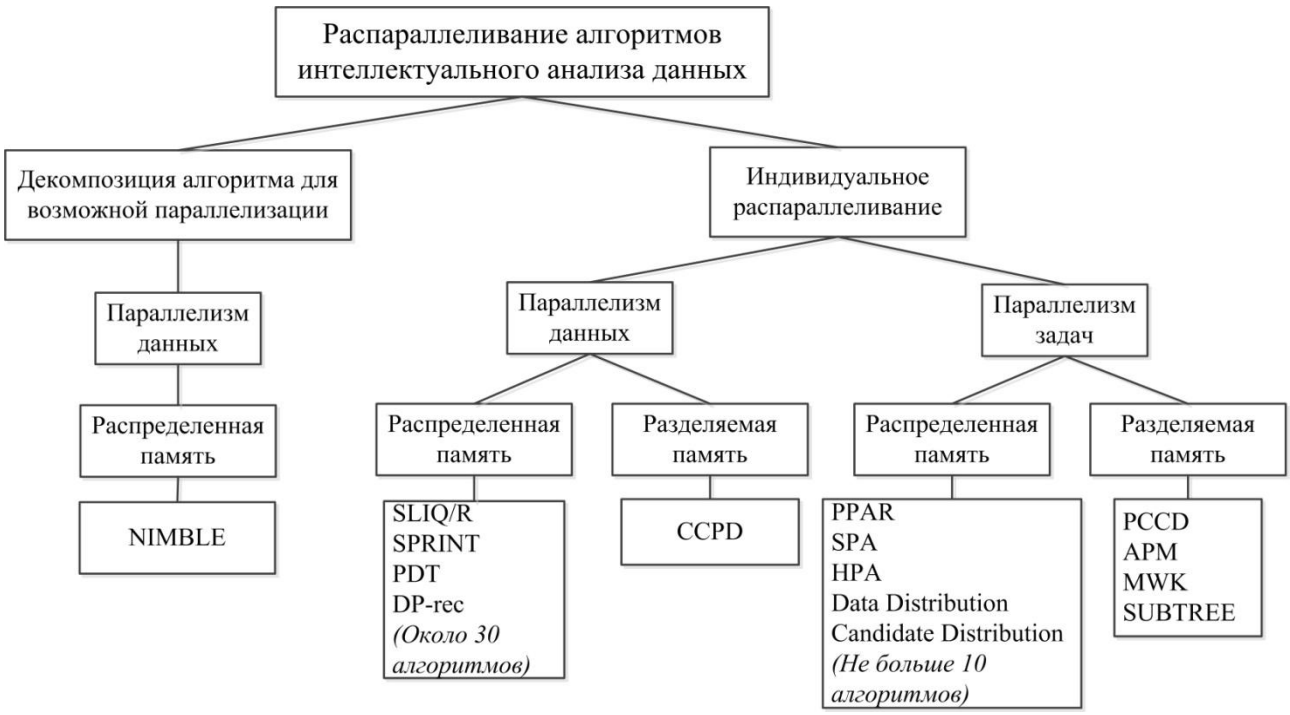

Рисунок 1. Систематизация параллельных алгоритмов интеллектуального анализа данных

В рамках первого подхода можно выделить коммерческий проект NIMBLE компании IBM. Данный проект направлен на разработку инфраструктуры, позволяющей осуществлять параллельное выполнение алгоритмов ИАД на средствах, реализующих концепцию MapReduce. Она предполагает, что алгоритм декомпозирован на отдельные части (части map и reduce) которые могут выполняться параллельно. Данный проект, во-первых, ограничен концепцией MapReduce, а во-вторых не содержит методики декомпозиции алгоритма ИАД.

В рамках второго подхода сосредоточены наибольшие усилия, которые оформились в два направления: параллельный и распределенный интеллектуальный анализ данных. Однако эти усилия направлены на оптимизацию параллельных структур отдельных алгоритмов для определенной среды выполнения. По этой причине предложенные параллельные алгоритмы не всегда являются эффективными при изменении среды выполнения.

Большинство предложенных параллельных алгоритмов ИАД используют параллелизм по данным (рисунок 1) и предназначены для выполнения в системах с распределенной памятью. Алгоритмы, использующие параллелизм задач, также в основном предназначены для работы с распределенной памятью.

Оптимизация алгоритма ИАД под конкретные условия, является трудоемкой задачей. Адаптация таких алгоритмов к новым условиям выполнения, требует затрат сравнимых с созданием новых алгоритмов. По этой причине данный подход не может рассматриваться как общий подход построения параллельных алгоритмов ИАД.

В итоге можно сделать вывод, что не существует известного подхода к созданию параллельных алгоритмов ИАД, общего для большинства алгоритмов и средств выполнения, определяющего также и методику распараллеливания.

Во второй главе описана блочная структура алгоритма ИАД и его формальное описание.

Проведенный анализ алгоритмов ИАД позволил выделить ряд особенностей таких алгоритмов:

1) итерационная обработка больших объемов данных;

2) наличие в алгоритмах типовых блоков (циклы по векторам и атрибутам и др.);

3) формирование на основе входных данных (набора данных, настроек выполнения и начальной модели знаний) результата в виде модели знаний.

Исходя из этого, алгоритм ИАД можно представить как некоторую функцию, возвращающую построенную модель знаний. Входными аргументами такой функции являются: набор данных, настройки и начальная модель знаний (пустая структура, не содержащая никаких закономерностей):

$$M = F\left(D,S,M_0\right).$$

Набор данных (D) опишем формально, как набор из трех элементов:

$$D\left(A,T,W\right),$$

где:

A – независимые атрибуты набора данных:

$$A = \left\{a_1,...,a_i,...,a_m\right\},$$

где a_i – атрибут набора данных;

T – множество целевых атрибутов определяемых, как и атрибуты выше:

$$T = \left\{a_{m+1},...,a_k,...,a_t\right\}.$$

Пара $\left(A,T\right)$ представляет собой метаданные набора данных.

$W = \left\{w_1,...,w_r,...,w_z\right\}$ – множество векторов данных представляющих собой наборы значений соответствующих атрибутов:

$$w_r = \left\{v_{1.r},...,v_{i.r},...,v_{m.r},v_{m+1.r},...,v_{k.r},...,v_{m+t.r}\right\},$$

где $v_{i.r} \in D\left(a_i\right)$ – значение атрибута a_i вектора $v_{i.r}$.

Модель знаний (M) формально можно описать следующим образом:

$$M = \{R, K_t\},$$

где

$R = \{r_1,...,r_i,...,r_n\}$ – множество правил;

$K_t = \{k_1,...,k_p,...,k_g\}$,

где $k_p = \left|\{w_r : w_r \in W, v_{t.r} = v_{t.p}\}\right|$ – количество векторов, у которых целевой атрибут a_t имеет значение $v_{t.p}$, при этом $v_{t.1} \cup ... \cup v_{t.p} \cup ... \cup v_{t.z} = D(t)$.

Модель знаний (M) может дискретно принимать одно из состояний:

- M_0 – начальное состояние модели знаний;
- M_1 –модель после 1-го изменения;
- ….....
- M – завершенная модель знаний.

Под дискретным состоянием понимается состояние модели знаний, при которой она является целостной, т.е. удовлетворяет всем ограничениям, которые на нее накладываются.

Настройки алгоритма можно представить как набор пар: параметр и его значение:

$$S = \left((s_1, p_1),...,(s_{y,} p_y),...,(s_z, p_z)\right),$$

где s_y – y-й параметр настройки алгоритма, а p_y – значение y-ого параметра настройки выполнения алгоритма.

Алгоритм ИАД (F) можно представить как набор последовательно выполняющихся операций:

$$F = \{O_1,...,O_p,...,O_q\},$$

где O_p – операция, изменяющая модель знаний от одного дискретного состояния в другое на основании преданных ей данных и настроек. Таким образом, аргументами операции должны быть данные D, настройки S и модель знаний до изменений M_i, а ее результатом модель знаний после изменения M_{i+1}:

$$M_{i+1} = O(D, S, M_i).$$

Очевидно, что модель знаний, передаваемая в качестве аргумента, и модель знаний, возвращаемая в результате выполнения операции, должны быть целостными.

В соответствии с этим, алгоритм ИАД можно представить как последовательность (упорядоченное множество) операций, результат каждой из которых передается следующей в последовательности операции в качестве аргумента:

$$F = \left\{O_1(D, S, M_0),...,O_p(D, S, M_p),...,O_q(D, S, M_q)\right\}.$$

При этом каждую такую операцию можно также декомпозировать на отдельные операции (функциональные блоки), выполняющиеся последовательно:

$$M_{p+1} = O_p(D, S, M_p) = \left\{O_{p.1}(D, S, M_p), O_{p.2}(D, S, M_{p.1}),...,O_{p.u}(D, S, M_{p.u}),...,O_{p.y}(D, S, M_{p.y})\right\}$$

Кроме операций, изменяющих модель знаний, в формальную модель добавлены следующие элементы, определяющие структуру алгоритма: условные операторы, циклы (в том числе циклы по векторам и по данным).

Условный оператор представим как элемент следующего вида:

$$M_{i+1} = D(D,S,M_i) = \{d(D,S,M_i), O_t(D,S,M_i), O_f(D,S,M_i)\},$$

где:

$d(D,S,M_i)$ – условная функция, возвращающая значение истина или ложь, на основании анализа данных D и текущей модели M_i, но не меняющая их;

$O_t(D,S,M_i)$ – операция, выполняющаяся при значении - истина, возвращаемом функцией $d(D,S,M_i)$;

$O_f(D,S,M_i)$ – операция, выполняющаяся при значении - ложь, возвращаемом функцией $d(D,S,M_i)$;

Цикл представим следующим образом:

$$M_{i+1} = C(D,S,M_i) = \{d(D,S,M_i), O_c(D,S,M_i)\},$$

где $O_c(D,S,M_i)$ – операция, выполняющаяся пока условная функция $d(D,S,M_i)$ возвращает значение – истина.

Таким образом, в алгоритме могут присутствовать как операции, так и структурные элементы. Обобщим их в понятие функционального блока алгоритма:

$$b_i(D,S,M_i) = \begin{cases} O(D,S,M_i) \in \{O^{add}(D,S,M_i), O^{del}(D,S,M_i), O^{ch}(D,S,M_i)\} \\ E(D,S,M_i) \in \{D(D,S,M_i), C(D,S,M_i)\} \end{cases} \quad (1)$$

Описанные таким образом функциональные блоки обладают двумя свойствами:

- детерминизм – при одних и тех же входных аргументах (данных, настройках и начальном состоянии модели знаний) возвращаемая модель знаний будет одна и та же;
- не имеет побочных эффектов – т.е. при выполнении функциональных блоков изменение внешних элементов не происходит.

Алгоритм ИАД может быть представлен как упорядоченная последовательность функциональных блоков:

$$M_q = F(M_0,S,D) = (b_0(D,S,M_0),...,b_k(D,S,M_k),...,b_q(D,S,M_{q-1})).$$

Таким образом, сам алгоритм ИАД также не имеет побочных эффектов и обладает свойством детерминизма.

Для описания с помощью формальной модели параллельно выполняющихся ветвей алгоритма добавим в нее следующий структурный элемент:

$$M_{i+1} = P(D,S,M_i) = \{s(D,S,M_i), b_d(D,S,M_i), b_1(D,S,M_i),...,b_x(D,S,M_i),...,b_z(D,S,M_i), j(D,S,M_i)\}$$

где $s(D,S,M_i)$ – операция split, выполняющая подготовительные действия для параллельного выполнения операций; $j(D,S,M_i)$ – операция join, выполняющая объединяющие действия после параллельного выполнения операций; $b_d(D,S,M_i)$ – блок, являющийся диспетчером для остальных блоков, если $b_d(D,S,M_i) = \varnothing$, то имеет место параллелизация без диспетчера.

Если параллельно выполняемые блоки одинаковы $(b_1 = ... = b_x = ... = b_z)$, то имеет место параллелизм по данным, иначе (если $b_1 \neq ... \neq b_x \neq ... \neq b_z$) имеет место параллелизм по задачам.

В случае параллелизма по данным должно выполняться условие:

$$D_1 \neq D_2 \neq ... \neq D_x \neq ... \neq D_z \text{ и } D = D_1 \cup D_2 \cup ... \cup D_x \cup ... \cup D_z,$$

т.е. все данные, обрабатываемые параллельными блоками, должны быть разными и их объединение должно составлять полный набор данных.

В случае параллелизма по задачам должно выполняться условие:

$$D_1 = D_2 = ... = D_x = ... = D_z,$$

т.е. все наборы данных, обрабатываемые параллельными блоками, представляют собой полные наборы данных.

Для взаимодействия параллельно выполняющихся последовательностей добавим в модель еще два структурных элемента:

– блок отправки данных:

$$S(D,S,M_i) = \{O(D,S,M_i), s(m)\},$$

где $O(D,S,M_i)$ – операция выполняемая, перед отправкой сообщения (например, подготовку сообщения); $s(m)$ – операция, выполняющая отправку сообщения *m*;

– блок получения данных:

$$R(D,S,M_i) = \{r(m), O(D,S,M_i)\},$$

где $r(m)$ – выполняющая прием сообщения m; $O(D,S,M_i)$ – операция выполняющая обработку полученного сообщения.

Таким образом, окончательный вид формальной модели алгоритма ИАД (1) для параллельного выполнения будет следующий:

$$b_i(D,S,M_i) = \begin{cases} O(D,S,M_i) \in \{O^{add}(D,S,M_i), O^{del}(D,S,M_i), O^{ch}(D,S,M_i)\} \\ E(D,S,M_i) \in \{D(D,S,M_i), C(D,S,M_i), P(D,S,M_i), S(D,S,M_i), R(D,S,M_i)\} \end{cases} \quad (2)$$

В соответствии с формальной моделью, алгоритм ИАД и блоки его составляющие не имеют побочных эффектов и обладают свойством детерминизма. Эти свойства гарантируют потокобезопастность таких блоков, следовательно, они могут быть выполнены параллельно.

В соответствии с формальной моделью, при разбиении на блоки необходимо соблюдать следующие требования:

- алгоритм является упорядоченной последовательностью функциональных блоков (1), которые удовлетворяют условиям описанным ниже;
- каждый блок может представлять собой или некоторую операцию, выполняемую над моделью или некоторый структурный элемент, характерный для алгоритма ИАД (2);
- блок, реализующий некоторую операцию, должен изменять модель знаний, поданную на вход таким образом, чтобы она оставалась целостной (т.е. должны выполняться все ограничения накладываемые на модель);
- внутри функционального блока не должно происходить обращений к внешним переменным, вся работа должна выполняться на основании: набора данных D, настроек S и исходной модели знаний M_{i-1}, переданных в функциональный блок.

В соответствии с формальной моделью функциональный блок может содержать как исполняемый код (неразделимую операцию *O*), так и последовательность других функциональных блоков. В общем случае блок может содержать не одну последовательность.

В результате алгоритм ИАД можно представить как иерархию вложенных потокобезопасных функциональных блоков (рисунок 3). При этом сам алгоритм, как отдельный функциональный блок также является потокобезопасным.

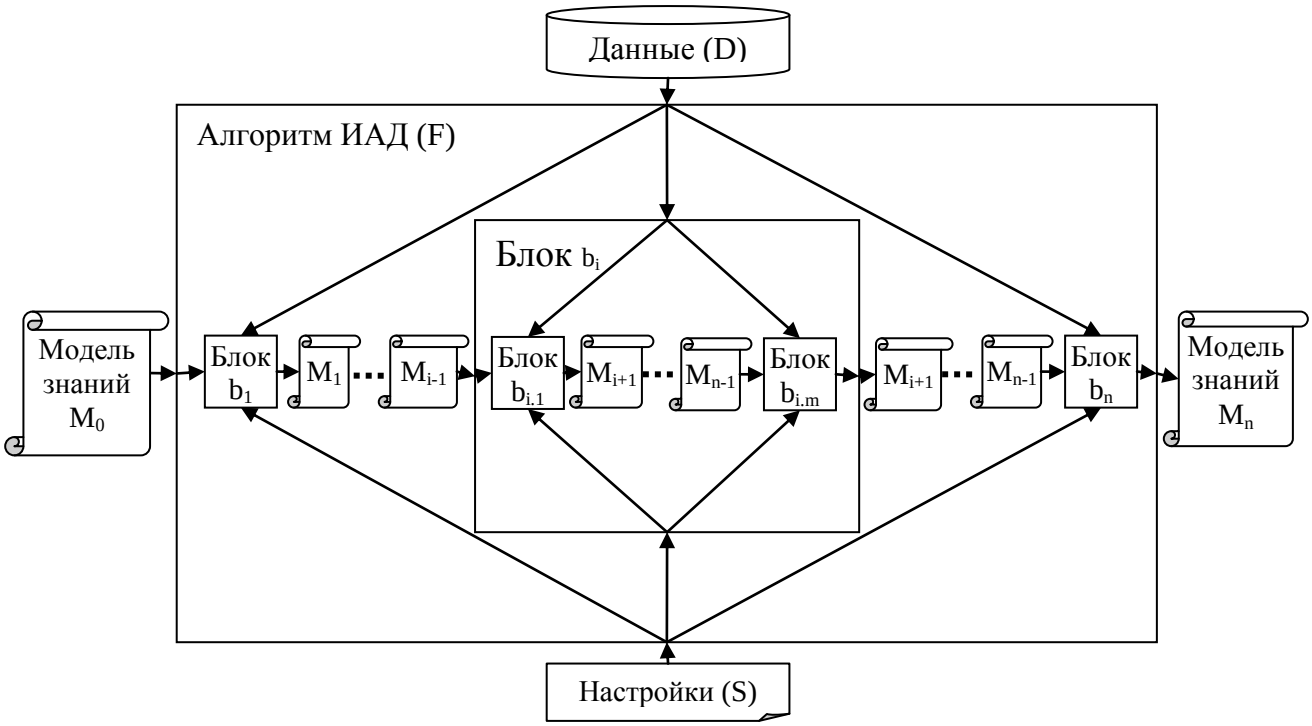

Рисунок 3. Блочная структура алгоритма

На основе выделенных в формальной модели блоков определены базовые потокобезопасные функциональные блоки для построения алгоритмов ИАД:

– блок решения $D(D,S,M_i)$;

– блок цикл $C(D,S,M_i)$ и характерные для алгоритмов ИАД:

- блок цикл по векторам $C_{bv}(D,S,M_i)$;
- блок цикл по атрибутам $C_{ba}(D,S,M_i)$;

– блок параллельного выполнения $P(D,S,M_i)$ и его разновидности:

- блок параллелизации по данным $P_{bd}(D,S,M_i)$;
- блок параллелизации по задачам $P_{bt}(D,S,M_i)$;

– блок отправитель $S(D,S,M_i)$;

– блок получатель $R(D,S,M_i)$.

На основе представления алгоритма в виде иерархии вложенных функциональных блоков, предложен метод построения параллельных алгоритмов ИАД. Для построения параллельного алгоритма ИАД метод предполагает выполнение следующих действий:

- декомпозировать алгоритм ИАД на элементарные действия;
- сгруппировать элементарные действия в функциональные блоки, выполняющие дискретное изменение модели знаний;
- выделить структурные элементы и вложенные в них последовательности функциональных блоков;
- добавить в структуру алгоритма ИАД функциональные блоки для параллельного выполнения.

Используя перечисленные выше функциональные блоки, предложенный метод позволяет построить следующие типы параллельных структур алгоритмов ИАД:

– с параллелизацией по задачам/данным;

– с/без взаимодействия между параллельными ветвями для систем с распределенной памятью;

– с/без диспетчера.

Настройка блоков параллельного выполнения позволяет реализовать следующие типы параллельных алгоритмов ИАД:

- с одним источником данных и одной моделью знаний (SDSM) – в этом случае параллельные последовательности функциональных блоков работают с одним источником данных и одной моделью знаний;
- с несколькими источниками и одной моделью знаний (MDSM) – в этом случае каждая последовательность работает с отдельным источником, но с одной моделью знаний;
- с одним источником и несколькими моделями знаний (SDMM) – в этом случае каждая последовательность работает с общим источником данных, но строит каждая свою модель знаний, которые впоследствии объединяются в одну;
- с несколькими источниками и несколькими моделями знаний (MDMM) – в этом случае каждая последовательность работает со своим источником данных и строит каждая свою модель знаний, которые впоследствии объединяются в одну.

В третьей главе описаны программные шаблоны в виде классов для построения алгоритмов ИАД из отдельных блоков. Классы определяют, как операции, так и блоки соответствующие структурным элементам: блоки решения и циклы. Кроме того, предложены классы для реализации параллельных алгоритмов ИАД.

Рассмотрим каждый из классов подробнее.

Класс Step – класс, реализующий блок – операцию и являющийся базовым классом для всех блоков алгоритма. Входными параметрами для него являются данные, настройки и модель. Наиболее значимым методом класса является метод execute(), запускающий выполнение блока. Этот метод переопределяется дочерними классами в зависимости от их назначения.

Класс SequenceOfSteps – класс, реализующий упорядоченную последовательность блоков. Класс хранит последовательность объектов класса Step и переопределяет метод execute() таким образом, что он последовательно запускает все блоки на выполнение и обеспечивает передачу модели знаний измененной предыдущим блоком следующему блоку.

Класс CyclicStep – абстрактный класс для реализации цикла. Он хранит последовательность блоков (как объект класса SequenceOfSteps). На каждой итерации цикла выполняется хранящаяся последовательность блоков.

Класс DecisionStep – абстрактный класс проверки условия. Включает в себя две последовательности блоков (объектов класса SequenceOfSteps) и абстрактную функцию condition(), возвращающую значения «истина» или «ложь». В зависимости от результата данной функции выполняется одна или другая последовательность блоков.

Класс CycleByVectors – класс, реализующий цикл по векторам. Данный класс расширяет класс CyclicStep и осуществляет цикл по элементам (векторам) входных данных.

Класс MiningAlgorithm **–** класс, реализующий сам алгоритм как упорядоченную последовательность функциональных блоков. Абстрактный класс MiningAlgorithm содержит основную последовательность блоков алгоритма (как объект класса SequenceOfSteps). Он производит необходимые действия по инициализации модели знаний и объектов класса Step, после чего запускает выполнение основной последовательности блоков.

Класс ParallelStep является абстрактным (он не реализует описанный в классе Step метод execute()) и содержит в себе массив обработчиков параллельного выполнения ветвей алгоритма – handlers (и методы доступа к ним), а также настройки параллельного выполнения paralellAlgSettings.

Класс ParallelBlock отвечает за выполнение параллельных ветвей алгоритма и наследуется от класса ParallelStep. В классе ParallelBlock кроме унаследованных методов определены еще два метода – split() и join(). В методе split() выполняются операции по запуску параллельных ветвей алгоритма. В методе join() объединяются потоки и результаты работы отдельных ветвей.

Для реализации параллелизации по данным и по задачам от класса ParallelBlock наследуются два класса ParallelByData и ParallelByTask.

Класс ParallelByData реализует параллелизацию по данным. Он содержит параметр branche – последовательность шагов параллельных ветвей алгоритма, которая клонируется при запуске алгоритма для каждого доступного обработчика параллельной ветви. Для работы с этим параметром в классе определены: метод getBranche() – для получения последовательности и метод setBranche() – для формирования последовательности шагов.

Класс ParallelByTask реализует параллелизацию по задачам. Он, в отличие от класса ParallelByData, содержит массив последовательностей шагов для каждой ветви параллельного алгоритма - branches. Для работы с этим параметром в классе определены: метод getBranches() – для получения списка последовательностей и метод addBranche() – для добавления новой последовательности шагов.

Класс SenderStep – класс, реализующий блок отправителя. Он в методе execute() рассылает сообщения посредством обработчиков handlers всем параллельно выполняющимся ветвям алгоритма ИАД. Алгоритм формирования сообщения, как и само сообщение, зависит от алгоритма ИАД и реализуется в классе, наследуемом от SenderStep и реализующем соответствующий функциональный блок алгоритма.

Класс ReceiverStep – класс, реализующий блок получателя. Он определяет переменную message, содержащую получаемое сообщение. В методе execute() данного класса выполняется непосредственное получение сообщения и инициализация переменной message. Получение сообщения выполняется в цикле ожидания, т. е. выход из цикла осуществляется только после получения сообщения. Переменная message доступна дочерним классам и может быть в дальнейшем обработана в соответствии с алгоритмом ИАД.

Описанные классы позволяют реализовать любую структуру параллельного алгоритма из приведенных во второй главе.

Реализация операций split и join в классе параллельного блока позволяет выполнять параллелизацию вида SDSM, SDMM, MDSM и MDMM без изменения структуры алгоритма. Изменение поведения алгоритма выполняется за счет настройки его выполнения.

Существующие в классе ParallelStep ссылки на обработчики handlers, являются экземплярами класса ExecutionHandler – общего для всех средств распределенного вы-

полнения. В настоящее время широко используются средства выполнения распределенных вычислений: многопоточное выполнение, сервис-ориентированное выполнение, модель акторов, выполнение на основе концепции MapReduce, мультиагентное выполнение и др.

Для того, чтобы реализация алгоритма и его параллельная структура не зависели от средства выполнения, используется шаблон проектирования «Адаптер». В качестве адаптера используются классы, наследуемые от класса ExecutionHandler (на который ссылается класс ParallelStep) для каждого средства реализации.

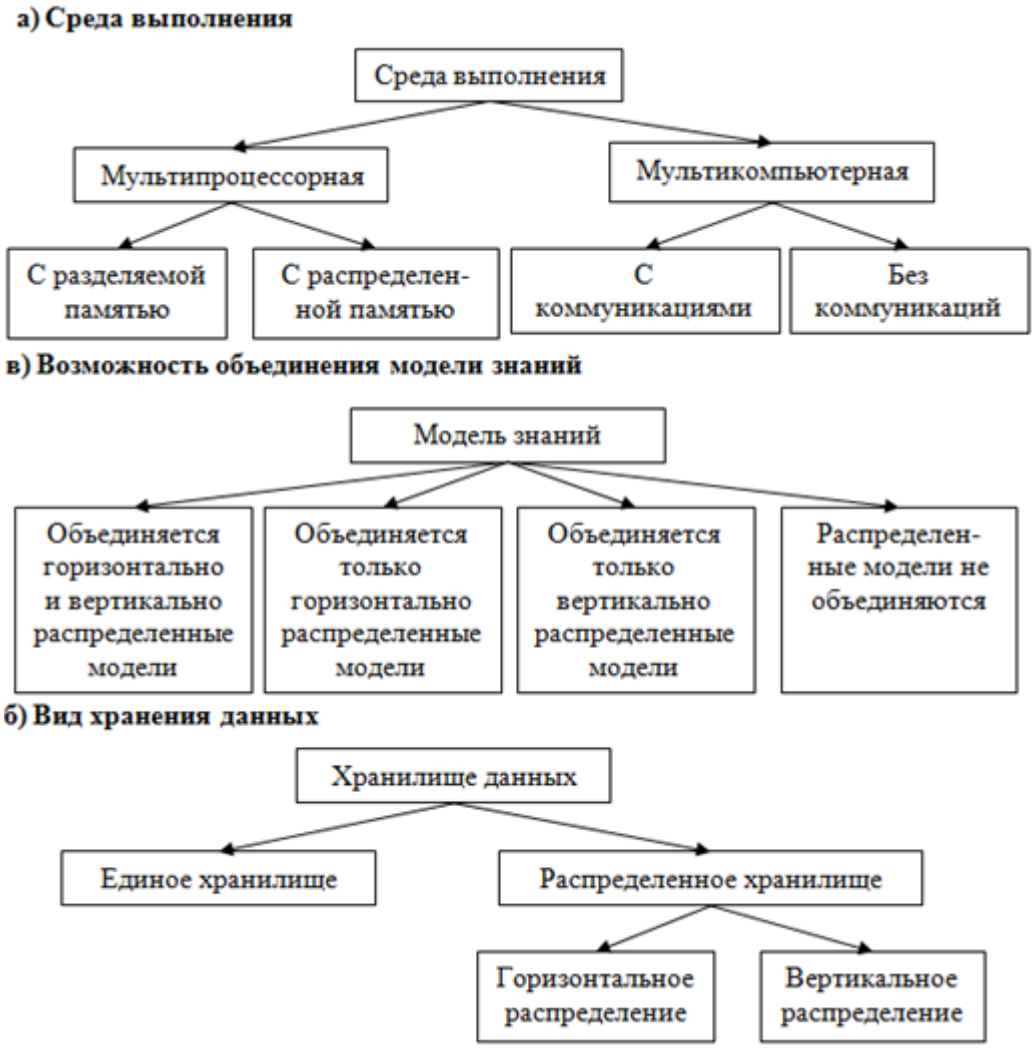

Рисунок 4. Исходные данные для методики распараллеливания алгоритмов ИАД.

Обобщением полученных результатов, является предложенная методика построения параллельных алгоритмов ИАД из потокобезопасных функциональных блоков для выполнения в распределенной среде. При решении задачи построения такого алгоритма исходными данными являются (рисунок 4):

– среда выполнения параллельного алгоритма (рисунок 4. а);
– вид хранения данных (рисунок 4. б);
– вид функции ИАД (рисунок 4. в).

Методика состоит из двух основных этапов:

I. определение структуры параллельного алгоритма ИАД;

II. настройка параллельного алгоритма ИАД.

На первом этапе выполняется анализ структуры алгоритма ИАД с целью выявления типовых элементов и других особенностей, присущих алгоритму. На основании результатов анализа формируется структура параллельного алгоритма.

Для определения структуры параллельного алгоритма необходимо выполнить следующие действия:

I.1. декомпозировать алгоритм ИАД на элементарные операции;

I.2. сгруппировать в функциональные блоки действия, выполняющие дискретное изменение модели;

I.3. выполнить программную реализацию блочной структуры последовательного алгоритма;

I.4. выполнить анализ полученной структуры с учетом оставшихся циклов и условных переходов и определить структуру параллельного алгоритма;

I.5. выполнить программную реализацию параллельной структуры алгоритма ИАД на основе ранее реализованных функциональных блоков.

Первые два шага выполняются в соответствии с предложенным методом построения параллельного алгоритма ИАД, описанного выше.

На втором этапе распараллеливания алгоритма производится настройка выполнения параллельного алгоритма и анализ его эффективности с последующей отладкой. Этап включает в себя следующие шаги:

II.1. настройка выполнения параллельного алгоритма ИАД в соответствии с параметрами среды и типом распределения данных;

II.2. анализ параллельного выполнения алгоритма ИАД на тестовых данных, путем оценки эффективности и ускорения;

a. возврат на шаг II.1. в случае если результаты неудовлетворительны и не все настройки были испытаны;
b. возврат на шаг I.4. в случае если результаты неудовлетворительны и все настройки были испытаны.

Описанную методику можно представить в виде блок схемы, изображенной на рисунке 5. На ней пунктирной линией обозначена граница этапов методики, затрагивающих структуру алгоритма и не затрагивающих ее (изменяющие характеристики параллельного выполнения за счет настройки).

Достоинством методики является то, что наиболее трудоемкие шаги, связанные с реализацией функциональных блоков, не повторяются при отладке алгоритма и повышении его эффективности. Все изменения алгоритма в данном направлении связаны или с реструктуризацией алгоритма без изменения функциональных блоков или с его перенастройкой.

Дальнейшее развитие полученных результатов может быть направлено на программную реализацию метода оценки эффективности и автоматизацию этапа связанного с выбором наиболее эффективной структуры параллельного алгоритма ИАД.

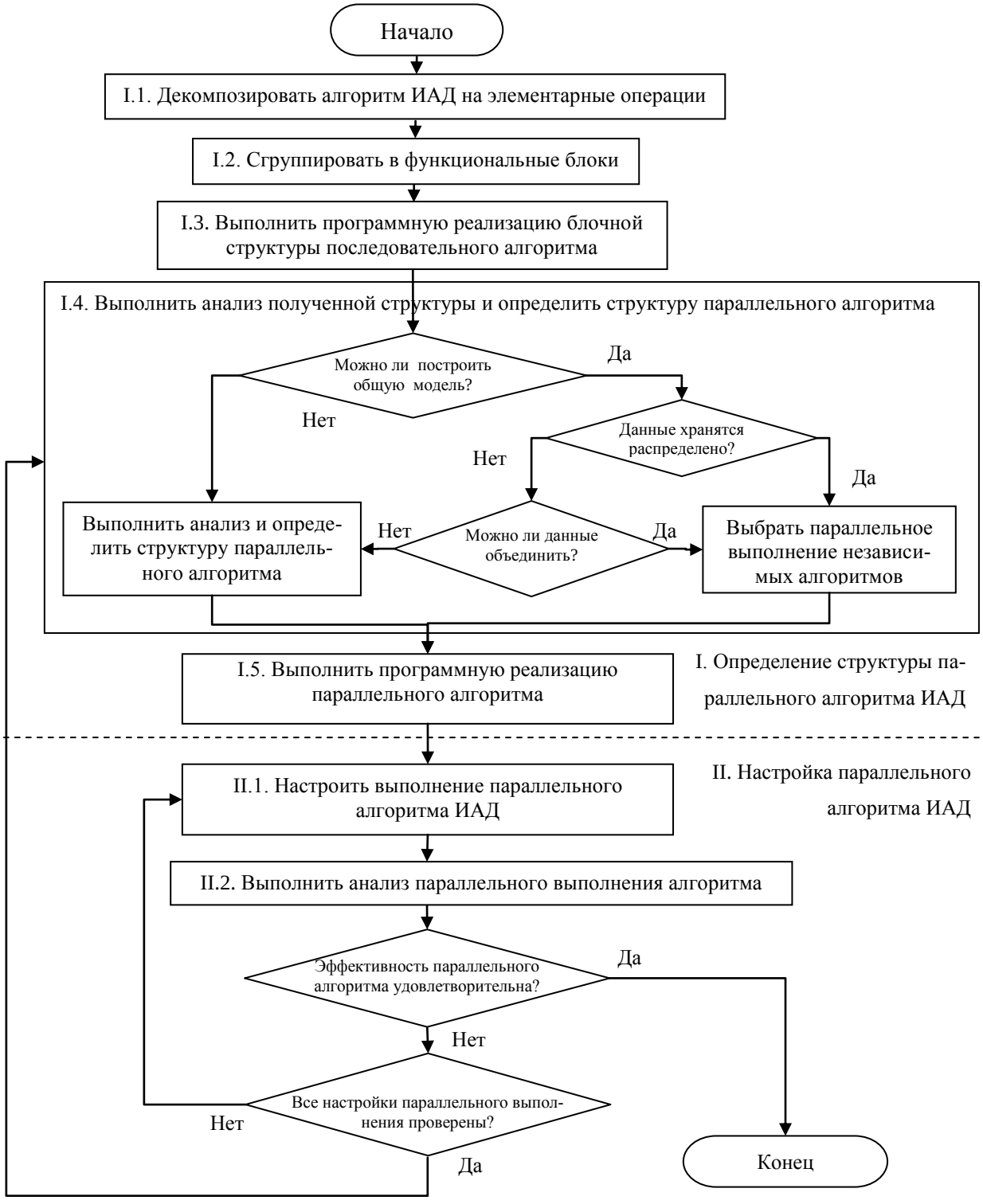

Рисунок 5. Методика распараллеливание алгоритма ИАД

В четвертой главе описана библиотека алгоритмов ИАД, реализующая блочный подход построения алгоритмов ИАД и их распараллеливания. Ядром библиотеки являются классы, реализованные в соответствии с предложенной моделью алгоритма ИАД и позволяющие реализовывать потокобезопасные функциональные блоки алгоритмов ИАД и выполнять их параллельно.

Для выполнения полного цикла анализа в библиотеку кроме классов для реализации алгоритмов добавлены классы, описывающие: исходные данные и их метаданные; модель знаний; настройки уровня алгоритма и функции ИАД.

Библиотека алгоритмов ИАД разделена на компоненты:

– CMW Classes – классы стандарта CWM;

– Library Core – базовые классы библиотеки;

– Parallel Core – классы для параллельного выполнения;

– Clustering Algorithms – алгоритмы кластеризации;

– Classification Algorithms – алгоритмы классификации и регрессии;

– Association Algorithms – алгоритмы поиска ассоциативных правил.

Для проверки выносимых на защиту результатов с использованием библиотеки было выполнено распараллеливание алгоритмов кластеризации и классификации – k-means и 1R в соответствии с предложенной методикой.

При распараллеливании алгоритма k-means первая структура параллельного алгоритма (предполагающая частичное распараллеливание) была признана не эффективной для всех типов параллельного выполнения алгоритма ИАД (Таблица 1).

Таблица 1

Количество векторов	150	1500	3000	6000
Последовательный алгоритм	27,834 мс	36,918 мс	553,274 мс	1035,382 мс
Параллельный алгоритм				
SDSM	40,893 мс	118,984 мс	1219,746 мс	2760,848 мс
SDMM	68,804 мс	112,436 мс	1451,084 мс	3783,674 мс
MDSM	35,516 мс	45,218 мс	619,356 мс	1466,711 мс
MDMM	49,449 мс	77,365 мс	811,494 мс	2138,980 мс

Для повышения эффективности в соответствии с методикой было принято решение о реструктуризации алгоритма и параллельном выполнении всего алгоритма. Реструктуризация была выполнена за счет незначительного изменения кода. Этот алгоритм реализован в 528 строках кода и для изменения структуры алгоритма были изменены только четыре из них, то есть 0,75 % кода. Повторные эксперименты показали эффективность новой структуры при всех типах параллельного выполнения кроме SDMM для данных с количеством векторов 150 и 1500 (Таблица 2).

Таблица 2

Количество векторов	150	1500	3000	6000
Последовательный алгоритм	27,834 мс	36,918 мс	553,274 мс	1035,382 мс
Параллельный алгоритм				
SDSM	19,445 мс	27,182 мс	341,904 мс	676,175 мс
SDMM	27,327 мс	41,651 мс	372,165 мс	722,527 мс
MDSM	15,669 мс	20,612 мс	303,576 мс	587,922 мс
MDMM	15,585 мс	21,106 мс	289,781 мс	613,402 мс

Модели, полученные в результате выполнения последовательного и параллельных алгоритмов k-means, не отличаются, то есть при распараллеливании алгоритма результат его работы не изменился.

При распараллеливании алгоритма 1R также были экспериментально проверены две параллельные структуры: с частичным распараллеливанием и с параллельным выполнением всего алгоритма. Первая структура алгоритма дает эффективные результа-

ты при всех типах параллельного выполнения кроме SDMM и SDSM для данных с количеством векторов 150 (Таблица 3). Вторая структура алгоритма дает эффективные результаты при всех типах параллельного выполнения кроме SDMM (Таблица 4). Реструктуризация параллельного алгоритма 1R также не потребовала больших усилий и была выполнена за счет незначительного изменения кода. Этот алгоритм реализован в 613 строках кода и для изменения структуры алгоритма были изменены только четыре из них, то есть 0,65 % кода.

Таблица 3

Количество векторов	150	1500	3000	6000
Последовательный алгоритм	18,114 мс	33,668 мс	42,704 мс	60,97 мс
Параллельный алгоритм				
SDSM	18,665 мс	31,919 мс	41,446 мс	58,316
SDMM	19,005 мс	35,365 мс	44,428 мс	63,674
MDSM	13,238 мс	27,103 мс	33,795 мс	47,813
MDMM	13,719 мс	26,762 мс	33,902 мс	46,265

Таблица 4

Количество векторов	150	1500	3000	6000
Последовательный алгоритм	18,114 мс	33,668 мс	42,704 мс	60,97 мс
Параллельный алгоритм				
SDSM	11,789 мс	20,803 мс	41,512 мс	58,789 мс
SDMM	18,258 мс	34,917 мс	44,014 мс	60,632 мс
MDSM	13,706 мс	27,983 мс	35,569 мс	47,812 мс
MDMM	13,506 мс	26,883 мс	32,352 мс	43,917 мс

Модели, полученные в результате выполнения последовательного и параллельных алгоритмов 1R, тоже не отличаются, то есть при распараллеливании алгоритма результат его работы не изменился.

Проведенные эксперименты доказали эффективность предложенных решений и подтвердили их правильность.

В заключении сформулированы основные научные и практические результаты работы:

1. Предложена формальная модель алгоритма ИАД, отличающаяся представлением алгоритма в виде набора независимых операций, выполняющих изменение состояния модели знаний и структурных блоков, позволяющих модифицировать структуру алгоритма, в том числе и для параллельного выполнения. Выделены основные типовые элементы алгоритмов, в том числе и для параллельного выполнения.

2. Предложен метод построения алгоритмов ИАД на основе потокобезопасных функциональных блоков, позволяющий выполнять распараллеливание, как на структурном уровне, так и на уровне настройки без необходимости изменения таких блоков.

3. Предложены программные шаблоны для параллельных алгоритмов ИАД на основании формальной модели, позволяющие строить параллельные алгоритмы, как путем структурных изменений, так и путем настройки выполнения и позволяющие отделить реализацию алгоритма от средств распределенного выполнения.

4. Предложена методика декомпозиции и построения параллельных алгоритмов ИАД из потокобезопасных функциональных блоков для выполнения в распределенной среде, которая позволяет не повторять наиболее трудоемкие шаги при оптимизации алго-

ритма. Все изменения алгоритма в данном направлении связаны или с реструктуризацией алгоритма без изменения функциональных блоков или с его перенастройкой.

5. Построена библиотека алгоритмов ИАД для выполнения в распределенной среде.

Публикации в журналах, входящих в перечень ВАК

1. Каршиев З.А. Обзор параллельных алгоритмов построения деревьев решений // Известия СПбГЭТУ «ЛЭТИ», № 9, Санкт-Петербург 2011. С. 43-48.

2. Куприянов М.С., Каршиев З.А. Формальная модель выполнения алгоритмов интеллектуального анализа // Известия СПбГЭТУ «ЛЭТИ», № 9, Санкт-Петербург 2012. С. 60-68.

3. Каршиев З. А., Голубев И. А., Прохоренко К. А. Оценка ускорения и эффективности параллельного выполнения алгоритмов интеллектуального анализа данных // Известия СПбГЭТУ «ЛЭТИ», № 10, Санкт-Петербург 2012. С. 46-52.

4. Холод И.И., Каршиев З.А. Методика распараллеливания алгоритмов интеллектуального анализа данных // Известия СПбГЭТУ «ЛЭТИ», № 3, Санкт-Петербург 2013. С. 38-45.

Другие статьи и материалы конференций

5. Куприянов М.С., Каршиев З.А., Громов И.А., Серебрянский Д.А., Холод И.И., Павлухин И.С. Восстановление моделей бизнес процессов в информационных системах // Сборник докладов XIV Международной конференции по мягким вычислениям и измерениям, Санкт-Петербург, 23-25 июня, 2010 г., Том 1, С. 114-117.

6. Холод И.И., Каршиев З.А. Анализ распределенных данных: обзор методов объединение моделей // Сборник докладов XIV Международной конференции по мягким вычислениям и измерениям SCM`2011, Санкт-Петербург, 23-25 июня, 2011 г., Том 1, С. 163-166.

7. Холод И.И., Каршиев З.А., Школьный Р.Е. Виды распределения данных и возможности их интеллектуального анализа // Сборник докладов XIV Международной конференции по мягким вычислениям и измерениям SCM`2011, Санкт-Петербург, 23-25 июня, 2011 г., Том 1, С. 255-258.

8. Холод И.И., Каршиев З.А. Параллелизация алгоритма Naïve Bayes на основе блочной структуры. // Сборник докладов XV Международной конференции по мягким вычислениям и измерениям SCM`2012, Санкт-Петербург, 25-27 июня, 2012 г., Том 1, С. 182-185.

9. Куприянов М.С., Холод И.И, Каршиев З.А. и др. Интеллектуальный анализ распределенных данных на базе облачных вычислений (монография). СПб.: Изд-во СПбГЭТУ "ЛЭТИ", 2011. – 148 с.

10. Куприянов М.С., Холод И.И, Каршиев З.А. и др. Интеллектуальный анализ данных в распределенных системах (монография). СПб.: Изд-во СПбГЭТУ "ЛЭТИ", 2012. – 110 с.

11. Каршиев З.А. Автоматическое построение графа структуры лексических единиц (слов). № 2012610985, Федеральная служба по интеллектуальной собственности. Зарегистрировано в Реестре программ для ЭВМ 23 января 2012 г.

12. Холод И.И., Каршиев З.А. Блоковая структура выполнения алгоритма классификации Naïve Bayes. № 2012660852, Федеральная служба по интеллектуальной собственности. Зарегистрировано в Реестре программ для ЭВМ 29 ноября 2012 г.