DOI: 10.18721/JCSTCS.11407
УДК 004.75:004.658.3, 004.657, 004.023

ОБЗОР МЕТОДОВ ДИНАМИЧЕСКОГО РАСПРЕДЕЛЕНИЯ ДАННЫХ В РАСПРЕДЕЛЕННЫХ СИСТЕМАХ УПРАВЛЕНИЯ БАЗАМИ ДАННЫХ

С.Г. Попов, В.С. Фридман

Санкт-Петербургский политехнический университет Петра Великого,
Санкт-Петербург, Российская Федерация

Приведен обзор методов и связанных с ними алгоритмов распределения данных между узлами системы управления распределенными базами данных. Предложена классификация алгоритмов перераспределения данных и приведены описания алгоритмов для случая установившегося состояния функционирования системы управления базами данных. Рассмотрены подходы, основанные на обучающихся автоматах с памятью, моделях прогнозирования потоков запросов методами анализа временных рядов, эволюционные подходы, базирующиеся на генетических алгоритмах, и связанные с ними алгоритмы распределения. Для описанных алгоритмов выделены критерии оптимальности функционирования подсистемы динамического перераспределения данных системы управления распределенной базой данных. Приведенные подходы могут использоваться для построения подсистем перераспределения данных в системах управления распределенными базами данных в процессе их функционирования.

Ключевые слова: данные, распределенные базы данных, система управления базами данных, распределение данных, оптимизация, алгоритмы.

Ссылка при цитировании: Попов С.Г., Фридман В.С. Обзор методов динамического распределения данных в распределенных системах управления базами данных // Научно-технические ведомости СПбГПУ. Информатика. Телекоммуникации. Управление. 2018. Т. 11. № 4. С. 82–107. DOI: 10.18721/JCSTCS.11407

REVIEW OF METHODS FOR DYNAMIC DISTRIBUTION OF DATA IN DISTRIBUTED DATABASE MANAGEMENT SYSTEMS

S.G. Popov, V.S. Fridman

Peter the Great St. Petersburg Polytechnic University,
St. Petersburg, Russian Federation

The article provides an overview of the methods and associated data distribution algorithms between nodes of a distributed database management system. The article proposes a classification of data redistribution algorithms and provides algorithms for a database management system functioning in a stable state. The article discusses approaches based on self-configuring finite-state machines with memory, forecasting query flows using time series analysis methods, heuristic and genetic algorithms. Redistribution algorithms are described for each method. The optimality criteria for the functioning of the dynamic data redistribution subsystem of the distributed database are highlighted for the described algorithms. The given approaches can be used to design data redistribution subsystems in control systems of distributed databases in the course of their operation.

Keywords: data, distributed databases, database management system, data distribution, optimization, algorithms.

Citation: Popov S.G., Fridman V.S. Review of methods for dynamic distribution of data in distributed database management systems. St. Petersburg State Polytechnical University Journal. Computer Science. Telecommunications and Control Systems, 2018, Vol. 11, No. 4, Pp. 82–107. DOI: 10.18721/JCSTCS.11407

Введение

В распределенной системе управления базами данных каждая часть базы должна участвовать в выполнении глобальных запросов, требующих доступа к данным на нескольких других узлах системы. Для получения данных с других узлов используется подсистема связи, время передачи данных по которой существенно влияет на общее время выполнения распределенного запроса [2]. Скорость получения данных из распределенных источников актуальна в таких областях, как геоинформационные системы [22], системы распределенных мультимедиа [15], распределенные системы хранения данных, ориентированные на Web [24]. В этом случае пренебречь временем передачи данных между отдельными частями распределенной базы данных нельзя, и задача оптимального размещения данных становится двухкритериальной: оптимальное размещение данных зависит как от объема данных запроса, так и от скорости каналов связи, соединяющих фрагменты базы.

Рассматривая размещение данных между узлами распределенной базы как динамическую систему, следует поддерживать оптимальность распределения данных между узлами с целью минимизации времени выборки данных из всей базы в целом. При этом глобальным критерием оптимальности функционирования такой системы остается минимизация времени выполнения всех запросов всех абонентов. Непрерывное поддержание распределенной СУБД в оптимальном состоянии невозможно, поскольку непрерывное распределение фрагментов, оказывающее решающее влияние на надежность и производительность [4, 5], является NP-полной задачей [11, 12].

В настоящее время исследователями активно предлагаются подходы к решению задачи дискретной оптимизации размещения данных в системах распределенных баз данных, основанные на методах оптимизации. Основными подходами являются реализации избыточных и неизбыточных методов распределения данных между узлами, ориентированными на технологии переноса данных [2] или их репликации [6, 20]. Рассматривая методы оптимизации размещения данных без избыточности, основной задачей становится выделение фрагмента в распределенной системе баз данных фрагмента данных для переноса на другой узел и определение этого узла. При этом основной принцип распределения – достижение максимальной локальности приложений и фрагментов. На практике это означает хранение фрагментов ближе к тому узлу, где они чаще используются.

Алгоритмы динамического распределения данных в распределенных базах данных можно классифицировать относительно их внутренних и внешних особенностей реализации, на основе их области применения, эффективности и критериев оптимальности. Выделенные классификационные признаки позволяют разделить алгоритмы по способу определения стоимости операций [17], анализу доступных ресурсов на узле [8, 9], типу применяемой избыточности данных [3, 7, 13], способу распределения данных [2], по методам описания [21] и применению распределенной транзакционной модели [22].

Метод пороговой оценки с ограничением по времени, объему и расстоянию

Пороговый алгоритм с ограничением по времени, объему и расстоянию [1] является результатом набора исследований [2, 8, 9, 11, 13] и обеспечивает множественное обращение к одному фрагменту данных от нескольких узлов, путем перераспределения данных на основе предварительно собранной статистики обращений к данным на каждом узле. Алгоритм выполняется на каждом узле и обеспечивает передачу фрагмента данных тому узлу, доступ из которого осуществляется с максимальной интенсивностью за прошедший временной интервал.

Алгоритм реализуется в две фазы: подготовки и сбора статистических данных и анализа необходимости перемещения фрагмента данных. Используемые в алгоритме обозначения приведены в табл. 1.

Таблица 1

Обозначения в пороговом алгоритме с ограничением по времени, объему и расстоянию

Table 1

Symbols definitions in the algorithm of threshold, time, volume, and distance constraint

Обозначение	Значение
F	Количество фрагментов всей распределенной базы данных
S	Число узлов распределенной базы данных
A	Ограничение доступа на перераспределение фрагментов
B	Ограничение времени на перераспределение фрагментов
A_p^q	p-я попытка доступа к узлу q
n_i^j	Общее количество попыток доступа от узла j к фрагменту i внутри временного интервала вплоть до текущего времени попытки доступа t
V_i^j	Размер данных, передаваемых между фрагментом i и узлом j внутри временного интервала вплоть до текущего времени t
D_i^j	Расстояние между узлами

Фаза подготовки и сбора исходных данных. В этой фазе производятся сбор статистических данных для определения необходимости перемещения фрагментов данных между узлами в базе данных.

Предположим, что имеется F фрагментов и S узлов в распределенной системе управления базами данных. Каждый узел имеет как минимум один расположенный на нем фрагмент. Для каждого узла формируется матрица, в которой сохраняется метрическое расстояние между узлами, которым может являться скорость передачи данных между узлами. Каждый узел также хранит значения двух ограничений: порог числа повторных доступов к фрагменту за единицу времени α и ограничение на время перераспределения фрагментов β. Эти параметры позволяют управлять задержкой перед переносом данных на другой узел и могут выбираться статически или динамически.

Процесс подготовки состоит из двух шагов.

Шаг 1. Расположить фрагменты распределенной базы данных в первоначальной расстановке на разных узлах, используя любой статичный метод распределения данных без использования избыточности.

Шаг 2. Написать таблицу логов доступа *Access_Log* для каждого узла. Она имеет следующую структуру: *Access_Log*(*AFID; ASID; ADateTime; DataVol*), где *AFID* – ID фрагмента, к которому осуществляется обращение; *ASID* – ID узла, инициирующего это обращение; *ADateTime* – дата и время обращения; *DataVol* – размер данных, переданных от и к фрагменту; каждый узел хранит записи логов для каждого обращения к фрагментам, расположенным на этом узле; каждая запись *Access_Log* обозначается как A_p^q – (p-е обращение на узле q, где $p = 1, 2, ..., \infty$ и $q = 1, 2, ..., S$).

Фаза принятия решения о переносе фрагмента. Фаза принятия решения о переносе фрагмента выполняется каждый раз после обращения к фрагменту данных на любом из узлов. Пусть обращение сделано от узла j к фрагменту i, расположенному на узле q во время t, где $i = 1, 2, ..., F$; $j = 1, 2, ..., S$; $q = 1, 2, ..., S$, причем $q = j$ или $q \neq j$. Локальная программа на узле q выполняет четыре шага при каждом обращении к фрагменту этого узла:

Шаг 1. Сделать запись A_p^q в *Access_Log* на узле q.

Ш а г 2. Если ID обращающегося узла в лог-записи A_p^q совпадает с ID узла q, то обращение локальное ($q = j$) и, следовательно, делать ничего не нужно.

Ш а г 3. Если ID обращающегося узла в *Access_Log* A_p^q не совпадает с *ID* узла q, значит обращение удаленное ($q \neq j$), следовательно, необходимо выполнить следующие подпункты.

Ш а г 3.1. Вычислить общее количество обращений к фрагменту i от всех узлов, включая локальные обращения, внутри временного интервала β вплоть до текущего времени обращения t. Предполагается, что n_i^s обозначает общее количество обращений от узла s к фрагменту i, расположенному на узле q внутри временного интервала β вплоть до текущего времени t, где $s = 1, 2, ..., S$.

Шаг 3.2. Вычислить средний размер данных в байтах, передаваемых между фрагментом i и всеми узлами, включая q, откуда были обращения к i внутри временного интервала β вплоть до текущего времени t. Пусть $A_p^q V_i^s$ обозначает размер данных в байтах, передаваемых между фрагментом i, расположенном на узле q, и узлом s в обращении A_p^q внутри временного интервала β вплоть до текущего времени t, где $s = 1, 2, ..., S$ и $p = 1, 2, ..., \infty$. Также пусть $V_i S_i$ обозначает средний размер данных, переданных между фрагментом i, расположенном на узле q, и узлом s внутри временного интервала β вплоть до текущего времени t

$$V_i^S t = \left(\sum A_p^q V_i^s\right) / \sum n_i^s .$$

Ш а г 3.3. Если общее количество обращений к фрагменту i от узла i внутри временного интервала β вплоть до текущего времени t больше, чем ограничение доступа α, то есть $n_i^j > \alpha$, где $j, s = 1, 2, ..., S$; $j \neq s$, и средний объем переданных между фрагментом i и узлом j данных в байтах внутри временного β вплоть до текущего времени t больше среднего объема переданных между фрагментом i и всеми узлами, включая q, то есть $V_i^j t > V_i^S t$, где $j, s = 1, 2, ..., S$,, то выполнить следующие подпункты.

Ш а г 3.3.1. Если обращающийся узел j не удовлетворяет ограничениям шага 3.3, то ничего не делать.

Ш а г 3.3.2. Если только один обращающийся узел j подходит под эти ограничения, то фрагмент i передислоцируется на узел j и удаляется из текущего узла q с попутным обновлением каталогов.

Ш а г 3.3.3. Если более чем один обращающийся узел j удовлетворяет ограничениям, обозначенным на шаге 3.3 в ходе одновременного обращения, то необходимо вычислить расстояние между узлом q, где располагается фрагмент i, и узлами, удовлетворяющими ограничениям, обозначенным на шаге 3.3. Выбрать узел с максимальной дистанцией от узла q и перенести фрагмент на этот узел, удалив этот фрагмент из q и обновив каталоги.

Преимущество порогового алгоритма – возможность динамической настройки момента переноса данных между узлами относительно двух параметров: времени и интенсивности запросов. Это достигается корректировкой размера временного окна и значения порогов срабатывания алгоритма относительно интенсивности поступающих внешних запросов. Основной недостаток – сложность выбора пороговых значений при скачкообразных изменениях интенсивности, что может привести к серьезному повышению накладных расходов на передачу данных без видимого приращения эффективности функционирования системы в целом.

Концепция распределенных торгов Mariposa

Подход Mariposa поддерживает функционально полный набор операций с фрагментами данных распределенной базы данных. Фрагментами являются единицы хранения данных, которые «покупаются» и «продаются» узлами. Текущий владелец фрагмента может в любое время разделить его на два фрагмента или наоборот, соединить два фрагмента в один. Идея подхода состоит в

применении механизма деятельности биржи для подготовки и проведения торговой операции: каждый участник заявляет о выполняемых им операциях, делает ставки на запрашиваемые операции, а централизованный брокер определяет, кому и когда выполнять запрос. Для определения параметров ставки используется эвристический алгоритм.

Каждый запрос Q имеет бюджет $B(t)$, который может использоваться для выполнения запроса. Бюджет – это не увеличивающаяся функция во времени, являющаяся оценкой стоимости, которую пользователь дает для ответа на свой запрос в конкретный момент времени t.

Брокер, обрабатывающий запрос Q, получает план запросов, содержащий множество подзапросов $\{Q_1; ...; Q_n\}$ и $B(t)$. Каждый подзапрос представляет собой ограничение одной переменной на фрагменте F таблицы или соединение двух фрагментов из двух таблиц. Брокер пытается решить каждый подзапрос Q_i, используя либо *протокол дорогих ставок*, либо более дешевый *протокол выставления заказа*. Решением является перечень узлов, которые будут выполнять подзапросы.

Протокол дорогих ставок. Реализация протокола дорогих ставок состоит из двух этапов. На первом этапе брокер отправляет запросы на участие в торгах. Запрос ставки включает в себя часть плана выполнения запроса, на который делается ставка. Участники торгов возвращают заявки, представленные в виде тройки C_i, D_i, E_i, указывающей, что претендент разрешит подзапрос Q_i для стоимости C_i в течение задержки D_i после получения подзапроса, и что эта ставка действительна только до истечения срока действия E_i.

На втором этапе протокола брокер уведомляет победивших участников торгов, что они были выбраны. Брокер также может уведомлять и проигравших. Если уведомления не произошло, то срок годности ставок истекает, и они могут быть удалены участниками торгов. Этот процесс требует обмена большим числом сообщений между узлами.

Протокол выставления заказа. Большинство запросов не требуют трудоемких вычислений. Чтобы оправдать такой уровень накладных расходов, они используют более дешевый и простой протокол выставления заказа, отправляющий каждый подзапрос на узел обработки, который несомненно выиграл бы процесс торгов, если бы в них участвовал. Этот узел получает запрос и обрабатывает его, возвращая ответ в форме «счета за услуги». Если узел отказывается от подзапроса, он может либо вернуть его брокеру, либо передать его на следующий по приоритету узел обработки. Если брокер использует более дешевый протокол заказа, существует некоторая опасность не найти решения для запроса в пределах выделенного бюджета. Брокер не всегда знает стоимость, которая будет «взиматься» с выбранного узла обработки, и временные задержки при передаче данных от этого узла. Однако риск компенсируется невысокими вычислительными затратами при реализации этого более быстрого протокола.

Протокол выбора плана. В процессе приема ставок все подзапросы на каждом шаге обрабатываются параллельно, тем не менее следующий шаг не может начаться до тех пор, пока предыдущий не завершен. Поэтому на каждом шаге вместо рассмотрения ставок для каждого отдельного подзапроса рассматриваются сразу множества ставок для подзапросов каждого шага.

При использовании протокола ставок брокеры должны выбирать выигрышную ставку для каждого подзапроса с совокупной стоимостью C и суммарной задержкой D так, чтобы суммарные затраты были меньше или равны затребованным затратам $B(D)$. Существует две проблемы, затрудняющие поиск наилучшего множества ставок: параллелизм подзапросов и комбинаторное пространство поиска. Так как общая задержка не является суммой задержек D_i каждого подзапроса Q_i, поскольку в каждом шаге плана запроса есть параллелизм и с большой вероятностью $D < \sum_i B_i(D)$. Кроме того, количество возможных множеств ставок растет экспоненциально с количеством шагов в плане запроса.

Предполагаемая задержка для обработки множества подзапросов на каждом шаге

равна наибольшему времени ставки в множестве. Количество различных значений задержек может быть не более, чем общее количество ставок по подзапросам в множестве подзапросов. Для каждого значения задержки оптимальное множество ставок состоит из наиболее дешевых ставок для каждого подзапроса, который может обрабатываться в течение данной задержки. Объединяя множества заявок в шаге и рассматривая их как совокупную ставку, брокер может свести проблему принятия ставки к более простой проблеме выбора одной ставки из числа агрегированных ставок для каждого шага.

При использовании протокола дорогих ставок брокер получает множества из нуля и более ставок для каждого подзапроса. Если для какого-либо подзапроса нет ставки, или не существует множества ставок, удовлетворяющих минимальным клиентским требованиям к цене и производительности из $B(D)$, то брокер должен запрашивать дополнительные ставки, соглашаться выполнить подзапрос самому или уведомить покупателя о том, что запрос не может быть запущен. Возможен случай, когда несколько множеств ставок удовлетворяют минимальным требованиям, поэтому брокер должен выбрать лучшее множество ставок. Чтобы сравнить коллекции ставок, определяется функция разности в множестве ставок – $B(D) - C$.

Протокол распределения ресурсов. Суть проблемы заключается в определении относительных количеств ресурсов времени и затрат, которые должны быть распределены для каждого подзапроса. В [6] для решения этих задач предложен эвристический алгоритм. Возможно, он не оптимален, но показывает хорошие результаты. Предложенный алгоритм является жадным. Он производит пробное распределение, в котором полная задержка является наименьшей из возможных, а затем делает обмены до тех пор, пока не перестанут появляться более выгодные решения. Таким образом, предлагается ряд решений с постоянно увеличивающимися значениями задержки для каждого этапа обработки. На любой итерации алгоритма предлагаемое решение содержит набор ставок с определенной задержкой для каждого этапа обработки. Для каждого набора ставок с большей задержкой вычисляется градиент затрат – снижение затрат, которое будет иметь место для этапа обработки путем замены коллекции в решении рассматриваемой коллекцией, поделенной на увеличение времени, которое, в свою очередь, было бы результатом замены.

Алгоритм начинается с рассмотрения множества ставок с наименьшей задержкой для каждого этапа обработки и вычисления общей стоимости C и общей задержки D. Вычисляется градиент стоимости для каждой неиспользованной ставки. Затем выполняется шаг обработки, который содержит неиспользованную ставку с максимальным градиентом затрат, B. Если эта ставка заменяет текущую, используемую на этапе обработки, то стоимость станет C с задержкой D. Если результирующая разница больше в D', чем в D, то необходимо сделать замену ставки. То есть если $B(D') - C' > B(D) - C$, то нужно заменить B на B', пересчитать все градиенты затрат для этапа обработки, включающие B', продолжить делать замены до тех пор, пока разность не перестанет увеличиваться.

Используя либо протокол дорогой ставки, либо протокола заказа, брокер должен иметь возможность определить один или несколько узлов обработки каждого подзапроса для поиска участников торгов. В Mariposa это достигается через систему объявлений.

Протокол объявлений о выполняемых действиях. Узлы заявляют о своей готовности выполнять различные услуги, размещая рекламные объявления. Серверы имен сохраняют эти рекламные объявления в таблицу объявлений. Брокеры просматривают таблицу объявлений, чтобы узнать, какие узлы могут выполнять задачи, в которых они нуждаются. В табл. 2 показаны поля таблицы объявлений.

Для обработки запросов на фрагментированных таблицах и поддержки купли, продажи, расщепления и объединения фрагментов алгоритмы подхода Mariposa делятся на три уровня, как показано на рис. 1.

Таблица 2

Обозначения, используемые в полях таблицы объявлений

Table 2

Symbols definitions in the fields of table notification

Поле таблицы	Описание
query_template	Описание предлагаемой услуги в виде запроса с пустыми параметрами
server_id	ID узла, предлагающего услугу
start_time	Время начала исполнения предлагаемой услуги (может быть в будущем)
expiration_time	Время, до которого предложение действительно
price	Цена, взимаемая узлом за услугу
delay	Время, за которое узел планирует выполнить услугу
limit_quantity	Максимальное количество выполнений услуги узлом по заданной цене и времени выполнения
bulk_quantity	Количество заказов, необходимых для реализации предложенной цены и времени выполнения
to_whom	Множество брокеров, которым доступно данное объявление
other_f ields	Комментарии и другая информация, специфичная для данного объявления

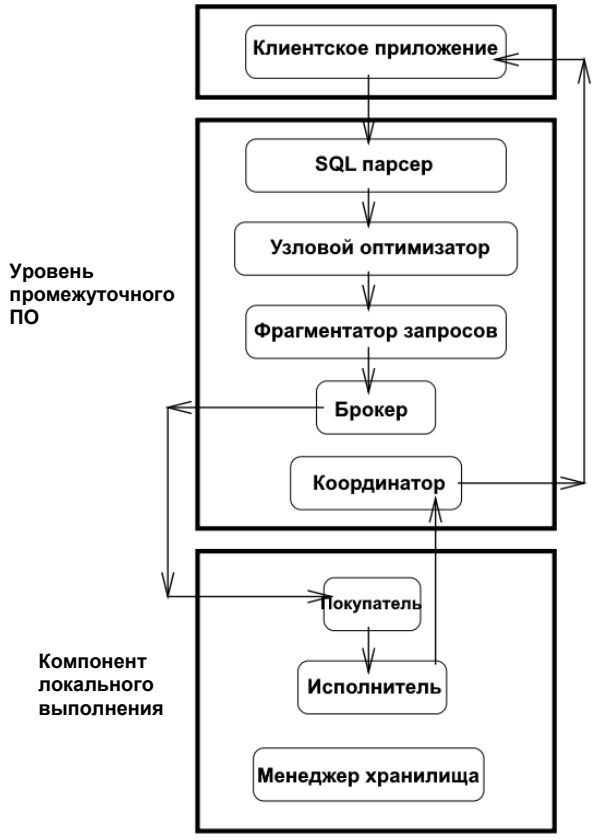

Рис. 1. Архитектура системы Mariposa
Fig. 1. The architecture of the system Mariposa

Архитектура разделяется на клиентскую программу, инициирующую запросы системе Mariposa, дополняя инструкциями по ценам и ставкам; слой промежуточного программного обеспечения; локальный узел исполнения. Уровень промежуточного программного обеспечения состоит из нескольких модулей подготовки запроса и брокера запросов. Локальный узел состоит из покупателя, менеджера хранилища данных и локального механизма исполнения запросов.

Достоинства предложенного подхода – возможность применения нескольких стратегий оценки заявки на выполнение запроса и технология иерархической агрегации заявок одного уровня в укрупненную заявку, что снижает вычислительную сложность построения плана запроса. К недостаткам следует отнести наличие централизованного брокера, что требует дополнительного времени для передачи заявок и несколько снижает надежность системы в целом.

Концепция одноранговых сетей в кластерной распределенной системе хранения

Концепт PeertoPeer сетей – потенциально наиболее эффективная архитектура для операций с данными. Алгоритм

используется в специфической кластеризованной архитектуре распределенных СУБД, названной FlexiPeer [3]. В этой архитектуре узлы кластеризованы по критериям приоритета локальности, каждый кластер управляется локальным администратором кластера (ЛАК), а вся архитектура администрируется глобальным администратором кластера (ГАК). Процесс кластеризации выполняется на основе количества узлов в регионе, которые заносятся в таблицу местоположения. В этой таблице имеется подробная информация о каждом узле: ID узла, местоположение и регион, по которому выполняется кластеризация узлов. ЛАК соответственного кластера обрабатывает атрибуты: локальный ID кластера (*LCA_id*), ID узла (*site_id*), регион кластера (*cluster_region*), местоположение узла (*site_location*) и тип данных, хранящихся на этом узле (*site_type*). Глобальные атрибуты всех кластеров (*GCA_id*, *LCA_id* и *cluster_region*) обрабатывает ГАК. Кластеризация помогает, если данные по запросу не найдены в конкретном узле: запрос будет перенаправлен к соответствующему ЛАК для дальнейшей обработки. Этот процесс называется *пересылкой неуспешных запросов*. Возрастающее количество таких запросов влияет на время обработки транзакции. Несмотря на то, что очень трудно избежать подобной ситуации, можно создать почву для сокращения числа неуспешных запросов, например, с помощью стратегии перераспределения для узлов одного кластера.

Перераспределение фрагментов осуществляется при помощи венгерского алгоритма оптимизации, где используется матрица стоимостей распределения, в которой полями является количество транзакций, обращающихся к данным конкретного узла. Количество обращений к конкретному узлу в матрице распределения записывается как вес обращения к конкретному фрагменту от конкретного узла. Функция стоимостей на основе венгерского алгоритма оптимизации для определения перераспределения фрагментов по узлам определяется в соответствии с выражением:

$$\min Z = \sum\sum C_{ij} \times X_{ij}.$$

Применяя венгерский алгоритм оптимизации, можно получить оптимальное распределение фрагментов по узлам.

Достоинство предложенного алгоритма заключается в получении нового решения за полиномиальное время, однако алгоритм предполагает жесткую иерархическую структуру подчиненности кластеров и отсутствие явного критерия перераспределения данных между узлами.

Прогнозирование состояния распределенной базы данных методами временных рядов

Для глобальной оптимизации с прогнозированием состояния распределенной базы данных применяется метод оценки будущего состояния базы данных на основе интегрированной модели авторегрессии и скользящего среднего (Бокса–Дженкинса) ARIMA. Алгоритм обеспечивает прогноз состояния базы данных и на его основе перемещение фрагментов базы данных из одного узла в другой, а также регулирует увеличение и уменьшение числа узлов, на которых хранятся фрагменты баз данных, т. е. осуществляет балансировку не только перераспределением объема данных по существующим узлам, но и изменением числа узлов. Алгоритм состоит из четырех фаз.

1. Выполняется прогнозирование на k шагов вперед для оценки рабочих нагрузок в ближайшие часы.

2. Выполняется корректировка количества узлов, в ходе которой выясняется, что необходимо сделать (добавить дополнительные узлы или удалить лишние).

3. Выбираются фрагменты с низкой стоимостью миграции для необходимых миграций.

4. Выполняется миграция фрагментов для балансировки нагрузки.

1: **For** $k = 1 : \omega$ **Do**
2: $\hat{n}_{t+k} \leftarrow \left\lceil \sum_{j=1}^{m} \hat{W}_{F_j,t}(k) / \theta_l \right\rceil$;
3: **EndFor**
4: **If** $n < \hat{n}_{t+1}$ **Then**
5: $n \leftarrow \hat{n}_{t+1}$;
6: добавить $\hat{n}_{t+1} - n$ узлов
7: **ElseIf** $n > max_{1 \le k \le \omega}(\hat{n}_{t+k})$ **Then**
8: $n_d \leftarrow max_{1 \le k \le \omega}(\hat{n}_{t+k})$;
9: **While** $n\_d < n$ **Do**
10: $N_r \leftarrow \{N_i : N_i \in \Omega_N, \nexists N_l \in \Omega_N$ такого, что $\hat{W}_{N_i,t}(1) > \hat{W}_{N_l,t}(1)\}$;
11: **ForAll** $F_j \in \Omega_F$ таких, что $a\_\{rj\} = 1$
12: $\Omega_M \leftarrow \Omega_M \cup \{F_j\}$;
13: $a_{rj} \leftarrow 0$;
14: **EndFor**
15: $n \leftarrow n - 1$;
16: $\Omega_N \leftarrow \Omega_N - \{N_r\}$;
17: **EndWhile**
18: **EndIf.**

Листинг 1. Фаза корректировки количества узлов
Listing 1. Node adjustment phase

Алгоритм корректировки числа узлов. Корректировка количества узлов, выполняемая в фазе 2, описана в листинге 1. Сначала оценивается требуемое количество рабочих узлов для следующих ω единиц времени: $\hat{n}_{t+1}, \hat{n}_{t+2}, ..., \hat{n}_{t+\omega}$ (шаги 1–3). Новые узлы добавляются, если количество существующих рабочих узлов недостаточно для следующего момента времени $n < \hat{n}_{t+1}$ (шаги 4–6). Некоторые узлы удаляются, если есть лишние работающие узлы для всех ω единиц времени $n > \max_{1 \le k \le \omega}(\hat{n}_{t+k})$ (шаг 7). В случае добавления узлов учитываются нагрузки в следующие моменты времени для обеспечения достаточного количества рабочих узлов в тот момент времени. Однако для удаления узлов учитываются все ω моменты времени для предотвращения появления ненужных трат на миграцию данных в течение следующих ω моментов времени.

В случае удаления нескольких узлов алгоритм итеративно выбирает узел для удаления с наименьшей нагрузкой в момент времени $t + 1$ (шаги 8–16). В [11] указано, что миграция фрагмента занимает от нескольких секунд до нескольких минут. Поскольку миграция любого фрагмента завершается за единицу времени (час), обозначим стоимость миграции фрагмента его нагрузкой в момент времени $t + 1$ и минимизируем общую стоимость миграции выбором узлов с низкой загрузкой в момент времени $t + 1$ с их последующим удалением.

Алгоритм оценки перегруженности узлов. После фазы корректировки количества узлов алгоритм проверяет, есть ли узлы, которые в следующий момент времени будут перегружены. Если это так, то некоторые фрагменты, принадлежащие этим узлам, выбираются в качестве перемещаемых. Фаза выбора таких фрагментов (фаза 3) изложена в листинге 2. Если узел работает со скоростью выше верхней границы θ_u рабочей нагрузки узла в момент времени $t + 1 - \hat{W}_{N_i}(1) > \theta_u$, алгоритм итеративно выбирает фрагмент для удаления из узла, пока нагрузка не упадет ниже θ_u. По возможности

выбранные фрагменты должны удовлетворять «ограничению 1» модели ARIMA [8] (шаги 3–5) для стабилизации узла после удаления из него фрагментов. Если таких фрагментов нет, то алгоритм использует основное правило выбора фрагментов с малой нагрузкой в момент времени $t + 1$ (шаг 7) для уменьшения стоимости миграции.

Алгоритм миграции фрагментов между узлами. На последнем шаге осуществляется этап миграции фрагментов. Фаза миграции фрагментов (фаза 4) описана в листинге 3. Фаза выполняется в случае, если некоторые фрагменты должны быть перенесены из своих начальных узлов. После сортировки фрагментов в порядке убывания нагрузок в момент времени $t + 1$ (шаг 1), алгоритм итеративно ищет узел назначения для каждого фрагмента. Для того чтобы избежать возникновения ненужных миграционных затрат, выбранные узлы проверяются на неперегруженность после получения новых фрагментов, что эквивалентно соблюдению условия $\hat{W}_{N_i}(1) > \theta_u$ (шаг 4). По возможности выбранные фрагменты должны удовлетворять «ограничению 2» модели ARIMA [8] (шаги 5–7), чтобы еще больше уменьшить вероятность ненужных затрат.

Положительной стороной такого подхода является построение прогноза использования данных на основе формальной модели анализа временных рядов и возможность динамического наращивания и исключения числа обрабатывающих данные узлов, что обеспечивает предсказательную способность и динамику алгоритма, отрицательной стороной – централизованный расчет модели, что требует дополнительных накладных расходов на поддержание актуальных данных о состоянии модели.

Эвристический подход к избыточному распределению фрагментов базы данных

Подход состоит в распределении фрагментов баз данных так, чтобы для каждого узла, выполняющего запрос доступа к данным, он занимал малое время, при условии исключения избыточности данных в базе данных в целом. В этом случае для более эффективного решения проблемы распределения необходимо не только определить наличие и количество копий каждого фрагмента на каждом узле, но и найти оптимальное распределение по узлам в соответствии с информацией о запросах, исключив избыточность.

1: **ForAll** $N_i \in \Omega_N$ **Do**
2: **While** $\hat{W}_{N_i,t}(1) > \theta_u$ **Do**
3: $]\Omega_T = \{F_j : F_j \in \Omega_F, a_{ij} = 1$ так, что $|\hat{G}_{N_i,t} - \hat{G}_{F_j,t}| \leq |\hat{G}_{N_i,t}|\}$;
4: **If** $\Omega_T \neq 0$ **Then**
5: $F_r \leftarrow \{F_j : F_j \in \Omega_T, \nexists F_l \in \Omega_T,$ такого, что $\hat{W}_{F_j,t}(1) > \hat{W}_{F_l,t}(1)$;
6: **Else**
7: $F_r \leftarrow \{F_j : F_j \in \Omega_F, a_{i,j} = 1, \nexists F_l \in \Omega_F,\ a_{il} = 1$ такого, что $\hat{W}_{F_j,t}(1) > \hat{W}_{F_l,t}(1)$;
8: **EndIf**
9: $\hat{W}_{N_i,t}(1) \leftarrow \hat{W}_{N_i,t}(1) - \hat{W}_{F_r,t}(1)$;
10: $\hat{G}_{N_i,t} \leftarrow \hat{G}_{N_i,t} - \hat{G}_{F_r,t}$
11: $\Omega_M \leftarrow \Omega_M \cup \{F_r\}$
12: **EndWhile**
13: **EndFor.**

Листинг 2. Фаза отбора фрагментов
Listing 2. Fragment selection phase

1: Отсортировать $F_j \in \Omega_M$ по убыванию $\hat{W}_{F_j,t}(1)$
2: **While** $\Omega_M \neq 0$ **Do**
3: $]F_r = F_j \in \Omega_M$ с наименьшим $\hat{W}_{F_j,t}(1)$
4: $]\Omega_T = \{F_j : F_j \in \Omega_F, a_{ij} = 1$ так, что $|\hat{G}_{N_i,t} - \hat{G}_{F_j,t}| \leq |\hat{G}_{N_i,t}|\}$;
5: $]\Omega_T = \{N_i : N_i \in \Omega_S$ так, что $\hat{G}_{N_i,t} + \hat{G}_{F_r,t} \leq 0\}$;
6: **If** $\Omega_t \neq 0$ **Then**
7: $N_d \leftarrow \{N_i : N_i \in \Omega_T, \nexists N_l \in \Omega_T$ такого, что $\hat{W}_{N_i,t}(1) > \hat{W}_{N_l,t}(1)\}$;
8: **Else**
9: $N_d \leftarrow \{N_i : N_i \in \Omega_S, \nexists N_l \in \Omega_S$ такого, что $\hat{W}_{N_i,t}(1) > \hat{W}_{N_l,t}(1)\}$;
10: **EndIf**
11: $\Omega_M \leftarrow \Omega_M - \{F_r\}$;
12: $a_{dr} \leftarrow 1$;
13: Перенести F_r на N_d;
14: **EndWhile**
15: Удалить все пустые узлы.

Листинг 3. Фаза миграции фрагментов
Listing 3. Fragment migration phase

Подход работает на модели, основанной на предположении о том, что имеется полносвязная сеть, состоящая из множества узлов $S = \{S_1, S_2, ..., S_m\}$, каждый и которых имеет объем C, лимит фрагментов FL и локальную БД, являющуюся частью распределенной базы данных. Ребро между двумя узлами S_i и S_j имеет атрибут CC_{ij}, представлено натуральным числом, обозначающим стоимость передачи единицы данных от S_i к S_j.

Каждый узел генерирует множество запросов $Q = \{Q_1, Q\_2, ..., Q_k\}$, рассматриваемых как наиболее часто выполняемые запросы, например 75 %, обрабатываемых в РБД. Каждый запрос Q_k может выполняться с любого узла с определенной частотой. Частоты выполнения k запросов на m узлах могут быть представлены как матрица $m \times k(QF_{ij})$. Пусть S содержит множество фрагментов $F = \{F_1, F_2, ..., F_n\}$, представляя собой разделение отношений в ходе фазы фрагментации РСБД.

Исходя из критерия оптимальности – минимальной стоимости записи каждого F_i на узел S_k и стоимости выполнения запроса F_i на S_k, и стоимости обновления F_i на всех узлах, где этот фрагмент записан, и стоимости передачи данных. Таким образом, решение проблемы оптимальности в соответствии с используемой моделью перераспределения может быть определено как минимизация стоимости связи в процессе переноса фрагмента F_i на узел S_k.

Описание используемых обозначений приведено в табл. 3.

Модель основана на применении данных о частоте запросов на обновление, связанной с распределением фрагментов по узлам. Для правильного распределения схемы каждый фрагмент должен быть прикреплен к одному или более узлам в РСБД на начальном этапе. Поэтому для заданного множества фрагментов разных размеров и количества сетевых узлов с определенным количеством исполняемых запросов проблема оптимального распределения заключается в нахождении распределения, минимизирующего общую стоимость обновления данных во всей сети без нарушения ограничений, накладываемых узлами.

Таблица 3

Обозначения, используемые в эвристической модели

Table 3

Symbols denotations in the heuristic model

Обозначение	Описание
M	Количество узлов РСБД
S_j	j-й узел
N	Количество фрагментов данных в РСБД
F_i	i-й фрагмент данных
K	Количество запросов в РСБД
RN	Количество связей
Q	Массив запросов
Q_k	k-й запрос
Q_i #	Количество запросов узла j
RF_{ij}	Частота обращения запроса на выборку i узла j
UF_{ij}	Частота обращения запроса на обновление i узла j
QF_{ij}	Частота обращения k-го запроса узла j
C_i	Емкость узла j
CC_{ij}	Стоимость передачи данных между узлами i и j
FL_j	Максимальное количество фрагментов данных на узле j
TC	Общая стоимость

В модели подразумевается, что один и тот же запрос на разных узлах воспринимается как разные запросы с разными частотными характеристиками. Однако для достижения оптимального динамического перераспределения используются частоты запросов всех распределенных фрагментов в целях возможного применения при каждом их перераспределении. Чтобы инициировать начальную фазу распределения фрагментов, которая может содержать дублирующие фрагменты, в предлагаемой методике считается, что фрагменты уже распределены по сетевым узлам на основе матрицы поиска RM и матрицы частот запросов QF, сохраняя при этом ограничения узлов. Предлагаемый подход заключается в том, что запрашиваемые фрагменты перемещаются на чаще всего запрашивающий их узел, и поэтому запросы будут обрабатываться без каких-либо затрат на передачу данных.

Используемая методика неизбыточна и применяется с использованием матрицы частот запросов UM, т. к. запрос на обновление данных включает в себя затраты на передачу данных и стоит больше, чем запрос на выборку данных и обработка частотной матрицы. Матрицы UM и QF будут использоваться в качестве входных данных для построения новой матрицы частот обновления фрагментов (Fragment Update Frequency Matrix – FUEM). FUEM может быть определена значениями запросов на обновление на конкретных узлах для вовлеченных фрагментов.

FUEM используется для формирования кумулятивной частотной таблицы обновлений (Cumulative Update Frequency Table – CUFT). С помощью CUFT и DM получается матрица оплаты фрагментов (Fragment Pay Matrix – FUPM). С помощью FUMP определяется узел с максимальной стоимостью обновления для конкретного фрагмента – такой узел является кандидатом на хранение фрагмента. Если имеются какие-либо нарушения ограничений для узла-кандидата, то выбирается следующий по стоимости. Процедура приоретизации (Fragments Priority, FP procedure) будет работать на всех узлах для гарантирования единственности текущего фрагмента.

Процедура FP показывает количество обращений к фрагменту: как много раз запросы каждого узла QS обратились к предполагаемому фрагменту. Используется, если к одному фрагменту F_i осуществляются запросы более чем с одного узла с одинаковыми значениями стоимости обновления этого фрагмента. Для исключения копий FP рассчитывается для каждого узла и используется для принятия решения о перемещении фрагмента F_i на узел с наивысшим FP. Последовательность шагов для перераспределения данных приведена на рис. 2.

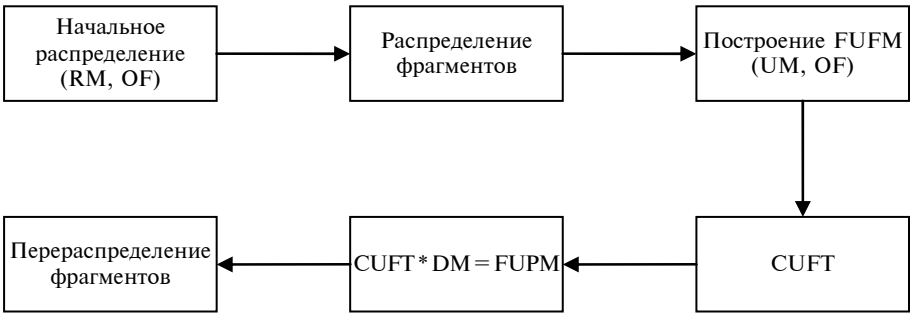

Рис. 2. Эвристический подход для перераспределения данных
Fig. 2. Redistributing data: heuristic approach

$$\mathrm{FP}(S_j, F_i) = \sum_{h=1}^{k} (\mathrm{QF}_{hi} \times \mathrm{QS}_{hi}), \quad 1 \le j \le m;$$

$$\mathrm{QS}_{hi} = \sum_{i=1}^{n} \mathrm{RM}_{hi} + \sum_{i=1}^{n} \mathrm{UM}_{hi}, \quad 1 \le h \le h.$$

Затраты вызваны запросами на выборку и обновление фрагментов на определенном узле. Для вычисления FUPM берется предыдущее значение FUPM на вход и используются следующие функции стоимостей:

$$\mathrm{CUFT} = \sum_{i=1}^{n} \sum_{j=1}^{m} (\mathrm{QF}_i \times \mathrm{UF}(F_j));$$

$$\mathrm{DM}_{ij} = \min(CC_{ij}), \quad 1 \le i, j \le m;$$

$$\mathrm{FUPM} = \sum_{j=1, j \ne j'}^{m} \mathrm{CUFT}(F_i) \times \mathrm{DM}_{jj'}, \quad 1 \le j, j' \le m;$$

$$\sum_{i=1}^{m} \max(\mathrm{FUPM}_{ij}), \quad 1 \le i \le n.$$

Процесс перераспределения может считаться оптимальным при достижении схемы распределения с минимальной стоимостью обновления. Поскольку эта проблема практически невычислима, предлагается эвристический алгоритм (листинг 4).

Предложенный эвристический подход выгодно отличается тем, что обеспечивает сбор появляющихся распределенных фрагментов данных в общий фрагмент, расположенный на одном узле. Это позволяет всегда выполнять операции добавления и обновления данных локально, не заботясь о сохранении фрагмента в наилучшем узле, что обеспечивает разделение процессов локальной работы системы управления базой данных и средств сетевой синхронизации. Недостаток предложенного метода – сложность реализации распределенной транзакционной модели: откат транзакции потребует перезапуска процедуры перераспределения данных.

Концепция автоматов с памятью, взаимодействующих с внешней средой

Цель применения обучающегося автомата в том, чтобы выбирать оптимальное действие с максимальной вероятностью вознаграждения за счет многократного взаимодействия с внешней средой. Если алгоритм обучения выбран правильно, то итеративно повторяющийся процесс может привести к выбору оптимального действия [13].

Автомат выбирает одно из предлагаемых действий в соответствии с вероятностным вектором, который в любой момент содержит вероятность выбора каждого действия. Выбранное действие активизирует среду, инициируя ее ответ (вознаграждение или штраф), в зависимости от вероятности вознаграждения выбранного действия. Автомат учитывает этот ответ и модифицирует вероятностный вектор с помощью алгоритма обучения. Обучающийся автомат ищет действие с максимальной вероятностью быть вознагражденным и в конечном итоге выбирает это действие чаще, чем другие действия.

1: Input:
2: Для каждой связи {1, ..., *K*}
3: $Q^N = \{Q_1, ..., Q_n\}$ множество запросов
4: $F = \{F_1, ..., F_n\}$ фрагменты данных в РБД
5: $S = \{S_1, ..., S_m\}$ множество сетевых узлов
6: матрицы *RM*, *UM*, *QF* и матрица стоимостей передачи данных

7: **Output:**
8: Схема перераспределения фрагментов

9: **Code:**
10: Начальное распределение данных с использованием *RM*, *QF*
11: Строится таблица *FUPM* с использованием *UM* и *QF*
12: Матрица стоимостей расстояний (*Distance* Cost *Matrix, DM*)= *min*(матрица стоимостей передачи данных)
13: **ForAll**узел $s \in \{1, ..., m\}$ **Do** // в *FUPM*, для получения нового *FUPM*
14: **ForAll** фрагмент $F_i \in R$; $1 \le i \le n$
15: Вычислить запрос на обновление F_i
16: **EndFor**
17: **ForAll**узел $s \in \{1, ..., m\}$ **Do**
18: Вычислить $FUPM(F_i)$
19: **EndFor**
20: Выбрать значение (υ), так что $Pay_v(F_i) = Max_s^m$ поиск узла с максимальной стоимостью обновления фрагмента (S_j);
21: Перенос F_i на $S_j(\upsilon)$;
22: **If** *S_j.constraints* (ограничения на узле S_j) были нарушены **Then**
23: Перейти на F_i на S_{j+1} со второй по величине стоимостью обновления
24: **EndIf**
25: **If** фрагмент F_i требуется нескольким узлам **Then**
26: Выполнить процедуру *FP* на всех узлах, где требуется F_i
27: Перенести фрагмент на S_j с максимальным *FP*
28: **EndIf**
29: **EndFor**.

Листинг 4. Эвристический алгоритм перераспределения фрагментов данных
Listing 4. The data redistribution heuristic algorithm

Если РБД состоит из коллекции *m* узлов $S = \{c_1, c_2, ..., c_m\}$, где каждый узел *i* характеризуется его объемом c_i и множество *n* фрагментов $F = \{s_1, s_2, ..., s_n\}$, то каждый фрагмент *j* характеризуется размером s_i. Каждый фрагмент востребован как минимум одним из узлов. Востребованность каждого фрагмента представляется в виде матрицы, где значения на пересечении узлов означают востребованность фрагмента *i* узлом *j*, которая обычно обозначается логическими нулем или единицей.

Стоимости передачи данных отражаются в матрице T, где t_{ij} обозначает стоимость доступа узла *i* к фрагменту, расположенному на узле *j*. Таким образом, проблема распределения в РБД сводится к задаче поиска оптимального размещения фрагментов на узлах, т. е. поиск размещения $P = \{p_1, p_2, ..., p_j, ..., p_n\}$, где равенство $p_j = i$ означает, что фрагмент *j* расположен на узле *i* для *n* фрагментов так, что объем каждого узла не переполнен. Общая стоимость передачи в этом случае составит $\sum_{i=1}^{m}\sum_{j=1}^{n} r_{i,j} \times t_{i,p_j}$, где минимизировано

$$\sum_{j=1}^{n} r_{i,j} \times s_j \le c_i \quad \forall i \,|\, 1 \le i \le m.$$

Ограничивая использование матрицы R и имея нулевую стоимость передачи, проблема распределения данных в РБД сводится к задаче об упаковке в контейнеры, т. е. к NP-полной задаче [9].

В РБД с m узлами и n фрагментами количество решений в области поиска велико. Если использовать обучающийся автомат для решения проблемы распределения фрагментов данных, у автомата будет слишком много возможных действий, что уменьшит скорость нахождения решения в автомате. Для устранения этой проблемы можно применять обучающийся автомат с миграцией объектов [14, 15]. Для решения проблемы используется автомат миграции объектов, обозначаемый как $<V, \alpha, \varphi, \beta, F, G>$. В этом автомате $\alpha = \{\alpha_1, \alpha_2, ..., \alpha_n\}$ представляет собой множество из n возможных действий автомата, причем количество действий равно количеству фрагментов данных. $V = \{V_1, V_2, ..., V_n\}$ представляет собой множество объектов, которые являются номерами узлов, на которых можно расположить фрагменты данных, а объекты принимают значения 1, 2, ..., m. Объекты перемещаются по различным состояниям автомата и создают новые распределения, $\varphi = \{\varphi_1, \varphi_2, ..., \varphi_{nN}\}$ – множество состояний, N – глубина памяти автомата. Множество состояний автомата подразделяется на k подмножеств $\{\varphi_1, \varphi_2, ..., \varphi_N\}$, $\{\varphi_{N+1}, \varphi_{N+2}, ..., \varphi_{2N}\}$, ..., $\{\varphi_{(k-1)N+1}, \varphi_{(k-1)N+2}, ..., \varphi_{kN}\}$ так, что объекты классифицируются в терминах их состояний. Если объект u расположен в множестве состояний $\{\varphi_{(j-1)N+1}, \varphi_{(j-1)N+2}, ..., \varphi_{jN}\}$, то в этом случае фрагмент данных j будет включен в u-й узел. В множестве состояний действия j состояния $\varphi_{(j-1)N+1'}$ и φ_{jN} называются соответственно внутренним и граничным состояниями. Объекты, лежащие в $\varphi_{(j-1)N+1'}$ и φ_{jN}, называются более и менее определенными объектами. Множество входов автомата $\beta = \{0,1\}$, где единица и ноль означают отказ и успех соответственно; $F: \varphi \times \beta \rightarrow \varphi$ обозначает функцию отображения состояний, которая на основании текущего состояния и входов автомата производит следующее состояние, тем самым определяя движение объектов по состояниям автомата.

Функция F различна для различных автоматов. $G: \varphi \rightarrow \alpha$ – выходная функция отображения, решающая, какое действие предпринять в ответ на любое состояние автомата. Если объект u в множестве состояний $\{\varphi_{(j-1)N+1}, \varphi_{(j-1)N+2}, ..., \varphi_{jN}\}$, выбирается действие j и, следовательно, фрагмент j будет на u-м узле.

Задача распределения реализована в алгоритме, представленном в листинге 5.

Алгоритм начинает работу с создания фрагментов данных со случайным распределением на узлах, с учетом ограничений на емкость узлов для выравнивания нагрузки. Для начала берется несколько узлов, у которых объекты действий автомата находятся в граничном состоянии. Затем выбирается случайный объект и, соответственно, получается либо штраф, либо вознаграждение. Общая стоимость переноса фрагмента данных на заданный узел вычисляется соотношением

$$cost(data\_fragment) = \sum_{i=1}^{m} r_{i,data\_fragment} \times t_{i,p_{data\_fragment}}.$$

Затем пороговое значение вычисляется как средняя общая стоимость передачи фрагментов данных. Если стоимость передачи фрагментов данных меньше или равна пороговому значению, то объект получает вознаграждение, иначе – штраф. Затем эти действия продолжаются для другого случайного объекта. Алгоритм продолжает выполняться, пока не выполнится условие остановки. Под условием остановки понимается превышение числа итераций алгоритма выше порога итераций, или достижение оптимального распределения, или локализация всех объектов во внутренних состояниях.

1: n = количество фрагментов в системе;
2: m = количество узлов в системе;
3: Случайное начальное распределение σ с учетом балансировки нагрузки на узлах;
4: Назначить узлы как объекты к граничным состояниям соответствующих действий автомата; *EvalFitness*(σ); // σ = начальное распределение
5: *EvalFitness*(σ) // начальное распределение
6: *iteration* = 1;
7: **Do**
8: **For** *u* **Do**//1 ≤ *u* ≤ *n*, *u* − случайный фрагмент
9: **If** *cost*(*u*) ≤ *ограничение* **Then**
10: *reward*(*u*)

//здесь ограничение = $(\sum_{i=1}^{m}\sum_{j=1}^{n} r_{i,j} \times t_{i,p_j}) / n$

// *cost*(*data_fragment*) = $\sum_{i=1}^{m} r_{i,data\_fragment} \times t_{i,p_{data\_fragment}}$

11: **Else**
12: *penalize*(*u*)
13: **EndIf**
14: **EndFor**
15: *Inc*(*iteration*)
16: *EvalFitness*(σ)
17: **Until** (*iteration* = *Max_iterations* или проход всех объектов в состояниях максимальной достоверности).

Листинг 5. Процедура распределения данных
Listing 5. The data distribution procedure

Преимуществом автоматного подхода с памятью, основанного на анализе вознаграждения или штрафа является возможность непрерывного анализа текущего состояния распределения фрагментов базы данных и определение наилучшего направления перераспределения за квадратичное время относительно числа узлов, что хорошо работает на базах данных с небольшим числом узлов, в то же время алгоритм является централизованным, что повышает накладные расходы на анализ состояний.

Эволюционный подход к миграции данных для выполнения операций выборки

Эволюционный подход состоит в постепенном перемещении данных, чаще всего участвующих в одном запросе на выборку, на один узел базы данных, в котором он чаще всего используется. Концепция состоит в анализе запросов на выборку данных с последующей реорганизацией размещения.

В процессе оптимизации размещения рассматриваются две стратегии обработки запроса.

- MoveSmall: если бинарная операция для двух отношений, например, объединение, включает в себя два фрагмента данных, расположенных на двух разных узлах, то необходимо доставить меньший фрагмент данных на узел большего фрагмента данных;
- QuerySite: отправить все фрагменты данных на узел источника запроса и выполнить запрос.

Цель распределения данных − минимизация общей стоимости передачи данных для обработки всех запросов с использованием одной из перечисленных выше стратегий выполнения. Первая задача распределения данных − максимизация локаль-

ности фрагментов для выполнения запросов. Вторая задача – в нужный момент выбрать стратегию выполнения запросов во время попытки доступа к фрагментам с нескольких узлов и снизить общую стоимость передачи данных для обработки всех запросов.

Рассматривается мультимедийная РБД с m узлами, каждый из которых имеет собственные вычислительные мощности, память и локальную БД. Пусть S_i – название узла i, где $0 \le i \le m - 1$. Связь между двумя узлами S_i и $S_{i'}$ (если он существует) имеет связанное с ней натуральное число $c_{ii'}$, обозначающее стоимость передачи единицы данных из узла S_i на узел $S_{i'}$. Если два узла не связаны напрямую, то стоимость передачи единицы данных будет являться суммой стоимостей связей на выбранном пути от S_i до узла $S_{i'}$. $Q = \{q_0, q_1, ..., q_{n-1}\}$ – рассматривается как множество наиболее выполняемых запросов – 80 % обработки в РБД. Каждый запрос q_x может выполняться из любого узла S_i с определенной частотой a_{ix}. Частоты выполнения n запросов на m узлах могут быть показаны матрицей $A(m \times n)$. Рассмотрим k фрагментов данных $Q = \{q_0, q_1, ..., q_{n-1}\}$.

Таблица 4

Обозначения, используемые в эволюционном алгоритме

Table 4

Symbols denotations in the evolutionary algorithm

Обозначение	Описание
m	Количество узлов РСБД
S_i	i-й узел
k	Количество фрагментов данных в РСБД
O_j	j-й фрагмент данных
n	Количество запросов в РСБД
Q	Множество запросов
q_x	x-й запрос
A	Матрица частот доступа
a_{ix}	Частота обращения x-го запроса к узлу i
C	Матрица стоимостей передачи единицы данных
$c_{ii'}$	Стоимость передачи единицы данных от узла i узлу i'
l	Вектор лимитов распределения по узлам
l_i	Лимит распределения для узла i
R	Матрица размеров передаваемых данных запросов
r_{xj}	Размер передаваемых данных фрагмента j запроса x
U	Матрица размеров передаваемых данных узлов
$u_{jj'}$	Размер передаваемых данных фрагмента j узлу j'
D	Матрица зависимостей фрагментов данных
d_{ij}	Размер передаваемых данных от узла фрагмента i узлу фрагмента j
t	Общая стоимость передачи данных

Пусть r_{xj} – размер фрагмента данных Q_j, который необходимо передать на узел S_j, откуда был a_{ix} раз за единичный интервал времени инициирован запрос q_x. Соответствующей матрицей является R размера $m \times k$, а матрица U размера $m \times k$ будет содержать в себе u_{ij}, отображающие размеры данных, которые необходимо перенести из узла расположения фрагмента O_j на узел S_i, откуда был инициирован запрос. Таким образом, $u_{ij} = \sum_{x=0}^{n-1} a_{ix} \times r_{xj}$.

В матричном представлении, $U = A \times R$.

В этом случае размер требуемого фрагмента зависит от распределения других фрагментов данных. Пусть $d_{jj'}$ – размер фрагмента данных $O_{j'}$, который необходимо передать на узел, где располагается $O_{j'}$, для выполнения определенных бинарных операций. Соответствующая матрица размера $k \times k - D$. Всё зависит от обрабатываемого запроса, поэтому для каждого запроса необходима информация о том, как много данных должно быть передано от узла, где располагается один фрагмента данных, на узел с другим фрагментом, с учетом того, что оба фрагмента требуются для выполнения запроса.

Пусть $\delta_{jj'}^{x}$ – размер данных O_j, необходимых для передачи на узел, где расположен $O_{j'}$, для обработки q_x, соответствующая матрица Δ_x. Количество данных для передачи из узла, на котором располагается O_j, на узел расположения $O_{j'}$ задается следующим образом:

$$d_{jj'} = \sum_{x=0}^{n-1} (\sum_{i=0}^{m-1} a_{ix}) \delta_{jj'}^{x}.$$

В матричном представлении:

$$D = \sum_{x=0}^{n-1} (\sum_{i=0}^{m-1} a_{ix}) \nabla^{x}.$$

Пусть $site(O_j)$ обозначает узел, где расположен O_j. Следовательно, общая стоимость передачи t задается как

$$t = \sum_{j=0}^{k-1} \sum_{j'=0}^{k-1} c_{site(O_j),site(O_{j'})} \times d_{jj'} + \sum_{i=0}^{m-1} \sum_{j=0}^{k-1} c_{i,site(O_j)} \times u_{ij},$$

где первое слагаемое обозначает стоимость передачи данных, требующейся для обработки бинарных операций между фрагментами разных узлов, а второе слагаемое – стоимость передачи данных для отправки результатов бинарных операций на узел, инициировавший запрос. Задача распределения данных минимизация t путем изменения функции $site(O_j)$, сопоставляющей фрагмент данных с определенным узлом.

Предлагается четыре эвристических алгоритма в поисковых техниках, хорошо показавших себя в области решения широкого круга задач оптимизации. Первый алгоритм основан на традиционном генетическом алгоритме. Второй – на технологии симулированной эволюции (использование мутаций, скрещиваний и кодирования информации). Третий базируется на технике комбинации нейронных сетей и алгоритма имитации отжига. Четвертый алгоритм основан на технике случайного поиска.

Генетический алгоритм поиска. Генетические методы поиска [16, 17] получены по аналогии с механизмами натуральной генетики, приводящей к выживанию наиболее приспособленных индивидов. ГА манипулируют популяцией потенциальных решений задачи оптимизации.

В предложенном ГА для решения проблемы распределения данных производится кодирование в бинарном представлении назначений каждого фрагмента данных узлам. Эти значения объединяются в одну двоичную строку. Каждая двоичная строка представляет потенциальное решение задачи распределения данных. «Пригодность» (fitness) строки – это стоимость распределения. Механизм селекции реализуется как простая схема пропорционального выбора: строке с пригодностью f выделяется потомство $f/(\bar{f})$, где $\bar{f}$ – среднее значение пригодности популяции. Строке с пригодностью выше среднего значения выделяется более одного потомства, ниже – менее.

Скрещивание (*Crossover*) – тоже важная операция ГА. Пары строк берутся случайно из популяции и подвергаются скрещиванию. Пусть L – длина строки, тогда алгоритм

случайно выбирает точку скрещивания, принимающую значения в диапазоне от единицы до $L - 1$ с равной вероятностью. Части двух строк за пределами этой точки скрещивания меняются местами, образуя две новые строки. Существует параметр p_c, характеризующий частоту скрещиваний.

После скрещивания строки подвергаются мутации. Под мутацией бита подразумевается его обращение. Так же как и в скрещивании, существует параметр p_m, характеризующий частоту мутаций битов. Биты строки мутируют независимо. Соответственно, мутация бита не подвержена влиянию вероятности мутации других битов.

Сложность распределения с помощью ГА сводится к $O(GN_p(k^2 + km))$, где G – количество поколений, а N_p – размер популяции.

Алгоритм симулированной эволюции. Это вариант традиционного ГА [18, 19]. Имитация отжига, ГА и эволюционные стратегии одинаковы в использовании вероятностных поисковых механизмов, направленных на уменьшение или увеличение целевой ценовой функции. Эти три метода имеют высокую вероятность нахождения оптимального общего решения, несмотря на то, что функция стоимостей мультимедиа имеет несколько локальных оптимальных решений. Однако каждый метод имеет существенную разницу в режимах работы: имитация отжига вероятностно генерирует последовательность состояний, чтобы в конечном итоге сойтись к глобальному оптимуму. Эволюционные стратегии используют мутации как поисковые механизмы и для выбора направления поиска в предполагаемых областях поискового пространства. ГА генерирует последовательность популяции с использованием механизмов селекции и использует скрещивание и мутацию в поисковых механизмах (листинг 6).

Принципиальная разница между генетическим алгоритмом и эволюционной стратегией состоит в том, что первый полагается на скрещивание, механизм вероятностного и полезного обмена информацией между решениями для нахождения лучшего решения, а последний использует мутации как первичный поисковый механизм (см. листинг 7).

Количество генов в *a* равно общему лимиту распределения, а количество генов в *b* равно общему количеству фрагментов.

Для *a* каждый ген есть один бит. Если бит равен единице, для этой хромосомы разрешено использовать соответствующее пространство распределения. Если бит равен нулю, пространство использовать нельзя. Это уменьшает эффективный лимит распределения для всех узлов.

1: Инициализация популяции. Каждый индивид популяции является конкатенацией двоичных представлений случайного начального распределения каждого фрагмента данных;
2: Вычислить популяцию;
3: no_of_generation = 0;
4: **While** *no_of_generations* < *MAX_GENERATION* **Do**
5: Выбрать индивидов для следующей популяции;
6: Провести скрещивание и мутацию для этих индивидов;
7: Вычислить популяцию;
8: *no_of_generations* + +;
9: EndWhile
10: Определить конечное распределение путем выбора наиболее подходящего (приспособившегося) индивида. Если конечное распределение невозможно, то необходимо рассмотреть каждый перенаселенный узел и перенести фрагменты на другие узлы с минимальным приростом цены.

Листинг 6. Генетический алгоритм распределения данных
Listing 6. The data distribution genetic algorithm

1: Построить первую хромосому на основе входных данных и заставить хромосому сгенерировать начальную популяцию;
2: Использовать эвристики отображения для генерации решения по каждой хромосоме;
3: Вычислить полученные решения;
4: *no_of_generations* = 0;
5: **While** *no_of_generations* < *MAX_GENERATION* **Do**
6: Выбрать хромосомы для следующей популяции;
7: Выполнить скрещивание и мутации для этих наборов хромосом;
8: Использовать эвристики отображения для генерации решения по каждой хромосоме;
9: Вычислить полученные решения;
10: *no_of_generations* + +;
11: **EndWhile**
12: Вывод лучшего найденного решения.

Листинг 7. Алгоритм симулированной эволюции для распределения данных
Listing 7. The stimulated evolution algorithm of data distribution

Для *b* каждый ген – это целое число, означающее приоритет рассматриваемого фрагмента: чем больше значение, тем выше приоритет.

Используя матрицу стоимости сетевой передачи *C* частоты доступа *b* размеров передаваемых данных, мы можем вычислить матрицу стоимостей распределения фрагментов данных по узлам *U'*, задаваемую как $U' = C \times U$. То есть каждый элемент *U'*, обозначаемый uu_{ij}', представляет собой стоимость распределения по узлам, возникшую при размещении фрагмента *j* на узле *i*.

В алгоритме симулируемой эволюции первая хромосома начальной популяции строится на основании информации u_{ij}' из таблицы. Для каждого фрагмента *j* вычисляется $x_j = \sum_{i=1}^{m} u_{ij}'$.

Идея в минимизации стоимости распределения путем задания высокого приоритета фрагменту с бóльшей *j*. Большое значение *j* означает, что этот фрагмент затратит больше времени на передачу, т. е. будет больше стоить. Таким образом, просто назначается *j* в качестве генов для каждой позиции фрагментов в части *b* первой хромосомы в начальной популяции.

Все гены в части *a* задаются равными единице для первой хромосомы в начальной популяции, т. к. это предел распределения исходной задачи. Для оставшихся хромосом в начальной популяции гены в *a* выбраны случайно (ноль или единица). Гены в *b* являются искажениями соответствующих генов в *b* у первой хромосомы.

Эвристика отображения состоит в том, что для каждой хромосомы ищется решение, перенося фрагмент *j* с наивысшим приоритетом на узел *i*, таким образом, чтобы u_{ij}' был наименьшим для всех u_{ij}'. $1 < x < m$. Если лимит распределения, включенный в гены части *a* хромосомы этого узла, превышен, т. е. узел переполнен, то этот фрагмент переносится на узел со следующим наименьшим значением u_{ij}' для всех u_{ij}', $1 < y < m$, $y \neq x$. Фрагмент гарантированно будет распределен на какой-нибудь узел, т. к. было проверено, что общий лимит распределения больше или равен общему количеству фрагментов для каждого поколения хромосомы.

Для каждой хромосомы функция стоимости – это общая стоимость передачи после распределения всех фрагментов на некоторые узлы с использованием эвристики отображения. Вычисление этой стоимости передачи учитывает зависимости между

фрагментами и размеры фрагментов, необходимые для каждого узла. Значение пригодности для хромосомы вычисляется как

$$f(i) = \frac{(\text{MaxCost} - \text{Cost}(i))^{\tau}}{\sum_{i=1}^{N_p}(\text{MaxCost} - \text{Cost}(i))^{\tau}},$$

где N_p – размер популяции; MaxCost – максимальная стоимость среди хромосом в популяции; τ – коэффициент сходимости, использующийся для контролирования скорости сходимости. Сложность алгоритма составляет $O(GP(k^2 + km))$.

Алгоритм отжига среднего поля. Техника отжига среднего поля (Mean Field Annealing – MFA) [3, 5] комбинирует свойство коллективного вычисления нейронной сети Хопфилда (Hopfield Neural Network – HNN) с техникой имитации отжига. MFA первоначально была создана для решения задачи коммивояжера как альтернатива HNN, недостаточно масштабируемой для крупных задач. Доказано, что MFA является общим подходом [3, 4], который может быть применен к различным задачам комбинаторики.

Алгоритм MFA [3] получен из аналогии с моделью спинов Изинга, использующейся для оценки состояния системы частиц или спинов в тепловом равновесии. В этой модели энергия системы с S спинами имеет вид:

$$H(s) = \frac{1}{2}\sum_{x=1}^{S}\sum_{y\neq x}\beta_{xy}s_x s_y + \sum_{x=1}^{S}h_x s_x,$$

где β_{xy} представляет собой уровень взаимодействия между спином x и y; $s_x \in 1,0$ – значение спина x. Предполагается, что $\beta_{xy} = \beta_{yx}$ и $\beta_{xx} = 0$ для $1 \le x, y \le S$. В тепловом равновесии среднее значение $\langle s_x \rangle$ спина x может быть вычислено с помощью распределения Больцмана:

$$\langle s_x \rangle = \frac{1}{1 + e^{-\Phi_x/T}},$$

где $\Phi_x = \langle H(s) \rangle|_{sx=0} - \langle H(s) \rangle|_{sx=1}$ представляет собой среднее поле, действующее на спин x, где средняя энергия системы $\langle H(s) \rangle$:

$$\langle H(s) \rangle = \sum_{x=1}^{s}\sum_{y\neq x}\beta_{xy}\langle s_x s_y \rangle + \sum_{x=1}^{s}h_x\langle s_x \rangle.$$

Сложность вычисления Φ_x с использованием этого уравнения экспоненциальная. Однако для большого числа спинов приближение среднего поля может использоваться для вычисления средней энергии:

$$\langle H(s) \rangle = \frac{1}{2}\sum_{x=1}^{s}\sum_{y\neq x}\beta_{xy}\langle s_x \rangle\langle s_y \rangle + \sum_{x=1}^{s}h_x\langle s_x \rangle.$$

Следовательно, $\langle H(s) \rangle$ линейна в $\langle s_x \rangle$, среднее поле Φ_x может быть вычислено через

$$\Phi_x = \langle H(s) \rangle|_{sx=0} - \langle H(s) \rangle|_{sx=1} =$$

$$= \frac{\partial\langle H(s) \rangle}{\partial\langle s_i \rangle} = -(\sum_{y\neq x}\beta_{xy}\langle s_y \rangle + h_x).$$

Таким образом, сложность вычисления Φ_x сводится к $O(S)$.

При каждой температуре, начиная с начальных средних спинов, вычисляется среднее поле x, действующее на случайно выбранный спин. После этого обновляется среднее значение спина. Этот процесс повторяется для случайной последовательности спинов, пока система не стабилизируется на текущей температуре (см. листинг 8).

Проблема распределения данных формулируется через MFA следующим способом (см. листинг 9). Вместо линейного массива спинов используется $x \times m$ матрица спинов s_{ij} для кодирования распределения фрагментов данных по узлам (x фрагментов данных по m узлам). Единица в записи означает, что фрагмент расположен на соответствующем узле. Правильное распределение – такое, когда в каждой строке матрицы спинов есть только одна единица. Каждая переменная спина непрерывна в диапазоне [0,1]. Значения спинов сходятся к единице или нулю в фиксированной точке. Таким образом, функция энергии в интерпретации стоимости передачи данных для задачи распределения данных может быть формализована следующим образом:

$$E(s) = \sum_{i=0}^{k-1}\sum_{i'=0}^{k-1}\sum_{j=0}^{m-1}\sum_{j'=0}^{m-1}c_{jj'}d_{ii'}s_{ij}s_{i'j'} + \sum_{j=0}^{m-1}\sum_{i=0}^{k-1}u_{ji}s_{ij}.$$

1: Получить начальную температуру T_0, задать $T = T_0$;
2: Инициализировать средние значения спина $\langle s \rangle = [\langle s_1 \rangle, ..., \langle s_s \rangle]$;
3: **While** T в диапазоне охлаждения **Do**
4: **While** система не стабилизирована на текущей температуре **Do**
5: Выбрать случайный спин x;
6: Вычислить x;
7: Обновить $\langle s_x \rangle$;
8: **EndWhile**
9: Обновить T в соответствии с расписанием охлаждения;
10: **EndWhile.**

Листинг 8. Общий алгоритм MFA
Listing 8. General MFA algorithm

1: Получить начальную температуру T_0, задать $T = T_0$;
2: Инициализировать средние значения спина $s = [s_{00}, s_{01}, ..., s_{k-1,m-1}]$, каждый s_{ij} инициализирован как случайное число между 0 и 1;
3: **While** температура T в диапазоне охлаждения **Do**
4: **While** E уменьшается **Do**
5: Выбрать случайный фрагмент данных i;
6: Вычислить среднее поле спинов $(\Phi_{ij}, \forall j)$ в строке i;
7: Вычислить сумму $\sum_{j'=0}^{m-1} e^{\Phi_{ij}/T}$;
8: Вычислить новые значения спина в строке i;
9: Вычислить изменения энергии исходя из вышеуказанных вычислений;
10: **EndWhile**
11: Обновить температуру T в соответствии с расписанием охлаждения;
12: **EndWhile**
13: Определить окончательное распределение, помещая каждый фрагмент данных на узел с максимальным значением спина.

Листинг 9. Алгоритм MFA распределения данных
Listing 9. The MFA algorithm of data distribution

Используя приближение среднего поля, выражение для Φ_{ij}, влияющего на спин s_{ij}, становится:

$$\Phi_{ij} = \frac{\partial E(s)}{\partial s_{ij}} = -\sum_{i' \neq i}^{k-1} \sum_{j' \neq j}^{m-1} c_{jj'} d_{ii'} s_{i'j'} - u_{ji}.$$

Сумма спинов каждой строки должна быть равна единице, т. е. каждый фрагмент данных должен быть распределен исключительно на один узел. Этого можно достичь, нормализируя каждый спин s_{ij}:

$$s_{ij} = \frac{e^{\Phi_{ij}/T}}{\sum_{j'=0}^{m-1} e^{\Phi_{ij'}/T}}.$$

Если окончательное распределение невозможно, то необходимо рассмотреть каждый перегруженный узел, чтобы перенести фрагменты данных на другие узлы с наименьшим приростом стоимости. Общая сложность алгоритма MFA $O(BK(k^2 + km))$.

Алгоритм случайного поиска. Используется не сложная, но эффективная оптимизационная техника, называемая поиском случайной окрестности (см листинг 10). Основная идея этого алгоритма – сгенерировать начальное решение среднего качества. Затем, с учетом предопределенной окрестности, алгоритм вероятностно выбирает и тестирует соседнее решение в области поиска на предмет того, лучше оно или нет. Если лучше, то алгоритм принимает его и начинает поиск в новых окрестностях, если нет, выбирается другое решение.

```
1: Используется раздельная кластеризация [19] для поиска начального
   распределения Initial_Alloc;
2: Best_Alloc = Initial_Alloc;
3: New_Alloc = Best_Alloc; iteration = 0;
4: Do
5:   searchstep = 0;  counter = 0;
6:   Do
7:     Выбрать случайно два узла из New_Alloc;
8:     Выбрать случайно два фрагмента данных из каждого узла;
9:     Поменять их местами;
10:    If стоимость уменьшилась Then
11:      принять изменения, counter = 0;
12:    Else
13:      отменить их, counter + +;
14:      EndIf
15:  Until + + searchstep > MAX_STEP || counter > MARGIN;
16:  If cost(New_Alloc) < cost(Best_Alloc) Then
17:    Best_Alloc = New_Alloc;
18:  EndIf
19:  Случайно обменять два фрагмента данных из двух отдельных
     случайных узлов из New_Alloc; /* Вероятностный прыжок*/
20: Until iteration > MAX_ITERATION.
```

Листинг 10. Алгоритм случайного поиска (Random Search (RS))
Listing 10. The random search algorithm

Алгоритм останавливается после определенного количества шагов или после фиксированного количества шагов, в ходе которых решение не улучшалось. Качество решения этой техники в значительной степени зависит от выбора и построения окрестностей.

Заключение

В данной обзорной статье рассмотрены подходы к решению задачи динамического оптимального размещения фрагментов данных распределенной базы данных, основанной на задаче глобальной минимизации времени выполнения запросов всех абонентов на всех узлах системы. Предложенные решения исключают полный перебор вариантов принятием дополнительных допущений, к которым относятся: оптимизация размещения только в установившихся режимах функционирования системы относительно частотного распределения запросов, ранжирование запросов по частоте и отсечение редко используемых, ранжирование запросов по времени исполнения и оптимизация самых медленных, выделение наиболее часто используемых и длительно выполняемых запросов.

Рассмотренные подходы реализуют различные технологические приемы к решению задачи оптимизации относительно двух критериев: времени доступа к данным и объема хранимых данных на каждом узле. К ним относятся задача минимизации суммарного времени доступа ко всем фрагментам, как в подходе с пороговой оценкой, концепция локальной оценки и минимизации путем выполнения распределенных «торгов» по технологии Mariposa, формирование иерархии кластеров для реализации распределения данных между кластерами в венгерском алгоритме, применение методов прогнозирования будущей нагрузки методами авторегрессии и скользящего среднего для распределения на основе прогнозирования, применение эвристики для оценки стоимости пути на полносвязном графе с выделением путей максимальной стоимости, автоматные подходы, состоящие в изменении состояния автомата для получения наиболее выгодного состояния системы, эволюционные алгоритмы, реализующие эффективные модификации генетических алгоритмов.

Таблица 5

Сравнительная характеристика подходов и алгоритмов к перераспределению данных в распределенных базах данных

Table 5

The comparison of approaches and algorithms for the redistribution of data in distributed databases

Подход	Ретроспектива	Прогноз	Анализируемые операции	Наличие централизованного брокера	Алгоритм оценки перераспределения
Пороговой оценки	Да, плавающим окном по времени	Нет	Выборка, добавление	Нет	Оригинальный
Распределенных торгов	Да, локальная на основе данных узла	Нет	Выборка добавление	Да, выборы победителя торгов	Оригинальный
Одноранговых сетей	Нет	Нет	Выборка	Да, требует иерархии брокетов	Венгерский
Прогнозирования состояния	Нет	Да, ARIMA	Выборка	Да, обеспечивает изменение числа узлов	Оригинальный
Избыточное распределение	Нет	Нет	Выборка	Да	Оригинальный
Автоматный подход	Да, на основе автоматов с памятью	Нет	Выборка, добавление	Да	Оригинальный
Эволюционный подход	Нет	Нет	Выборка	Да	Генетические алгоритмы: поиска, симулированной эволюции, среднего поля, случайного поиска

Результаты анализа подходов и применяемых в них алгоритмов приведены в табл. 5.

Предложенный обзор алгоритмов может использоваться для реализации эффективных методов распределения данных в системах управления распределенными базами данных.

Работа подготовлена в ходе реализации комплексного проекта в рамках Постановления Правительства РФ от 09.04.2010 № 218 при финансовой поддержке Министерства образования и науки РФ. Договор № 03.G25.31.0259 от 28.04.2017.

СПИСОК ЛИТЕРАТУРЫ

1. **Kumar R., Gupta N.** An extended approach to non-replicated dynamic fragment allocation in distributed database systems // IEEE Internat. Conf. on Issues and Challenges in Intelligent Computing Techniques. 2014. Pp. 861–865.

2. **Nilarun Mukharjee** Non-replicated dynamic fragment allocation in distributed database systems // CCSIT 20 II, Part I. Berlin Heidelberg: Springer-Verlag, 2011. CCIS 131. Pp. 560–569.

3. **Ceri S., Pelagatti G.** Distributed databases: principles systems. McGraw-Hili Internat. Editions, 1985.

4. **Horea-Adrian Grebla, Anca Gog** Redesign based optimization for distributed databases // Studia univ. Babe._bolyai, informatica. 2005. Vol. 1. No. 1.

5. **Agrawal S., Narasayya V., Yang B.** Integrating vertical and horizontal partitioning into automated physical database design // Proc. 2004 ACM

SIGMOD Internat. Conf. Management of Data. 2004. Pp. 359–370.

6. **Chu W.W.** Optimal file allocation in multiple computer systems // IEEE Transaction on Computers. 1969. Vol. C-18. No. IO.

7. **Ozsu M., Valduriez P.** Principles of distributed database systems. 2nd ed. Prentice Hall, 1999.

8. **Raju Kumar, Neena Gupta** Non-redundant dynamic data allocation in distributed database systems // Special Issue of Internat. J. of Computer Applications, on Issues and Challenges in Networking, Intelligence and Computing Technologies. 2012. Pp. 6–10.

9. **Singh A., Kahlon K.S.** Non-replicated dynamic data allocation in distributed database systems // Internat. J. of Computer Science and Network Security. 2009. Vol. 9. No. 9.

10. **Kumar R., Gupta N.** Dynamic data allocation in distributed database systems: A systematic survey // Internat. Review on Computers and Softwares. 2013. Vol. 8. No. 2. Pp. 660–667.

11. **Brunstroml A., Leutenegger S.T., Simhal R.** Experimental evaluation of dynamic data allocation strategies in a distributed database with changing workload // ACM Trans. Database Systems. 1995.

12. **Eswaran K.P.** Placement of records in a file and file allocation in a computer network // Proc. IFIP Congr. North-Holland, 1974.

13. **Ulus T., Uysal M.** Heuristic approach to dynamic data allocation in distributed database systems // Pakistan J. of Information and Technology 2003. No. 2(3). Pp. 231–239.

14. **Sarathy R., Shetty B., Sen A.** A constrained nonlinear 0-1 program for data allocation // European J. Operational Research. 1997. Vol. 102. Pp. 626–647.

15. **Hababeh I.O., Ramachandran M., Bowring N.** A high-performance computing method for data allocation in distributed database systems // J. Supercomput. Springer, 2007. No. 39. Pp. 3–18.

16. **Chiu G., Raghavendra C.** A model for optimal database allocation in distributed computing systems // Proc. IEEE INFOCOM 1990. Vol. 3. Pp. 827–833.

17. **Ram S., Narasimhan S.** Database allocation in a distributed environment: Incorporating a concurrency control mechanism and queuing costs // Management Science. 1994. Vol. 40. No. 8. Pp. 969–983.

18. **Huang Y., Chen J.** Fragment allocation in distributed database design // J. Information Science and Eng. 2001. Vol. 17. Pp. 491–506.

19. **Tamhankar A., Ram S.** Database fragmentation and allocation: A integrated methodology and case study // IEEE Trans. Systems, Man and Cybernetics. 1998. Part A. Vol. 28. No. 3.

20. **Wolfson O., Jajodia S., Huang Y.** An adaptive data replication algorithm // ACM Trans. Database Systems. 1997. Vol. 22. No. 2. Pp. 255–314.

21. **Zou L., Peng P.** A survey of distributed RDF data management // Jisuanji Yanjiu Yu Fazhan/Computer Research and Development. 2017. No. 54(6). Pp. 1213–1224.

22. **Eldawy A., Mokbel M.F.** The era of big spatial data // 31st IEEE Internat. Conf. on Data Engineering Workshops. Seoul, 2015. Pp. 42–49.

23. **Papastavrou Stavros, Chrysanthis Panos, Samaras George, Pitoura Evaggelia.** An evaluation of the JAVA-BASED approaches to web database access // Internat. J. of Cooperative Information Systems. 2000. No. 10.

24. **Jun-Lin Lin, Dunham M.H.** A survey of distributed database check pointing // Distrib. Parallel Databases. 1997. Vol. 5. No. 3. Pp. 289–319.

Статья поступила в редакцию 27.11.2018.

REFERENCES

1. **Kumar R., Gupta N.** An extended approach to non-replicated dynamic fragment allocation in distributed database systems. *IEEE International Conference on Issues and Challenges in Intelligent Computing Techniques*, 2014, Pp. 861–865.

2. **Nilarun Mukharjee** Non-replicated dynamic fragment allocation in distributed database systems. *CCSIT 20 II, Part I*, Berlin Heidelberg: Springer-Verlag, 2011, CCIS 131, Pp. 560–569.

3. **Ceri S., Pelagatti G.** *Distributed databases: principles systems.* McGraw-Hili International Editions, 1985.

4. **Horea-Adrian Grebla, Anca Gog** Redesign based optimization for distributed databases. *Studia univ. Babe._bolyai, informatica.* 2005, Vol. 1, No. 1.

5. **Agrawal S., Narasayya V., Yang B.** Integrating vertical and horizontal partitioning into automated physical database design. *Proc. 2004 ACM SIGMOD International Conf. Management of Data*, 2004, Pp. 359–370.

6. **Chu W.W.** Optimal file allocation in multiple computer systems. *IEEE Transaction on Computers*, 1969, Vol. C-18, No. IO.

7. **Ozsu M., Valduriez P.** *Principles of distributed database systems.* 2nd ed. Prentice Hall, 1999.

8. **Raju Kumar, Neena Gupta** Non-redundant dynamic data allocation in distributed database sys-

tems. *Special Issue of International Journal of Computer Applications, on Issues and Challenges in Networking, Intelligence and Computing Technologies*, Nov. 2012, Pp. 6–10.

9. **Singh A., Kahlon K.S.** Non-replicated dynamic data allocation in distributed database systems. *International Journal of Computer Science and Network Security*, 2009, Vol. 9, No. 9.

10. **Kumar R., Gupta N.** Dynamic data allocation in distributed database systems: A systematic survey. *International Review on Computers and Softwares*, 2013, Vol. 8, No. 2, Pp. 660–667.

11. **Brunstroml A., Leutenegger S.T., Simhal R.** Experimental evaluation of dynamic data allocation strategies in a distributed database with changing workload. *ACM Trans. Database Systems*, 1995.

12. **Eswaran K.P.** Placement of records in a file and file allocation in a computer network. *Proc. IFIP Congr. North-Holland*, 1974.

13. **Ulus T., Uysal M.** Heuristic approach to dynamic data allocation in distributed database systems. *Pakistan Journal of Information and Technology*, 2003, No. 2(3), Pp. 231–239.

14. **Sarathy R., Shetty B., Sen A.** A constrained nonlinear 0-1 program for data allocation. *European J. Operational Research*, 1997, Vol. 102, Pp. 626–647.

15. **Hababeh I.O., Ramachandran M., Bowring N.** A high-performance computing method for data allocation in distributed database systems. *J. Supercomput.* Springer, 2007, No. 39. Pp. 3–18.

16. **Chiu G., Raghavendra C.** A model for optimal database allocation in distributed computing systems. *Proc. IEEE INFOCOM 1990*, Vol. 3, Pp. 827–833.

17. **Ram S., Narasimhan S.** Database allocation in a distributed environment: Incorporating a concurrency control mechanism and queuing costs. *Management Science*, 1994, Vol. 40, No. 8, Pp. 969–983.

18. **Huang Y., Chen J.** Fragment allocation in distributed database design. *J. Information Science and Eng.*, 2001, Vol. 17, Pp. 491–506.

19. **Tamhankar A., Ram S.** Database fragmentation and allocation: A integrated methodology and case study. *IEEE Trans. Systems, Man and Cybernetics*, 1998, Part A, Vol. 28, No. 3.

20. **Wolfson O., Jajodia S., Huang Y.** An adaptive data replication algorithm. *ACM Trans. Database Systems*, 1997, Vol. 22, No. 2, Pp. 255–314.

21. **Zou L., Peng P.** A survey of distributed RDF data management. *Jisuanji Yanjiu Yu Fazhan/Computer Research and Development*, 2017, No. 54(6), Pp. 1213–1224.

22. **Eldawy A., Mokbel M.F.** The era of big spatial data. *31st IEEE International Conference on Data Engineering Workshops*, Seoul, 2015, Pp. 42–49.

23. **Papastavrou Stavros, Chrysanthis Panos, Samaras George, Pitoura Evaggelia.** An evaluation of the JAVA-BASED approaches to web database access. *International Journal of Cooperative Information Systems*, 2000, No. 10.

24. **Jun-Lin Lin, Dunham M.H.** A survey of distributed database check pointing. *Distrib. Parallel Databases,* 1997, Vol. 5, No. 3, Pp. 289–319.

Received 27.11.2018.

СВЕДЕНИЯ ОБ АВТОРАХ / THE AUTHORS

ПОПОВ Сергей Геннадьевич
POPOV Sergey G.
E-mail: popovserge@spbstu.ru

ФРИДМАН Виктор Сергеевич
FRIDMAN Viktor S.
E-mail: esselllesse@gmail.