РОССИЙСКАЯ ФЕДЕРАЦИЯ

(19) **RU** (11) **2 628 208**(13) **C2**

(51) МПК
G06F 15/16 (2006.01)

ФЕДЕРАЛЬНАЯ СЛУЖБА
ПО ИНТЕЛЛЕКТУАЛЬНОЙ СОБСТВЕННОСТИ

(12) ОПИСАНИЕ ИЗОБРЕТЕНИЯ К ПАТЕНТУ

(21)(22) Заявка: **2014126065, 19.12.2012**

(24) Дата начала отсчета срока действия патента:
19.12.2012

Дата регистрации:
15.08.2017

Приоритет(ы):
(30) Конвенционный приоритет:
27.12.2011 US 13/337,291

(43) Дата публикации заявки: **27.01.2016** Бюл. № **3**

(45) Опубликовано: **15.08.2017** Бюл. № **23**

(85) Дата начала рассмотрения заявки PCT на национальной фазе: **26.06.2014**

(86) Заявка PCT:
US 2012/070427 (19.12.2012)

(87) Публикация заявки PCT:
WO 2013/101563 (04.07.2013)

Адрес для переписки:
129090, Москва, ул. Б. Спасская, 25, строение 3, ООО "Юридическая фирма Городисский и Партнеры"

(72) Автор(ы):
ЧАНДРАМОУЛИ Бадриш (US),
НАТХ Суман К. (US),
ЧЖОУ Вэньчао (US)

(73) Патентообладатель(и):
МАЙКРОСОФТ ТЕКНОЛОДЖИ ЛАЙСЕНСИНГ, ЭлЭлСи (US)

(56) Список документов, цитированных в отчете о поиске: **US 2011/126168 A1, 26.05.2011. US 2011/072489 A1, 24.03.2011. US 6,332,163 B1, 18.12.2001. US 2010/332818 A1, 30.12.2010. RU 2 435 236 C1, 27.11.2011.**

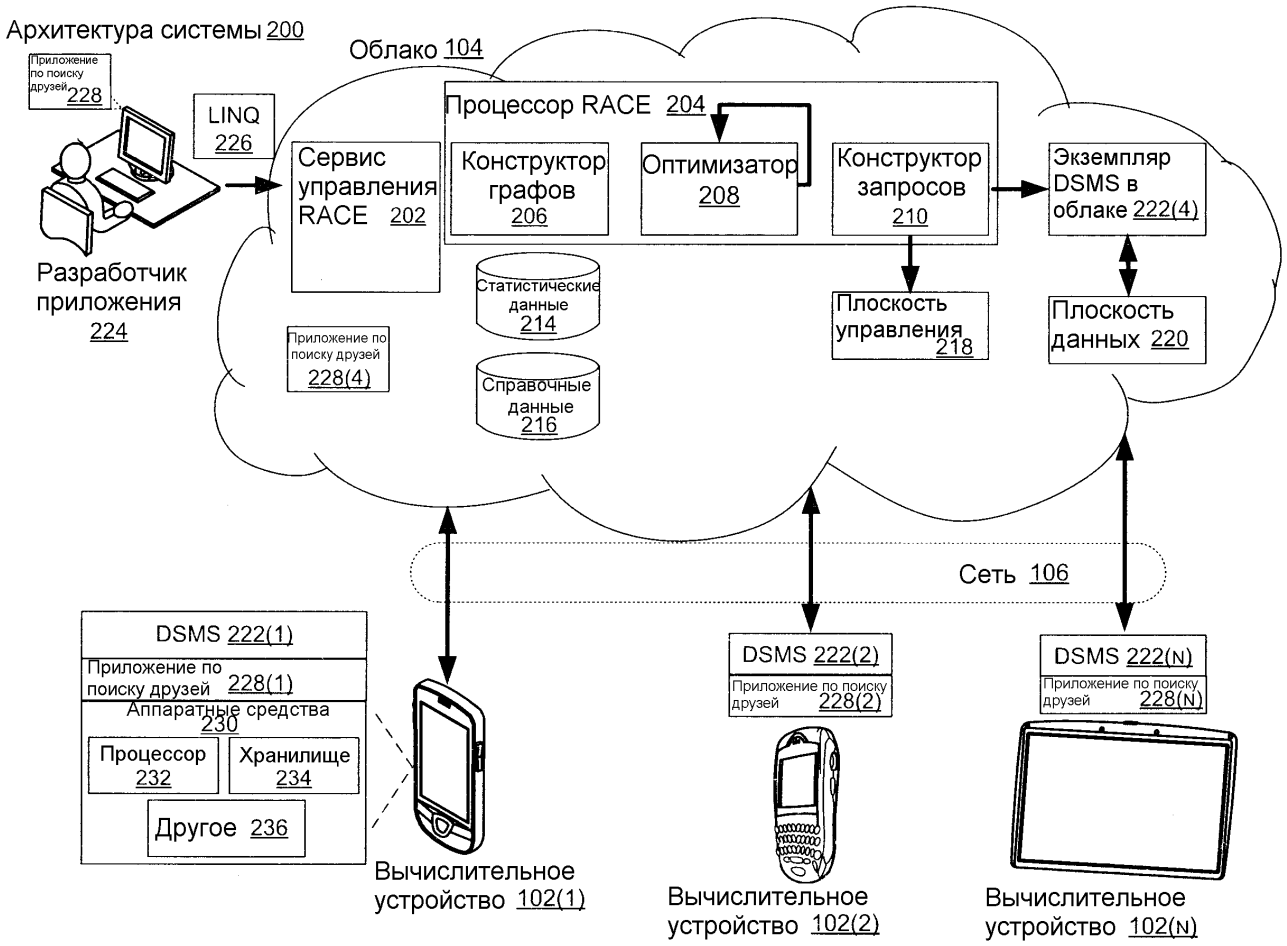
(54) ОБЛАЧНО-ГРАНИЧНЫЕ ТОПОЛОГИИ

(57) Реферат:

Изобретение относится к области облачно-граничной топологии сети. Техническим результатом является сокращение использования ресурсов системы. Машиночитаемый носитель информации, на котором сохранены машиноисполняемые инструкции, которые при их исполнении вычислительным устройством предписывают вычислительному устройству выполнять действия: оценивание запроса потоковой передачи данных реального времени, сравнение использования ресурсов между первым сценарием, который задействует выгрузку данных запросов, ассоциированных с упомянутым запросом потоковой передачи данных реального времени, из упомянутого множества различных граничных вычислительных устройств в облачные ресурсы для обработки, и вторым сценарием, который задействует выгрузку данных запросов из всех, кроме одного из упомянутого множества, различных граничных вычислительных устройств в облачные ресурсы и скачивание данных запросов в подмножество этого множества различных граничных вычислительных устройств для обработки. 9 з.п. ф-лы, 1 табл., 10 ил.

RU 2628208 C2

RU 2628208 C2



ФИГ.2

RU 2628208 C2

RU 2628208 C2

RUSSIAN FEDERATION

FEDERAL SERVICE
FOR INTELLECTUAL PROPERTY

(19) **RU** (11) **2 628 208**(13) **C2**

(51) Int. Cl.
G06F 15/16 (2006.01)

(12) ABSTRACT OF INVENTION

(21)(22) Application: **2014126065, 19.12.2012**

(24) Effective date for property rights:
19.12.2012

Registration date:
15.08.2017

Priority:
(30) Convention priority:
27.12.2011 US 13/337,291

(43) Application published: **27.01.2016** Bull. № **3**

(45) Date of publication: **15.08.2017** Bull. № **23**

(85) Commencement of national phase: **26.06.2014**

(86) PCT application:
US 2012/070427 (19.12.2012)

(87) PCT publication:
WO 2013/101563 (04.07.2013)

Mail address:
129090, Moskva, ul. B. Spasskaya, 25, stroenie 3, OOO "Yuridicheskaya firma Gorodisskij i Partnery"

(72) Inventor(s):
CHANDRAMOULI Badrish (US),
NATKH Suman K. (US),
CHZHOU Venchao (US)

(73) Proprietor(s):
MAJKROSOFT TEKNOLODZHI LAJSENSING, ElElSi (US)

RU 2628208 C2

(54) CLOUD-BORDER TOPOLOGIES

(57) Abstract:

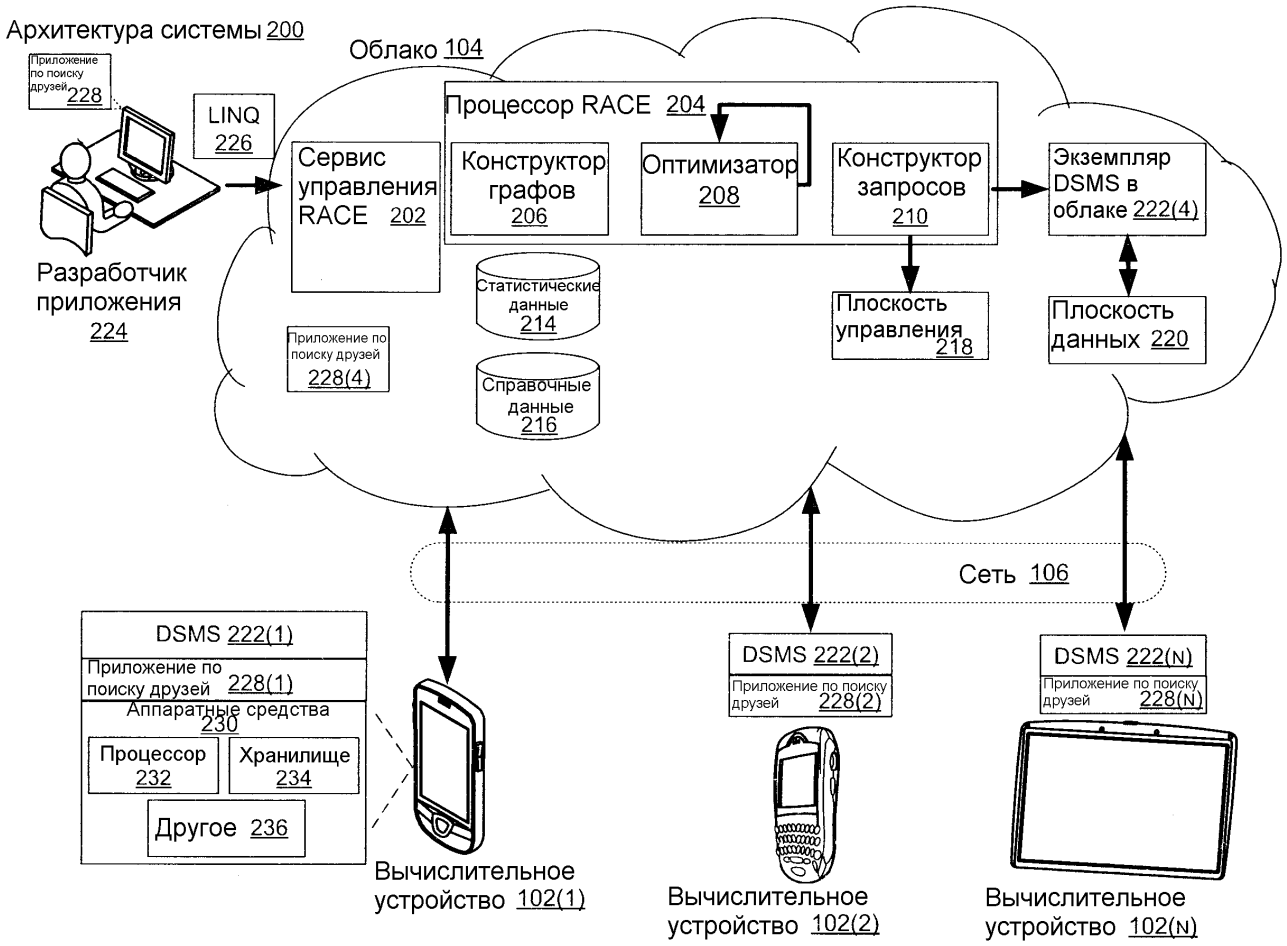
FIELD: physics.

SUBSTANCE: computer-readable medium storing computer-executable instructions that, when executed by a computing device, instructs the computing device to perform actions: estimating a real-time streaming request, comparing resource utilisation between the first scenario that involves downloading query data associated with mentioned request of streaming transmitting of real-time data from mentioned plurality of different edge computing devices to an area of the second scenario, which involves downloading query data from all but one of the plurality, different boundary computing devices to cloud resources, and downloading query data into a subset of this set of different boundary computing devices for processing.

EFFECT: reducing the use of system resources.

10 cl, 1 tbl, 10 dwg

RU 2628208 C2



ФИГ.2

RU 2628208 C2

RU 2628208 C2

ОБЛАСТЬ ТЕХНИКИ, К КОТОРОЙ ОТНОСИТСЯ ИЗОБРЕТЕНИЕ

Широко распространенное применение «интеллектуальных» портативных вычислительных устройств, таких как, например, смартфоны, потребителями и доступность большого количества облачных вычислительных ресурсов привели к так называемой «облачно-граничной (Cloud-Edge) топологии». Эти интеллектуальные портативные вычислительные устройства называют «интеллектуальными» потому, что улучшенные процессор и память позволяют этим устройствам иметь значительные вычислительные ресурсы, доступные пользователю. Интеллектуальные портативные вычислительные устройства могут формировать данные в режиме реального времени, как, например, местоположение GPS, расход батареи, скорость и так далее. Эти интеллектуальные портативные вычислительные устройства могут также рассматриваться как облачно-граничные устройства, так как связь между отдельным устройством и облачными ресурсами может рассматриваться как через границу.

С учетом значительных вычислительных ресурсов, доступных на интеллектуальных портативных вычислительных устройствах, пользователь может выбирать различные приложения для выполнения на своем устройстве. Многие из этих приложений могут быть обозначены как облачно-граничные приложения, так как один экземпляр приложения функционирует на интеллектуальном портативном вычислительном устройстве, а другой экземпляр приложения функционирует на облачных вычислительных ресурсах. Существует широкий класс облачно-граничных приложений, которые коррелируют данные по множеству интеллектуальных портативных вычислительных устройств и облаку, чтобы достигнуть функциональности приложения. Примером является приложение по поиску друзей, которое функционирует, чтобы уведомить пользователя, есть ли поблизости друзья. Функциональность этого приложения зависит от корреляции местоположений в режиме реального времени и медленно меняющихся данных, как, например, социальные сети. В то время как огромное количество вычислительных ресурсов доступно на интеллектуальных портативных вычислительных устройствах и облачных ресурсах, использование ресурсов, как, например, ресурсов связи, может быть значительным, когда огромные количества интеллектуальных портативных вычислительных устройств исполняют облачно-граничные приложения.

РАСКРЫТИЕ ИЗОБРЕТЕНИЯ

Описание относится к облачно-граничным топологиям. Некоторые аспекты относятся к облачно-граничным приложениям и использованию ресурсов в различных облачно-граничных топологиях. Согласно одному примеру, можно оценивать запрос потоковой передачи данных реального времени, который использует данные из множества различных граничных вычислительных устройств. Это множество различных граничных вычислительных устройств может быть выполнено с возможностью связываться с облачными ресурсами, но не связываться напрямую друг с другом. Отдельные граничные вычислительные устройства включают в себя экземпляр приложения, выраженного в описательном временном языке. Согласно этому примеру, можно сравнивать использование ресурсов между первым и вторым сценарием. Первый сценарий включает в себя выгрузку данных запросов из множества различных граничных вычислительных устройств в облачные ресурсы для обработки. Второй сценарий включает в себя выгрузку данных запросов из всех кроме одного узла из множества различных граничных вычислительных устройств в облачные ресурсы и скачивание данных запросов в один узел из множества различных граничных вычислительных устройств для обработки.

Другой аспект настоящих облачно-граничных топологий может относиться к спецификации облачно-граничных приложений с использованием временного языка. Дополнительный аспект может включать в себя архитектуру, которая выполняет машины систем управления потоками данных (DSMS) в облаке и облачно-граничных компьютерах для выполнения частей запроса.

Вышеперечисленные примеры предназначены для обеспечения быстрой ссылки для помощи читателю и не предназначены для определения объема концепций, описанных в настоящей заявке.

КРАТКОЕ ОПИСАНИЕ ЧЕРТЕЖЕЙ

Сопровождающие чертежи изображают варианты реализации идей, выраженных в настоящей заявке. Признаки изображенных вариантов реализации могут быть более понятны с помощью ссылки на нижеследующее описание, рассмотренное совместно с сопровождающими чертежами. Одинаковые ссылочные позиции в различных чертежах используются, где это возможно, для указания одинаковых элементов. Дополнительно, крайнее левое число каждой ссылочной позиции выражает фигуру и связанное с ней обсуждение, где в первый раз вводится ссылочная позиция.

Фиг. 1 изображает пример системы, в которой настоящие идеи использования ресурсов облачно-граничного приложения могут быть применены в соответствии с некоторыми вариантами осуществления.

Фиг. 2 изображает пример архитектуры системы, в которой настоящие идеи использования ресурсов облачно-граничного приложения могут быть применены в соответствии с некоторыми вариантами осуществления.

Фиг. 3-9 изображают примеры графов, в которых настоящие идеи использования ресурсов облачно-граничного приложения могут быть применены в соответствии с некоторыми вариантами осуществления.

Фиг. 10 изображает логическую блок-схему примера способа использования ресурсов облачно-граничного приложения в соответствии с некоторыми вариантами осуществления концепций настоящего изобретения.

ОСУЩЕСТВЛЕНИЕ ИЗОБРЕТЕНИЯ

ВВЕДЕНИЕ

Концепции настоящего изобретения относятся к облачным системам и динамической, с учетом ресурсов, обработке приложениями, исполняемыми в облачных системах и подсоединенных устройствах.

С целью объяснения рассмотрим вводную фиг. 1, которая изображает пример системы 100, которая может осуществлять концепции настоящего изобретения. Система 100 включает в себя три облачно-граничных вычислительных устройства (здесь и далее «вычислительные устройства») 102(1), 102(2) и 102(N) (где N обозначает, что любое число вычислительных устройств может быть использовано). Вычислительные устройства 102(1)-102(N) могут связываться с облаком 104 по сети 106, как указано линиями 108(1)-108(3), соответственно. В этом примере отдельные вычислительные устройства могут связываться друг с другом по облаку 104, но не напрямую с другими вычислительными устройствами. Облако 104 может предложить большое количество вычислительных ресурсов 110, хотя точное физическое местоположение этих вычислительных ресурсов может быть не очевидным. Облачное вычисление продолжает набирать популярность вследствие относительно дешевых и многочисленных вычислительных ресурсов, которое оно предлагает.

Вычислительные устройства 102(1)-102(N) могут быть любым типом вычислительных устройств. В общем, эти вычислительные устройства являются портативными

вычислительными устройствами, как, например, смартфоны и планшетные компьютеры. Термин «компьютер» или «вычислительное устройство», как использовано в настоящем документе, может обозначать любой тип устройства, которое имеет некоторую способность обработки данных. В то время как конкретные примеры таких устройств изображены с целью объяснения, другие примеры таких устройств могут включать в себя традиционные вычислительные устройства, как, например, персональные компьютеры, сотовые телефоны, смартфоны, персональные цифровые помощники (PDA) или любые из огромного числа когда-либо разработанных или находящихся в разработке типов устройств. Дополнительно, компьютер может быть встроен в другое устройство, а не быть автономным. Например, настольный компьютер может быть включен в автомобиль или другое средство передвижения.

При рассмотрении с одной перспективы, вычислительные устройства 102(1)-102(N) могут быть рассмотрены как «граничные устройства» в топологии, поддерживаемой облаком 104 и сетью 106. Многие из этих граничных устройств оборудованы сенсорами, которые производят частые или непрерывные потоки данных в реальном времени, как, например, местоположение GPS пользователя, скорость, текущая активность, расход батареи устройства и так далее. Дополнительно, может быть растущее количество медленно изменяемых справочных данных, как, например, графы социальной сети и цены на топливо на автозаправочных станциях, которые становятся доступными в облаке, например, по рынкам данных. Этот быстрый рост вычислительных устройств и данных подкрепил растущий интерес к развивающемуся классу облачно-граничных приложений в реальном времени (или облачно-граничных приложений (APP) для краткости). Эти облачно-граничные приложения могут обеспечивать сервисы, как, например, уведомления или рекомендации, на основании веб-каналов в реальном времени, собранных из большого числа граничных вычислительных устройств и облака.

В некоторых сценариях вычислительные устройства 102(1)-102(N) передают свои данные в облако 104 для обработки с помощью одного или нескольких провайдеров сервисов, выполняемых на облачных вычислительных ресурсах 110. Например, предположим с целью объяснения, что один такой сервис является сервисом по поиску друзей, который уведомляет пользователя, есть ли рядом с ее текущим местоположением любой из ее друзей. В некоторых вариантах осуществления сервис по поиску друзей может осуществляться приложением по поиску друзей, исполняемым на облачных вычислительных ресурсах 110, и соответствующими приложениями по поиску друзей, исполняемыми на отдельных вычислительных устройствах 102(1)-102(N).

Активация приложения по поиску друзей влечет за собой корреляцию местоположений в реальном времени от смартфонов пользователей (например, вычислительных устройств 102(1)-102(N)), а также медленно изменяемых справочных данных, таких как, например, социальная сеть (определение дружеских взаимоотношений). Для облегчения объяснения рассмотрим только вычислительные устройства 102(1) и 102(2) и предположим, что вычислительное устройство 102(1) принадлежит Пользователю 1 и что вычислительное устройство 102(2) принадлежит Пользователю 2. Дополнительно, предположим, что Пользователь 1 и Пользователь 2 были определены как «друзья». Каждое вычислительное устройство может время от времени выгружать данные в облако, как указано стрелками 112(1) и 112(2). Выгруженные данные могут быть обработаны провайдером сервисов, функционирующим на облачных вычислительных ресурсах 110. Провайдер сервисов может определять результаты для каждого вычислительного устройства и передавать эти результаты обратно соответственным вычислительным устройствам 102(1) и 102(2). В некоторых случаях такой процесс может повлечь за

собой большие количества осуществлений связи по выгрузке и скачиванию по сети 106 между облаком 104 и вычислительными устройствами 102(1) и 102(2). Концепции настоящего изобретения могут позволить альтернативный вариант. Этот альтернативный вариант можно рассматривать как динамический вариант с учетом ресурсов. В динамическом варианте с учетом ресурсов одно из вычислительных устройств 102(1) и 102(2) может определять, что использование ресурсов системы, как, например, этих сетевых передач, может быть сокращено путем получения отдельным вычислительным устройством данных другого вычислительного устройства из облака и локальной обработки на отдельном вычислительном устройстве. (Сетевые передачи могут учитываться по их количеству и/или использованию полосы пропускания сети). В таком случае отдельное вычислительное устройство не выполняет выгрузку. Другие (оставшиеся) вычислительные устройства выполняют выгрузку как обычно, и отдельное вычислительное устройство выполняет скачивание. Этот динамический вариант с учетом ресурсов можно рассматривать как динамический, потому что вычисления использования ресурсов могут меняться по мере изменения сценария. Один такой пример описан ниже по отношению к частоте, с которой вычислительное устройство формирует данные местоположения. Вычисления использования ресурсов могут производить другой результат, когда частота формирования данных местоположения меняется. Таким образом, определение, вместо того чтобы быть единовременным, может повторяться альтернативным образом по мере изменения условий или параметров.

Для иллюстрирования этого сокращенного использования ресурсов предположим, что вычислительное устройство 102(1) принадлежит Пользователю 1, а вычислительное устройство 102(2) принадлежит Пользователю 2. Дополнительно предположим, что Пользователь 1 работает в своем офисе (например, относительно стационарном), а Пользователь 2 ведет машину по соседству. В вышеописанной фиксированной конфигурации существующее приложение по поиску друзей попросит Пользователя 2 (вычислительное устройство 102(2)) выгружать (112(2)) его/ее местоположение часто (например, каждые 10 секунд), чтобы облако знало о его/ее обновлении местоположения для корреляции с местоположением Пользователя 1. Пользователь 1 (вычислительное устройство 102(1)), однако, может выгружать (112(1)) его/ее местоположение нечасто (например, раз в час), так как он/она не передвигается много. В этом примере общая нагрузка на связь Пользователя 1 и Пользователя 2 составляет 361 сообщение в час (с игнорированием финальных сообщений уведомления) по сети 106. Использование сети может быть дорогостоящим, особенно когда пользователь имеет много друзей или запускает много подобных приложений. Это может серьезно ограничить применимость приложения, так как приводит к ограничению того, как часто коррелируются данные пользователей, что приводит к большому времени задержки уведомлений. Более того, пользователи могут просто отключить приложение вследствие высокого использования им ресурсов. Однако эту неэффективность можно легко решить в вышеуказанном примере с помощью динамического варианта с учетом ресурсов. Вместо использования методологии корреляции в облаке, местоположение Пользователя 1 может быть отправлено вычислительному устройству 102(2) Пользователя 2 (через облако 104, как указано стрелками 114 и 116, соответственно). Корреляция может затем быть выполнена вычислительным устройством Пользователя 2. В этом случае Пользователю 2 не нужно отправлять его/ее местоположение, и общая нагрузка станет только 2 сообщения в час (одно - от Пользователя 1 в облако, и другое - из облака Пользователю 2). Необходимо отметить, что в последующую точку времени, как, например, когда Пользователь 1

едет домой, динамический вариант с учетом ресурсов может определять другой подход, такой как, например, обработка в облаке 104.

Подводя итог, динамический вариант с учетом ресурсов может определять, какое (если оно есть) вычисление принудительно направить и какому граничному вычислительному устройству его принудительно направить. Это определение можно рассматривать как проблему оптимизации, которая зависит от различных факторов, таких как, например, топология сети, скорости передачи потоков данных, затраты на выгрузку и скачивание данных, пары потоков для корреляции и так далее. Более того, так как эти параметры могут меняться с течением времени (например, частота передачи данных от Пользователя 1 может меняться, когда он/она начинает перемещение после рабочего дня), то данное определение может динамически обновляться. Одна реализация динамического варианта с учетом ресурсов называется RACE и описана подробно ниже.

Кратко, RACE (приложения реального времени через границу облака) является структурой и системой спецификации и эффективного исполнения облачно-граничных приложений. RACE может использовать технологии баз данных для решения вопросов, касающихся двух главных аспектов. Во-первых, RACE решает вопрос, касающийся спецификации облачно-граничных приложений реального времени. Во-вторых, RACE решает вопрос, касающийся использования ресурсов системы, связанных с исполнением облачно-граничных приложений реального времени. Использование ресурсов системы может быть улучшено и/или оптимизировано (здесь и далее, для краткости, термин «оптимизированный» означает «улучшенный и/или оптимизированный»).

СПЕЦИФИКАЦИЯ ОБЛАЧНО-ГРАНИЧНЫХ ПРИЛОЖЕНИЙ

RACE решает вопрос, касающийся спецификации облачно-граничных приложений реального времени, путем абстрагирования базовой логики облачно-граничных приложений как независимых от платформы непрерывных запросов (CQ) по набору ресурсов потоковых данных.

Облачные граничные приложения часто написаны на стандартных процедурных языках, таких как, например, Objective C, Java или C#. От разработчиков приложений требуется вручную реализовать механизмы, которые обрабатывают связи между устройствами, публикацию и подписку на потоки данных и связанную со временем семантику в логике приложений, как, например, временные присоединения и оконные вычисления. Этот процесс отнимает много времени и подвержен ошибкам. RACE может добавить поддержку платформы для общего функционала, совместно используемого большинством облачно-граничных приложений. Разработчики приложений могут затем сфокусироваться на базовой логике приложений, а не на деталях реализации.

Настоящие варианты реализации используют тот факт, что, хотя различные облачно-граничные приложения имеют разнообразные характеристики, зависящие от приложений (например, визуализацию и поддержку личной информации), они могут иметь общие черты. Например, данные и базовая логика приложений для облачно-граничных приложений часто являются временными по своей природе. Другими словами, облачно-граничные приложения можно рассматривать как непрерывные запросы, которые непрерывно коррелируют данные в реальном времени и медленно меняющиеся (но по-прежнему временные) справочные данные в высокой степени распределенной системе.

Например, приложение по поиску друзей можно рассматривать как временное присоединение (связывание) между местоположениями GPS в реальном времени граничных устройств и более медленно меняющимся потоком социальной сети. Приложение купонов, учитывающее местоположение, коррелирует информацию о

местоположении пользователя с профилями пользователей (вычисленными в историческом окне времени) и текущими рекламными объявлениями. Таким образом, в некоторых вариантах осуществления язык спецификации для облачно-граничных приложений должен содержать родную поддержку временной семантики. Такая поддержка способствует ясному выражению временно-ориентированных операций, как, например, временные присоединения и оконные объединения. Альтернативно или дополнительно, язык может иметь другие свойства. Например, одним таким свойством является описательная природа языка спецификации. Это может позволить разработчикам приложений определять приложения описательным и независимым от топологии сети образом, где они могут сфокусироваться на том, «какие» приложения, а не «как» они реализуются. Детали осуществления могут быть прозрачными для разработчиков приложений и быть обработанными автоматически с помощью базовой платформы. Другое свойство может относиться к краткости. Спецификация приложений может быть краткой, что позволяет продуктивную разработку прототипов, подготовку к эксплуатации и отладку для разработчиков приложений. Краткость естественным образом связана с внедрением описательных спецификаций. Гибкость может быть другим свойством. Язык спецификации может быть гибким, чтобы разработчики приложений могли легко настраивать приложение в соответствии с различными входящими/выходящими источниками и конфигурациями.

Пространство разработки для языков спецификации описано сейчас в свете этих свойств. Описательные языки, такие как, например, SQL и Datalog (и его варианты, например, Network Datalog), могут обеспечить краткое и гибкое описание непрерывных запросов в распределенных окружениях. Однако эти языки не имеют родной поддержки временной семантики, что может быть важным для большинства облачно-граничных приложений. С другой стороны, системы управления потоками данных (DSMS) используют описательные временные языки, которые удовлетворяют желаемым свойствам. Примеры включают в себя LINQ™ для StreamInsight™, StreamSQL™ для Oracle® CEP и StreamBase™. Описание ниже использует LINQ для StreamInsight в качестве языка спецификации, но он применим и к другим конфигурациям. LINQ обеспечивает описательную спецификацию временных запросов и основан на хорошо определенной алгебре и семантике, которые соответствуют временной природе облачно-граничных приложений.

В следующем обсуждении приводится пример спецификации облачно-граничных приложений. Вспомним, что запрос по поиску друзей ищет все пары пользователей (Пользователь 1, Пользователь 2), которые удовлетворяют условиям: 1) Пользователь 2 является другом Пользователя 1; и 2) два пользователя географически близки друг к другу в данное время. Здесь, с целью объяснения, предположим, что дружеские отношения асимметричны, то есть тот факт, что Пользователь 2 является другом Пользователя 1, необязательно подразумевает обратное, в данный момент времени. Существует два ввода в приложение по поиску друзей, а именно потоки местоположения GPS, направленные граничными устройствами, и данные социальной сети. Местоположения GPS активно собираются во время исполнения, в то время как данные социальной сети относительно медленно меняются и, в общем, доступны в облаке. Приложение по поиску друзей может быть написано в качестве двухступенчатого временного запроса на присоединение, как изображено ниже.

```
var query0 = from e1 in location
from e2 in socialNetwork
where e1.UserId==e2.UserId
```

```
select new {e1.UserId, e1.Latitude,
e1.Longitude, e2.FriendId};
var query1 = from e1 in query0
from e2 in location
where e1.FriendId == e2.UserId &&
Distance(e1.Latitude, e1.Longitude,
e2.Latitude, e2.Longitude) < THRESHOLD
select new {User1 = e1.UserId, User2 = e2.UserId}
```

Первый запрос (query0) присоединяет поток (location) местоположения GPS к справочному потоку социальной сети (socialNetwork), и получившийся выходной поток присоединяется опять к местоположениям GPS (в query1) для проверки расстояния между каждой парой друзей. Конечный вывод - поток пар (Пользователь 1, Пользователь 2), где два пользователя являются друзьями и географически близки друг к другу.

Вышеуказанная спецификация запроса определяет высокоуровневую логику запроса в качестве временных присоединений и ссылается на схемы потока location и потока socialNetwork. Это записано в потоке социальной сети и концептуально объединено с вводом потока местоположения GPS, и, таким образом, является независимым от топологии сети. В качестве другого примера, предположим, что желаемая функция - найти друзей, которые посещали конкретное местоположение (например, ресторан) на прошлой неделе. Чтобы это определить, концепции настоящего изобретения могут позволить заменить ввод местоположения в query1 на location.AlterEventDuration (TimeSpan.FromDays(7)). Это расширяет «время существования» событий в местоположении до 7 дней, что позволяет при присоединении учесть события друзей за прошлую неделю.

Подводя итог, RACE может использовать описательную спецификацию облачно-граничного приложения. RACE может выполнять логику на распределенной системе, состоящей из граничных устройств и облака. RACE может использовать немодифицированную DSMS в качестве черного ящика для локального выполнения запросов на отдельных граничных устройствах и в облаке. Некоторые реализации RACE могут функционировать, основываясь на предположении о том, что DSMS обеспечивает пользователям прикладной программный интерфейс (API) управления для подачи запросов (что определяет непрерывные запросы для выполнения), типов событий (что определяет схему входящих потоков данных) и адаптеров ввода и вывода (что определяет то, как потоковые данные достигают DSMS из внешнего мира и наоборот). Дополнительно, API также позволяет пользователям начинать и прекращать запросы DSMS.

Говоря другими словами, некоторые реализации могут передвинуть различные потоки данных (или части потоков) в облако и/или на другие граничные вычислительные устройства через облако. Некоторые другие потоки данных могут остаться локально в устройстве и не выгружаться в облако. Дополнительно, эти разнообразные (перемещенные и локальные) потоки могут служить в качестве вводов в сегменты запросов приложений, выполняемых в разнообразных местоположениях (как, например, подгруппа устройств или даже облако). Выходные потоки таких запросов сами могут либо остаться локально для дополнительных вычислений, либо выгружаться в облако (и затем, возможно, быть направленными на другие устройства). В общем, вычисление, определенное конечным пользователем, может быть выполнено распределенным образом.

АРХИТЕКТУРА RACE

Фиг. 2 изображает общую систему или архитектуру 200 системы одного варианта осуществления RACE. Архитектура 200 системы распространяется на вычислительные устройства 102(1)-102(N), облако 104 и сеть 106 на фиг. 1. Архитектура 200 системы вводит сервис 202 управления RACE и процессор 204 RACE. Процессор RACE включает в себя конструктор 206 графов, оптимизатор 208 и конструктор 210 запросов. Архитектура 200 системы также включает в себя статистические данные 214, справочные данные 216, плоскость 218 управления и плоскость 220 данных. Вычислительные устройства 102(1)-102(N) включают в себя экземпляр 222(1)-222(3) DSMS, соответственно. Экземпляр 222(4) DSMS также есть в облаке 104.

Архитектура 200 системы объяснена относительно опыта, обеспечиваемого разработчику 224 приложений. Разработчик приложений может взаимодействовать с сервисом 202 управления RACE путем написания приложения на описательном и временном языке, таком как, например, LINQ 226. Предположим с целью объяснения, что приложение является приложением 228 по поиску друзей. Функциональность приложений по поиску друзей была введена выше по отношению к фиг. 1. Приложение 228 по поиску друзей может быть манифестом на отдельных вычислительных устройствах 102(1)-102(N) в качестве экземпляров 228(1)-228(3), соответственно, и в облаке 104 в качестве экземпляров 228(4) приложений по поиску друзей. Дополнительно, хотя это изображено только по отношению к вычислительному устройству 102(1) для краткости, отдельные вычислительные устройства могут включать в себя разнообразные аппаратные средства 230. В этом примере изображенные аппаратные средства - это процессор 232, хранилище 234 данных и другие 236. Вышеуказанные элементы описаны более подробно ниже.

Процессор 232 может выполнять данные в форме машиночитаемых команд для обеспечения функциональности, такие как, например, функциональности приложения по поиску друзей. Данные, как, например, машиночитаемые команды, могут храниться в хранилище 234 данных. Хранилище данных может быть внутренним или внешним по отношению к вычислительному устройству. Хранилище 234 данных может включать в себя одно или несколько энергозависимых или энергонезависимых запоминающих устройств, жестких дисков и/или оптических устройств хранения (например, CD, DVD и так далее), среди прочих.

Компьютер 102(1) также может быть выполнен с возможностью принимать и/или формировать данные в форме машиночитаемых команд от хранилища 234 данных. Компьютер 102(1) может также принимать данные в форме машиночитаемых команд по сети 106, которые затем сохраняются на компьютере для выполнения процессором.

В альтернативной конфигурации, компьютер 102(1) может быть осуществлен в качестве типа дизайна системы на кристалле (SOC). В таком случае функциональность, обеспечиваемая компьютером, может быть интегрирована на единой SOC или множестве соединенных SOC. В некоторых конфигурациях вычислительные устройства могут включать в себя общие ресурсы и закрепленные ресурсы. Интерфейс(ы) усиливает связь между общими ресурсами и закрепленными ресурсами. Как подразумевает название, закрепленные ресурсы можно рассматривать как включающие отдельные части, которые закреплены для достижения конкретной функциональности. Общие ресурсы могут быть хранилищем, обрабатывающими блоками и так далее, которые могут быть использованы множеством функциональностей.

В общем, любая из функций, описанных в настоящей заявке, может быть осуществлена с использованием программного обеспечения, программно-аппаратных средств

(firmware), аппаратных средств (например, неизменяемой логической схемы), ручной обработки или комбинации этих вариантов осуществления. Термин «инструмент», «компонент» или «модуль», как использовано в настоящей заявке, в общем, представляет собой программное обеспечение, программно-аппаратные средства, аппаратные средства, целые устройства или сети, либо их комбинацию. В случае осуществления в программном обеспечении, например, это может представлять собой программный код, который выполняет конкретные задания при исполнении на процессоре (например, CPU или несколько CPU). Программный код может храниться на одном или нескольких машиночитаемых запоминающих устройствах, таких как, например, машиночитаемый запоминающий носитель данных. Характеристики и способы компонента являются независимыми от платформы, что означает, что они могут быть осуществлены на различных коммерческих платформах, имеющих разнообразные конфигурации обработки.

Как использовано в настоящем документе, термин «машиночитаемые носители» может включать в себя переходные и непереходные команды. Напротив, термин «машиночитаемые запоминающие носители» исключает переходные экземпляры. Машиночитаемые запоминающие носители могут включать в себя «машиночитаемые запоминающие устройства». Примеры машиночитаемых запоминающих носителей включают в себя энергозависимые запоминающие носители, как, например, RAM, и энергонезависимые запоминающие носители, как, например, жесткие диски, оптические диски и флэш-память, среди прочих.

Другие аппаратные средства 236 могут включать в себя устройства отображения, устройства ввода/вывода, сенсоры и так далее, которые могут быть реализованы на различных вычислительных устройствах.

Сервис 202 управления RACE может выполняться в облаке 104 и предоставлять сервис управления, который полностью совместим с API управления DSMS. Таким образом, отдельные вычислительные устройства 102(1)-102(N) могут направить их описательную логику облачно-граничных приложений сервису 202 управления RACE в качестве регулярных временных описательных запросов, поддерживаемых соответствующими DSMS 222(1)-222(N). Необходимо отметить, что с точки зрения граничного устройства (например, вычислительных устройств 102(1)-102(N)), они просто связываются с обычным модулем DSMS.

Если посмотреть с другой точки зрения, сервис 202 управления RACE можно рассматривать как выполненный с возможностью взаимодействовать с приложением, выполняющимся в облаке и на отдельных граничных вычислительных устройствах в связи с облаком. Сервис 202 управления RACE может быть выполнен с возможностью воспроизводить модуль DSMS для приема временных описательных запросов от отдельных граничных вычислительных устройств.

Кратко, процессор 204 RACE можно рассматривать как перехватывающий и проводящий синтаксический анализ входящего запроса, адаптеров и типов от отдельных вычислительных устройств 102(1)-102(N), исполняющих приложение 228 по поиску друзей. Процессор 204 RACE затем компилирует эти вводы в объектное представление исходного запроса. Это объектное представление подается в модуль 206 конструктора графов, который преобразует исходный запрос в больший граф запросов. Например, больший граф запросов может включать в себя граничные входящие потоки и операторы. Граф запросов подается в модуль 208 оптимизатора для решения оптимального размещения оператора. Наконец, модуль 210 конструктора запросов может формировать представления типов, адаптеров и (под-)запросов для выполнения

на отдельном вычислительном устройстве 102(1)-102(N) или в облаке 104. Эти объекты отправляются в отдельные DSMS (через их API управления) соответствующих вычислительных устройств для выполнения логики приложения распределенным образом. Необходимо отметить, что хотя в этой конфигурации сервис 202 управления RACE и процессор 204 RACE реализованы в облаке 104, в других вариантах осуществления, альтернативно или дополнительно, сервис управления RACE и/или процессор RACE могут быть реализованы в одном или нескольких вычислительных устройствах 102(1)-102(N). Сервис управления RACE и/или процессор RACE, реализованные на вычислительных устройствах, могут быть автономными или работать во взаимодействии с соответствующим сервисом управления RACE и/или экземплярами процессора RACE в облаке.

Конструктор 206 графов можно рассматривать как получающий объектное представление запроса в качестве ввода вместе со статистикой потоковых скоростей передачи и информацией о метаданных по каждому вводу. Конструктор графов сначала может использовать объектное представление запроса для формирования шаблона запросов, который представляет собой шаблон или основу формирования расширенного графа запросов. Например, фиг. 3 изображает шаблон 302 запросов, выведенный конструктором 206 графов для запроса поисковика друзей, описанного выше по отношению ко второму абзацу на стр. 14.

Некоторые из входных потоков в шаблоне 302 запросов соответствуют потокам данных отдельных устройств, таких как, например, источники местоположения GPS. Конструктор 206 графов может создавать множество экземпляров шаблона запросов путем разделения таких потоков на множество вводов, один на ребро. Вводы медленно меняющихся справочных данных, таких как, например, социальная сеть, могут быть материализованы для ограничения размера сформированного графа запросов. Например, фиг. 4 изображает социальную сеть 400 из четырех пользователей P, Q, R и S. Фиг. 5 изображает соответствующие созданные экземпляры шаблонов 502(1), 502(2) и 502(3) запросов для запроса приложения по поиску друзей. Необходимо отметить, что для того, чтобы позволить делиться информацией и избежать дублирующих ребер в созданных экземплярах шаблонов запросов, созданный экземпляр источника и операторы присоединения тщательно именуются, как изображено на фиг. 5. Конечный этап - это вшить созданные экземпляры шаблонов 502(1)-502(3) запросов в завершенный граф запросов.

Фиг. 6 изображает конечный граф 602 запросов, полученный из созданных экземпляров шаблонов запросов, изображенных на фиг. 5. Необходимо отметить, что при комбинировании созданных экземпляров шаблонов запросов вершины (в созданных экземплярах шаблонов) с одинаковым именем соотносятся с одной и той же вершиной в конечном графе запросов. Например, вершина Join<GPS-P, SNP> совместно используется созданными экземплярами шаблонов для ребер (P; R) и (P; S).

Возвращаясь к фиг. 2, модуль 208 оптимизатора принимает конечный граф 602 запросов в качестве ввода и решает, где выполнять каждый оператор (например, часть запроса) в графе запросов, чтобы общие или коллективные издержки связи приложения были минимальными (или, по меньшей мере, сокращены). При участии тысячей или даже миллионов пользователей в облачно-граничной системе конечный граф запросов может быть огромным - содержащим миллионы операторов. Для такого огромного графа запросов оптимальное размещение оператора не очевидно. Модуль оптимизатора RACE может использовать разнообразные технологии для определения оптимального размещения оператора. Один такой способ описан ниже под заголовком «Оптимальное

размещение оператора». RACE может выполнять периодическую повторную оптимизацию для регулирования размещения изменений в графе запросов и/или статистике.

После того как решения для улучшенного/оптимального размещения оператора приняты, процессор 204 RACE имеет группу корневых графов (каждый состоит из направленных ациклических графов (DAG) временных операторов). Каждый такой граф соответствует некоторому местоположению (границе или облаку). Модуль 210 конструктора запросов может формировать объектные представления компонентов запроса (включая типы событий, адаптеры и запросы) для каждого графа. Модуль конструктора запросов может затем направить объектные представления соответствующим DSMS через плоскость 218 управления. Необходимо отметить, что два дополнительных адаптера могут быть установлены на каждом экземпляре DSMS - один для отправки данных события в плоскость 220 данных, а другой для приема данных события от плоскости данных.

Плоскость 218 управления RACE используется для развертывания сформированных фрагментов запроса и метаданных в облачный экземпляр и граничные экземпляры DSMS, с использованием API управления DSMS. Затруднением является то, что граничные устройства (например, телефоны) обычно не являются напрямую доступными или адресуемыми от сервиса 202 управления RACE. Напротив, сервис управления RACE может поддерживать сервер, с которым граничные устройства создают и поддерживают постоянные соединения, чтобы принимать команды управления, которые направляются экземплярам DSMS на границах. Во время выполнения запроса события идут потоком между граничными устройствами и облаком. Сервис 202 управления RACE может использовать отдельную плоскость 220 данных, которая расположена в качестве сервера в облаке 104 и к которой граничные вычислительные устройства 102(1)-102(N) могут присоединиться через плоскость 218 управления. Сформированные запросы на граничных вычислительных устройствах и в облаке подписываются на и публикуют именованные потоки, которые зарегистрированы в плоскости 220 данных. Плоскость данных маршрутизирует события из облака 104 в граничные вычислительные устройства 102(1)-102(N) и наоборот.

При участии тысяч или даже миллионов пользователей в облачно-граничной системе конечный граф запросов может быть огромным - содержащим миллионы операторов. Так как источники данных распределены (например, потоки данных GPS различных пользователей исходят из их граничных устройств), то размещение каждого оператора имеет свое влияние на непроизводительные издержки оценки запросов. Есть экспоненциально много различных комбинаций размещения операторов. Примитивный подход, согласно которому осуществляют поиск по всему пространству разработки, может быть невозможным. Дополнительно, рассмотрение совместного использования промежуточных результатов делает эту проблему еще труднее.

Следующее обсуждение относится к примеру эффективного алгоритма для оптимального размещения операторов путем усиления специальной «звездной» топологии облачно-граничных систем. Для некоторых вариантов осуществления правильность алгоритма может быть доказана с учетом двух предположений, указанных ниже. Дополнительно, издержки нахождения оптимального размещения могут быть очень низкими.

Предположение 1. Конечный вывод запросов относительно меньше, чем входящие потоковые данные, и, следовательно, его издержки можно игнорировать.

Это предположение обосновано при учете общей природы облачно-граничных

приложений. Дополнительно, на основании особенностей частной информации некоторые варианты осуществления могут ограничить дозволенные местоположения операторов. Например, потоковые данные могут включать в себя конфиденциальную личную информацию (например, геолокация отслеживает мобильный телефон). Граничный клиент может не желать представлять сырую информацию, пока она не обработана должным образом (путем исключения конфиденциальных данных из конечных результатов операции присоединения), или если она отправляется только тем узлам, которые авторизованы.

Предположение 2. Для любого присоединения A □□ B (где A и B являются входящими потоками присоединения) операция присоединения выполняется либо в облаке, либо на исходных узлах A или B.

Необходимо отметить, что это предположение не упрощает проблему размещения; по-прежнему существует экспоненциальное число возможных размещений операторов. До представления причины и предложенного алгоритма описаны несколько обозначений на основании графов.

Определение (Demand (Спрос)) Может быть обозначено как пара (v_1, v_2), у которой источник v_2 потоковых данных «запрашивает» (то есть, с которой ему необходимо коррелироваться) данные, сформированные другим источником v_1.

Определение (Demand Graph (Граф спроса)) С учетом облачно-граничного приложения, граф спроса G=(V, E) определяется следующим образом: группа вершин V={v/v является источником потоковых данных}, и E={(v_1, v_1)|(v_1, v_2) является парой спроса}. Каждое граничное e=(i,)∈E связано со скоростью r_{ij} передачи, указывающей скорость передачи потока v_j, которая запрашивается v_j.

Алгоритм 1. Сформировать граф спроса из графа запроса

func DemandGraph (G^Q=(V^Q, E^Q))

$V^D \leftarrow \phi$; $E^D \leftarrow \phi$

For $\forall v_1 \in V^Q$ do

suppose e^1=(v_2, v_1) $\in E^Q$, e_2=(v'_2, v_1) $\in E^Q$

$V^D \leftarrow V^D + \{v_1\}$

$E^D \leftarrow E^D + \{e'_1=(v_2, v'_2), e'_2=(v'_2, v_2)\}$

end for

return G^D=(V^D, E^D)

Фиг. 7 изображает соответствующий граф 702 спроса для запроса приложения по поиску друзей с учетом социальной сети, изображенной на фиг. 4. Ребра графа 702 спроса изображают взаимоотношения спроса. Например, ребро (GPS-P, SN_P) указывает, что считывание GPS с P (GPS-P) должно быть коррелированно с социальной сетью (SN_P). На графе спроса операторы присоединения рассматриваются как виртуальные источники данных на графе спроса (будто они производят результаты присоединений в качестве потоков). На самом деле, однозначное соответствие между графами спроса и графами запроса. С учетом графа запроса G^Q=(V^Q, E^Q) алгоритм 1 формирует соответствующий граф спроса G^D=(V^D, E^D). Граф запроса может быть перестроен из графа спроса с помощью следующего подобного алгоритма.

Задание: скачивание в сравнении с выгрузкой. В общем, известно, что решение об

оптимальном размещении операторов для распределенной оценки запроса является сложной задачей. Сущность предложенного алгоритма заключается в усилении особенного свойства сети облачно-граничной архитектуры. В этом случае граничные вычислительные устройства не могут связываться друг с другом напрямую. Напротив, для обмена информацией граничные вычислительные устройства должны выгружать или скачивать данные через облачные серверы.

Определение (Выгрузка или скачивание). С учетом графа спроса G=(V, E) для ребра (I, j)∈E, данная реализация характеризует v_j как «выгрузку» на (I, j), если, независимо от того, где размещен v_j (либо на граничном вычислительном устройстве или на облачном сервере), он всегда делает попытку иметь соответствующий поток (запрашиваемый v_j), доступный на облачном сервере; иначе, v_i характеризуется как «скачивание» на (i, j).

Интуитивно, как только вершина решает осуществить выгрузку на ребро (которое представляет собой требуемую корреляцию данных), нет причины скачивать какие-либо данные для этой корреляции с облачного сервера, так как корреляция может быть просто выполнена на облачной стороне (когда данные уже доступны на облачной стороне). Рассмотрим следующую лемму.

Лемма 1. С учетом графа спроса G=(V, E), в его оптимальном размещении оператора, ∀ (i, j)∈E, (i, j) должен быть в одном из двух состояний: либо v_i выгружает (но не скачивает), либо скачивает (но не выгружает) на (i, j).

Доказательство. Предположим, что вершина v_i∈V решает одновременно осуществлять выгрузку и скачивание на (i,). Оператор присоединения для соответствующей корреляции может быть размещен в трех местоположениях (в соответствии с предположением 2), а именно v_i, v_j и облаке. В этом случае оператор присоединения не может быть размещен в v_i в оптимальном размещении: v_i уже выгружает его поток. Оператор присоединения мог бы быть выполнен в облаке, и в таком случае он сбережет издержки связи для скачивания данных v_j' на v_i. Следовательно, v_i не осуществляет скачивание на (i,) (так как никакой оператор присоединения не размещен на v_i).

Лемма 1 оказывает поддержку заключению касаемо того, что, с учетом графа спроса G=(V, E), существует соотнесение из его оптимального размещения с группой решений касаемо выгрузки в сравнении со скачиванием, принятых на каждой вершине в G. Такая группа решений определена как задание.

Определение (Задание). С учетом графа спроса G=(V, E), задание A:E→{D, U} определено так: $A_{i,j}$=U, если вершина v_j решает выгрузить ее потоковые данные на ребро (i, j), иначе $A_{i,j}$=D.

Оптимальное размещение и его соответствующее задание могут быть обозначены как P^{opt} и A^{opt}. Фиг. 8 изображает оптимальное размещение (P^{opt}) для графа 702 спроса на фиг. 7. Фиг. 9 изображает соответствующее задание (A^{opt}). В оптимальном размещении оператора присоединение между GPS-P и SN_P выполняется в узле P, что означает, что разделенный граф SN_P социальной сети должен быть отправлен узлу P, то есть SN_P «выгружается» в облако, а GPS-P - нет. Это соответствует заданию, данному на фиг. 9.

Естественно задать вопросы: 1) существует ли обратное преобразование из A^{opt} в P^{opt}, и 2) существует ли эффективный алгоритм для нахождения A^{opt} с учетом графа

спроса. Обсуждение ниже первоначально относится к первому вопросу, а затем постепенно раскрывает ответ на второй вопрос.

Не все задания могут быть соотнесены с эффективным планом оценки. Есть фундаментальное ограничение: присоединение требует совместного местоположения всех его вводов. Следовательно, для любого присоединения, которое берет вводы из разных источников (граничных устройств), скачивание осуществляет максимум одно устройство.

Определение (Эффективность и Конфликт). С учетом графа спроса G=(V, E), задание A эффективно, если оно удовлетворяет следующему условию: $\forall e=(i,)\in E, A_{i,j}\neq D \vee A_{j,i}\neq D$. Ребро, которое нарушает это условие, называется ребром конфликта.

Например, фиг. 9 изображает эффективное задание с учетом графа спроса, изображенного на фиг. 7, так как для любой корреляции максимум один источник данных решает осуществлять скачивание. Если $A_{SNP,GPS\text{-}P}$ изменяется на скачивание, то оно признает задание недействительным, так как ребро (SN, GPS-C) является ребром конфликта.

Алгоритм 2. Вычислить размещение из задания

func Placement($G^Q=(V^Q, E^Q)$, Assign)

//Инициализировать размещение концевых вершин (то есть, непосредственных источников)

Placement←{}

for $\forall$ $v\in V^Q$ do

if $!\exists$ $e=(v', v)\in E^Q$ then

$Placement_v\leftarrow v$

end if

end for

//Определить размещение оператора восходящим образом

TopoOrder←V^Q sorted by topology sort.

for $\forall$ v∈TopoOrder in the bottom-up order do

Suppose $e_1=(v_1, v)\in E^Q$, $e_2=(v_2, v)\in E^Q$

if $Assign_{v1}$=D then $Placement_v\leftarrow Placement_{v1}$

else if $Assign_{v2}$=D then $Placement_v\leftarrow Placement_{v2}$

else $Placement_v\leftarrow$Cloud

end for

return Placement

Лемма 2. С учетом эффективного задания A, A может быть соотнесено с соответствующим размещением операторов.

Доказательство. Докажем по построению. Размещение операторов сделано восходящим образом (изображено в алгоритме 2). В качестве базового случая местоположения концевых вершин в графе запросов известны (заведомо как источники потоков). Для внутренней вершины (то есть виртуальной вершины, которая представляет собой оператор присоединения), в соответствии с предположением 2, она может быть размещена либо в облачном сервере, либо совместно располагаться с одним из вводов. Если все ее источники ввода решили осуществлять выгрузку, то оператор присоединения должен быть размещен в облаке; иначе, есть один и только один источник ввода (с учетом эффективного задания A), решающий осуществлять скачивание, тогда оператор

присоединения должен быть совместно расположен с источником ввода.

Теорема 4.5. Проблему оптимального размещения операторов можно сократить до нахождения эффективного задания с оптимальными издержками (выводится напрямую из Леммы 1 и Леммы 2).

Одноуровневые запросы присоединения

Это обсуждение начинается с простого сценария, где приложения определены как одноуровневые запросы присоединения. Это обсуждение будет расширено до многоуровневых запросов присоединения в последующем обсуждении.

Одинаковая частота спроса

Обсуждение сперва рассматривает особый случай одноуровневых запросов присоединения, в которых, для любой вершины i на графе спроса, частоты потока для всех исходящих ребер одинаковы, а именно, $\forall(i, j)\in E; r_{i,j}=r_i$. По существу, оператору присоединения требуются полные потоковые данные от каждого входящего потока для выполнения операции присоединения. Это соответствует запросам, где до присоединения не выполняется никакой фильтрации (как, например, проекция или выбор).

Вместо прямого рассмотрения издержек задания некоторые варианты осуществления могут вычислять прирост от переключения выгрузки и скачивания (который может быть положительным и отрицательным) по сравнению с базовым эффективным заданием - примитивное решение о том, что все вершины решают выгружать их потоковые данные. Путем переключения вершины i с выгрузки на скачивание прирост может быть вычислен следующим образом: $gain_i=r_i-\Sigma_{(i,j)\in E}r_j$. А именно, прирост можно рассматривать как выгоду не выгружать потоковые данные i по издержке скачивания всех потоков, которые коррелируются с потоком i.

Определение (Глобальная оптимальность). С учетом графа спроса G=(V, E), глобальное оптимальное задание - это эффективное задание A, которое максимизирует общий прирост.

Следующий способ нахождения задания A^{opt}, который дает глобальную оптимальность, рассматривает жадный подход, при котором каждая вершина в графе спроса локально принимает решение о задании на основании своей собственной выгоды.

Определение (Локальная оптимальность). С учетом графа спроса G=(V, E), для каждой вершины $v\in V$, локальное оптимальное задание для v - локальное решение о A_v, которое максимизирует локальную выгоду. Более конкретно, $A_v=D$, если и только если $gain_v>0$.

Может быть доказано, что локальная оптимальность на самом деле совместима с глобальной оптимальностью, что приводит к двум последствиям: во-первых, издержки на вычисление локальной оптимальности низкие, они линейны числу степеней вершины в графе спроса. Во-вторых, это означает, что проблема задания может быть разделена и решена параллельно. Это особенно важно в случаях, когда граф спроса огромен, так как этот способ может усилить обширные ресурсы вычисления в облаке для эффективного решения.

Теорема 1. С учетом графа спроса G=(V, E), задание $A=\{A_v|A_v$ = локальное оптимальное задание в v, $v\in V\}$ эффективно.

Доказательство. Доказательство от противного. Предположим, что существует ребро конфликта e=(i, j), что означает, что $A_i=D$ и $A_j=D$. $A_i=D$ обеспечивает $gain_i=r_i-\Sigma_{(i,j)\in E}r_j>0$. Следовательно, $r_i>r_j$. Подобным образом, $r_j>r_i$ может быть извлечено из $A_j=D$.

Противоречие.

Теорема 2. Локальная оптимальность совместима с глобальной оптимальностью, а именно, глобальная оптимальность может быть извлечена путем отдельного применения локальной оптимальности.

Доказательство. 1) Теорема 1 показывает, что задание, полученное путем отдельного применения локальной оптимальности, эффективно. 2) Каждая локальная оптимальность вычисляет максимальный прирост для изолированной физической линии, а глобальная оптимальность является просто добавлением приростов к физическим линиям.

Различные скорости спроса

Теперь расширим обсуждение для рассмотрения сценария, при котором, для конкретной вершины i, скорости потока, запрашиваемые каждой из других вершин, могут отличаться. Например, в случае приложения по поиску друзей частота события для конкретного пользователя может отличаться по отношению к каждому из его друзей. Здесь предполагается, что поток с более низкой скоростью может быть создан с использованием потока с более высокой скоростью, что соответствует запросам, применяющим фильтры выборки. Другими словами, фильтр, который нужен для выборки x событий/сек, может быть обеспечен другим фильтром, который проводит выборку y событий/сек, для любого $y \geq x$. При таком сценарии решения по выгрузке в сравнении со скачиванием нужно делать для каждого ребра (вместо каждой вершины) в графе спроса.

Предположим, что скорости $r_{i,vj}$ отсортированы в вершине i, так что $r_{i,v1}<r_{i,v2}<\ldots<r_{i,vp}$, и не трудно понять, что оптимальное задание для p отсортированных ребер должно иметь шаблон [U, …, U, D, …, D].

Определение (Локальная оптимальность). Рассмотрим прирост в задании $\forall\forall j \leq k$, $A_{i,vj}=U$, $\forall\forall j>k$, $A_{i,vj}=D$: $gain_{i,vk}=r_{i,vp}-r_{i,vk}-\Sigma_{k+1\leq s\leq p}r_{vs,i}$. Некоторые реализации могут выбирать $k=argmax_{1\leq j\leq p}gain_{i,vk}$ и конфигурировать задание в шаблоне, описанном выше.

Лемма 4.8. После применения локальной оптимальности в вершине i, где $A_{i,vj}=D$, предполагается, что $r_{i,vj}>r_{vj,i}$.

Доказательство. Доказательство от противного. Предположим, что $r_{i,vj}\leq r_{vj,i}$. В соответствии с определением локальной оптимальности:

$Gain_{i,vk}=r_{i,vp}-r_{i,vk}-\Sigma_{k+1\leq s\leq p}r_{vs,i}$

$Gain_{i,vj}=r_{i,vp}-r_{i,vj}-\Sigma_{j+1\leq s\leq p}r_{vs,i}$.

Необходимо отметить, что j>k, так как $A_{i,vj}=D$. Также, $gain_{i,vj}-gain_{i,vk}=r_{i,vk}+\Sigma_{k+1\leq s\leq j-1}r_{vs,i}+(r_{vj'}-r_{i,vj})>0$. Это создает противоречие (так как $gain_{i,vk}$ является оптимальным).

Теорема 3. Теорема эффективности (теорема 1) не изменяется.

Доказательство. Доказательство от противного. Предположим, что существует ребро конфликта $e(v_1, v_2)$. Применение Леммы 3 дает, что $r_{v1,v2}>r_{v2,v1}$ из $A_{v1,v2}=D$, и $r_{v2,v1}>r_{v1,v2}$ из $A_{v2,v1}=D$, что дает противоречие.

Многоуровневые запросы присоединения

При рассмотрении многоуровневых запросов присоединения могут быть трудности, которые мешают примитивному применению алгоритма, разработанного для одноуровневых запросов присоединения. Например, для одноуровневых запросов присоединения издержки выходящих потоков операторов присоединения не учитываются (так как предполагается, что конечный выход пренебрежительно мал по сравнению с входящими потоками). Однако это не так в случае с многоуровневыми запросами присоединения. Например, при примитивном применении алгоритма,

представленного в предыдущем разделе, граничное устройство может отдельно решать скачивать другие потоки и выполнять вычисление локально. Однако если граничные устройства осведомлены о том факте, что выходящий поток затем требуется для оператора более высокого уровня присоединения (чье оптимальное размещение находится на стороне облака), то они могут принять другое решение. Обсуждение ниже относится к тому, как эта проблема решается путем расширения алгоритма для одноуровневых запросов присоединения.

Предположение 3. Поток данных для конкретного ребра появляется не более чем в одном дочернем поддереве любого оператора в графе запросов.

Это обоснованное предположение, так как можно легко комбинировать потоки из одного граничного устройства в один поток, либо локально выполнить необходимые вычисления того, что эти потоки вовлечены. Необходимо отметить, что это предположение не препятствует общему использованию источника или быстрых результатов, и, более конкретно, оно всегда остается неизменным в случае, когда шаблон запроса является лево-углубленным деревом по различным источникам данных.

Размещение оператора нисходящим образом

Внутренние операторы присоединения в графе запроса можно рассматривать как виртуальные источники потоков, за исключением того, что необходимо решить их местоположения. Интуитивно, с учетом графа запросов, настоящий способ может принимать решения скачивания в сравнении с выгрузкой для операторов нисходящим образом. Например, решение может быть принято для конкретной вершины v_1, которая соответствует оператору присоединения, при условии, что известно местоположение, в котором его вывод должен быть отправлен (на основании решения о размещении, принятом его родительским оператором). Алгоритм для одноуровневых запросов присоединения может быть напрямую расширен путем дополнительного включения издержек отправления выходящего потока в место назначения.

Необходимо отметить, что единственное рассматриваемое место назначения - это облако. Например, даже если место назначения является другим граничным устройством (так как выходящий поток требуется другой вершиной v_2, расположенной в граничном устройстве), то способу не нужно учитывать часть, соответствующую скачиванию, в издержках отправления (то есть, издержках отправления выходящего потока из облака в это граничное устройство), так как эти издержки скачивания уже учтены при вычислении прироста для v_2. Необходимо отметить, что Предположения 1 и 3 убеждаются в том, что при учете вершины v_1 актуальное решение о размещении может быть не учтено для его места назначения, так как оно определенно будет размещено либо в облаке, либо на каком-то другом ребре, с которым v_1 (или ее поддерево) не пересекается. Это ключевое наблюдение делает возможным расширение алгоритма, и можно легко показать, что расширенный алгоритм по-прежнему гарантирует эффективное и оптимальное задание.

Выгрузка в сравнении со скачивание с нисходящим образом

Необходимо отметить, что предыдущий подход (для одноуровневых запросов присоединения) извлекает размещение операторов восходящим образом после того, как приняты решения выгрузки в сравнении со скачиванием. Алгоритм 3 можно корректировать для решения задания выгрузки в сравнении со скачиванием на основании задания родительских операторов вместо их размещения (так как размещение недоступно).

Как только решение родительской вершины v_1 известно, некоторые реализации

могут рассматривать, какое решение нужно принять для дочерней вершины v_2. Опять, v_2 имеет два выбора - выгружать либо скачивать.

В одном сценарии, если решение родительской вершины v_1 - скачивание, это означает, что не нужно делать усилия для того, чтобы иметь вывод, доступный в облачном сервере. Следовательно, при нахождении локальной оптимальности для v_2 издержки выходящего потока не учитываются при вычислении прироста.

В другом сценарии, если решение родительской вершины v_1 - выгрузка, это означает, что выходящий поток v_2 должен быть доступен в облачном сервере. Следовательно, при нахождении локальной оптимальности для v_2 издержки выходящего потока должны быть учтены.

Алгоритм 3 берет граф спроса G=(V, E) в качестве ввода и вычисляет оптимальное размещение оператора. Алгоритм применяет обобщенный сценарий, в котором он предполагает многоуровневый запрос присоединения, и скорости спроса на каждом ребре (то есть скорости, связанные с ребрами спроса, начинающимися от заданной вершины, могут отличаться). В соответствии с Теоремами 1 и 2 несложно понять, что извлеченное задание эффективно и оптимально.

Алгоритм 3. Вычислить оптимальное задание.

func Assignment($G^Q=(V^Q, E^Q)$, $G^D=(V^D, E^D)$)
//Вычислить локальную оптимальность нисходящим образом
TopoOrder$\leftarrow V^Q$ sorted by topology sort:
Assign$\leftarrow\{\}$;
for $\forall$ v $\in$ TopoOrder in the top-down order do
EStart$\leftarrow\{e_k=(v, v')| e_k\in E^D\}$
Сортировать EStart в соответствии с $r_{v,v'}$
$r^{max}\leftarrow max_{(v,v')\in EStart} r_{v,v'}$
for $\forall$ $e_k=(v_k,v'_k)\in$EStart do
$gain_k\leftarrow r^{max}-r_{vk,v'k}-\Sigma_{k+1\leq s\leq p} r_{vs,vk}$
$V_p\leftarrow$родитель v_k в графе запроса
if $Assign_{vp}$=U then
//Прирост должен включать в себя издержки вывода присоединения.
$gain_k\leftarrow gain_k+r_{(i,j)}$//$r_{(i,j)}$ является издержками результата присоединения
end if
end for
$k^{opt}\leftarrow argmax_{1\leq k\leq p} gain_k$
for $\forall 1\leq k<k^{opt}$ do $Assign_{v,k}\leftarrow$U
for $\forall k^{opt}\leq k\leq p$ do $Assign_{v,k}\leftarrow$D
end for
return Assign

Асимметричные издержки выгрузки/скачивания

До этого момента вышеуказанные способы функционировали на предположении о том, что издержки выгрузки и издержки скачивания одинаковы. Однако в действительности это может быть не так. Например, стоимость использования полосы

пропускания на единицу для выгрузки и скачивания может отличаться (например, провайдер облачного сервиса может ввести асимметричные издержки, чтобы побудить пользователей передавать данные в облако). В качестве другого примера, граничное устройство может по-разному расходовать батарею при выгрузке и скачивании.

Следующее обсуждение рассматривает асимметричные издержки выгрузки/скачивания. Издержки на единицу для выгрузки и скачивания обозначены как C^u и C^d. Для сценариев, где $C^u<C^d$, результаты для $C^u=C^d$, представленные в предыдущих разделах, остаются в силе. В основном, обоснование теоремы (Теоремы 1) ключевой эффективности не изменяется.

С другой стороны, решение об оптимальном размещении операторов является более трудной проблемой для случаев, когда $C^u>C^d$. Для особого случая, когда $C^d=0$, можно показать, что проблему оптимального размещения операторов сложно доказать путем сокращения от проблемы классического взвешенного минимального вершинного покрытия (WMVC). Существенным образом, теорема эффективности нарушается в этих случаях, следовательно, отдельное применение локальной оптимальности граничными устройствами может привести к конфликтам. В таких случаях эффективное задание все еще можно получить путем решения конфликтов, группируя вершины в графе спроса для выгрузки с более высокими скоростями. Следовательно, проблема сокращается до проблемы WMVC в остаточном графе, которому не хватает эффективного общего решения. Следующее обсуждение относится к условию. Если условие удовлетворено, то проблему оптимального размещения оператора можно решить эффективно.

Определение. С учетом графа спроса G=(V, E), отклонение вершины $v\in V$, S_v определено как отношение максимальной и минимальной скорости, связанной с выходящими ребрами от v. А именно, $S_v=\max_{(v,i)\in E,i)}r_{v,i}/\min_{(v,j)\in E}r_{v,j}$.

Определение. С учетом графа спроса G=(V, E), отклонение G определено как максимальное отклонение среди узлов в G. А именно, $S=\max_{v\in V}S_v$.

	Условие	Локальная сложность
Выбрать условия	Никакое	O(N), N=# друзей
	Выборка	O(N logN), N=# друзей
	Условие	Глобальная сложность
Сложность запроса	Одноуровневый	Параллелизуемый локальный алгоритм
	Многоуровневый	Локальный алгоритм нисходящим образом
Асимметричные издержки	$C^u<C^d$	Параллелизуемый локальный алгоритм
	$C^u>C^d$	DP с ациклическим остаточным графом

Таблица 1 показывает итоги алгоритма размещения операторов. Глобальная оптимальность достигнута во всех случаях.

Лемма 4. С учетом отклонения S графа G, если $C^d<C^u<(1+1/S)*C^d$, то после применения локальной оптимальности на всех вершинах остаточный граф G', который состоит из ребер конфликта, является ациклическим (то есть разделенные деревья).

Доказательство. Доказательство от противного. Предположим, что существует цикл $(v_1, v_2), (v_2, v_3), \ldots, (v_{(p-1)}, v_p), (v_p, v_1)$ в остаточном графе G'. С целью представления обозначим, что $v_0=v_p$, а $v_{(p+1)}=v_1$. Так как каждое ребро в цикле является ребром конфликта, то $\forall 1\leq i\leq p$ есть свободная грань, где

$C^u*\max(r_{vi,vi-1}, r_{vi,vi+1})>C^d*(r_{vi-1,vi}+r_{vi+1,vi})$.

Путем добавления этих неравенств можно получить, что

$C^u*\Sigma_{1\le i\le p}\max(r_{vi,vi-1}, r_{vi,vi+1})>$

$C^d*\Sigma_{1\le i\le p}(r_{vi-1,vi}, r_{vi+1,vi})=$

$C^d*\Sigma_{1\le i\le p}\max(r_{vi,vi-1}, r_{vi,vi+1})+$

$C^d*\Sigma_{1\le i\le p}\min(r_{vi,vi-1}, r_{vi,vi+1})$.

Следовательно, эта реализация может извлечь следующее противоречие:

$$C^u/C^d > 1+\frac{\sum_{1\le i\le p}\min(r_{v_i,v_{i-1}}, r_{v_i,v_{i+1}})}{\sum_{1\le i\le p}\max(r_{v_i,v_{i-1}}, r_{v_i,v_{i+1}})} > 1+1/S.$$

Теорема 4. Если $C^d<C^u<(1+1/S)*C^d$, то оптимальное размещение оператора можно найти в P-время.

Доказательство. Его можно вывести путем применения Леммы 4 о том, что G' ацикличен. Это обсуждение показывает, что для каждого дерева в остаточном графе G' его взвешенное минимальное вершинное покрытие можно найти за линейное время, используя динамический программный алгоритм.

Начиная с концевых вершин, для каждой вершины v рассмотрим издержки вершинного покрытия для поддерева (с корнем в v), имеющего (или не имеющего) v в группе покрытий. Для любой внутренней вершины v, если v не находится в группе покрытий, то все дети v должны быть в группе покрытий. Следовательно, $Cost^-_v=\Sigma_{i\in child(v)}Cost^+_v$. С другой стороны, если v находится в группе покрытий, то каждое поддерево может независимо выбирать свое вершинное покрытие: $Cost^+_v=c_v+\min_{i\in child(v)}(Cost^-_v, Cost^+_v)$.

Необходимо отметить, что для особого случая, когда скорости потока, запрашиваемые различными друзьями, одинаковы, оптимальное размещение можно найти за P-время, если $C^d<C^u<2*C^d$ (что не изменяется в большинстве практических сценариев). Эмпирически, даже если $C^u\ge 2*C^d$, то конфликтующие ребра по-прежнему формируют изолированные деревья.

Вывод

Таблица 1 подводит итог теоретическим результатам и сложности времени предложенного алгоритма размещения оператора, с учетом различных комбинаций сложности запроса, условий выбора и отношений издержек выгрузки/скачивания.

Алгоритм размещения операторов вычисляет глобально оптимальное решение путем

отдельного рассмотрения локальной оптимальности для каждой вершины в графе спроса. Это обсуждение доказывает, что локальная оптимальность совместима с глобальной оптимальностью (если $C^u \leq C^d$). Эффективный жадный алгоритм предложен для вычисления локальной оптимальности. При таком эффективном жадном алгоритме каждый узел отдельно выбирает решение, которое максимизирует локальный прирост.

Этот алгоритм обрабатывает как одноуровневые, так и более сложные многоуровневые запросы присоединения. В случае с многоуровневыми запросами присоединения внутренние операторы присоединения в графе запросов рассматриваются как виртуальные вершины. Локальная оптимальность может быть вычислена для каждой отдельной вершины нисходящим образом. Дополнительно, в общем случае, когда остаточный граф ациклический (для $C^u > C^d$), существует эффективный динамический программный (DP) алгоритм нахождения оптимального задания для графа спроса. Следовательно, можно определить оптимальное размещение оператора для графа запросов. Расширение этих идей до общих графов запросов с операторами в черном ящике также объяснено.

С учетом природы облачно-граничных приложений (которые, как правило, являются корреляциями данных в реальном времени), вышеописанное обсуждение сфокусировано в основном на запросах присоединения (с фильтрами выборки). Следующее обсуждение относится к тому, как предложенный алгоритм можно применить для поддержки общих графов запросов в облачно-граничной топологии. Обсуждение дополнительно объясняет, как можно обрабатывать динамизм времени исполнения, как, например, изменения в графе запросов и частоты события.

Обработка общих графов запросов

Граф G запросов определен как направленный ациклический граф (DAG) по группе операторов в черном ящике (обозначенных как O), где концевые узлы в G называют источниками, а корневые узлы называют получателями. Каждый оператор в O может принимать ноль (для источников) или несколько вводов, и его вывод может быть использован в качестве ввода для других операторов. Выбор и проекция являются примерами операторов с одним вводом, в то время как операция присоединения является примером операторов с двумя вводами (или оператором со множеством вводов для кустовых присоединений). Высокоуровневая интуиция алгоритма размещения оператора по-прежнему не изменяется в том смысле, что каждый оператор может отдельно решить (нисходящим образом), следует ли ему выгружать (или скачивать) свой ввод для оптимизации его локальных издержек. В этом случае эффективность задания по-прежнему гарантирована, как и раньше. Более того, с учетом рассмотрения операторов как черных ящиков, нет дополнительной возможности применить совместное использование вывода различных операторов. В этом случае совместимость локальной оптимальности и глобальной оптимальности по-прежнему не изменяется, согласно подобному обоснованию, что и в Теореме 2. Следовательно, проблема снова может быть сокращена до нахождения оптимальных заданий выгрузки/скачивания, и можно использовать предложенные эффективные алгоритмы локальной оптимальности.

Обработка динамизма

Некоторые случаи алгоритма предполагают доступность графа запросов и статистику скорости для всех потоков. Оптимальное размещение вычисляется на основании этой информации, собранной на этапе оптимизации. Однако граф запросов может меняться с течением времени, например, вследствие добавления и удаления ребер в социальной сети. Подобным образом, частоты событий также меняются с течением времени. Таким образом, может быть необходимо адаптироваться к этим переменам во время

выполнения. С учетом того, что предложенный алгоритм оптимизации весьма эффективен, периодическая повторная оптимизация является эффективным решением. Однако повторная оптимизация может столкнуться с издержками на внедрение (например, отправление сообщений плоскости управления, таких как, например, определения запросов). Если варианты осуществления выполняют повторную оптимизацию очень часто, то издержки на повторную оптимизацию могут перекрыть выгоду от повторной оптимизации.

Для решения этой проблемы одно решение заключается в использовании онлайнового алгоритма на основании издержек. Например, такой алгоритм может оценивать и поддерживать накопленные потери вследствие невыполнения повторной оптимизации, и выбирать выполнить повторную оптимизацию, если накопленные потери превышают издержки повторной оптимизации. Потенциально выгодное качество этого подхода заключается в том, что оно 3-конкурентноспособно - гарантировано, что общие издержки ограничены тройной оптимальностью (даже с учетом априорного знания об изменениях).

Вышеописанное обсуждение предлагает конкретные подробности конкретных реализаций RACE. RACE может поддерживать широкий класс облачно-граничных приложений в реальном времени. RACE решает две важные технические проблемы: (1) спецификацию таких приложений; и (2) их оптимизированное выполнение в облачно-граничной топологии. Что касается (1), то обсуждение показывает, что использование описательного временного языка запросов (как, например, LINQ для StreamInsight) для выражения этих приложений является очень производительным и интуитивным. Что касается (2), то использование модулей DSMS предлагается для совместной обработки и выполнения различных частей логики приложения на граничных устройствах и в облаке. Здесь, инновационные алгоритмы высокоэффективны и при этом доказуемо минимизируют глобальные сетевые издержки, одновременно обрабатывая асимметричные сети, общие графы запросов и совместное использование быстрых результатов. Вышеуказанные реализации RACE выполнены с возможностью работать с Microsoft® StreamInsight®, коммерчески доступной DSMS. Другие реализации могут быть выполнены с возможностью использовать другие опции DSMS.

Эксперименты с реальными базами данных показали, что оптимизатор RACE в разы более эффективен, чем способы оптимального размещения текущего уровня техники. Дополнительно, размещения, достигнутые путем настоящих реализаций, привели к ряду факторов с более низкими издержками, чем более простые схемы для приложений по поиску друзей в реалистичном графе социальной сети с 8:6 миллионами ребер. RACE легко параллелизовать внутри облака. Он также хорошо масштабирует, используя только одну машину с реальным внедрением до 500 граничных клиентов. Детали некоторых реализаций описаны выше на точном уровне степени детализации. Последующее обсуждение предлагает более широкое описание, которое может относиться к вышеуказанным осуществлениям и/или другим осуществлениям.

ДОПОЛНИТЕЛЬНЫЕ ПРИМЕРЫ СПОСОБА

Фиг. 10 изображает логическую блок-схему методики или способа 1000, который совместим, по меньшей мере, с некоторыми вариантами осуществления концепций настоящего изобретения.

На этапе 1002 способ может получить описательный запрос потоковых данных в облачно-граничной топологии, которая включает в себя множество граничных устройств и облачных ресурсов.

На этапе 1004 способ может преобразовать описательный запрос потоковых данных

в граф запросов, который отражает множество граничных устройств.

На этапе 1006 способ может определить, следует ли выполнять операторы графа запросов на отдельных граничных устройствах или в облачных ресурсах на основе использования ресурсов для облачно-граничной топологии.

Порядок, в котором описаны вышеуказанные способы, не следует понимать как ограничение, и любое число описанных этапов может быть скомбинировано в любом порядке для осуществления этого способа или альтернативного способа. Дополнительно, способ может быть осуществлен в любых подходящих аппаратных средствах, программном обеспечении, программно-аппаратных средствах или их комбинации, чтобы вычислительное устройство могло осуществить способ. В одном случае способ хранится на машиночитаемом носителе данных в качестве группы команд, чтобы их выполнение вычислительным устройством побуждало вычислительное устройство выполнять способ.

ВЫВОД

Хотя технологии, способы, устройства, системы и так далее, относящиеся к облачно-граничным ресурсам и их распределению, описаны на языке, конкретном для структурных признаков и/или методологических действий, необходимо понимать, что изобретение, определенное в приложенной формуле изобретения, необязательно ограничено конкретными описанными признаками или действиями. Скорее, конкретные признаки и действия раскрыты в качестве примеров осуществления заявленных способов, устройств, систем и так далее.

(57) Формула изобретения

1. Машиночитаемый носитель информации, на котором сохранены машиноисполняемые инструкции, которые при их исполнении вычислительным устройством предписывают вычислительному устройству выполнять действия, содержащие:

оценивание запроса потоковой передачи данных реального времени, который использует данные из множества различных граничных вычислительных устройств, каковое множество различных граничных вычислительных устройств выполнены с возможностью сообщаться с облачными ресурсами и опосредованно сообщаться друг с другом через облачные ресурсы, но не сообщаться напрямую друг с другом, при этом отдельные граничные вычислительные устройства включают в себя экземпляр приложения или часть приложения, которая выражена в описательном временном языке; и

сравнение использования ресурсов между первым сценарием, который задействует выгрузку данных запросов, ассоциированных с упомянутым запросом потоковой передачи данных реального времени, из упомянутого множества различных граничных вычислительных устройств в облачные ресурсы для обработки, и вторым сценарием, который задействует выгрузку данных запросов из всех, кроме одного из упомянутого множества, различных граничных вычислительных устройств в облачные ресурсы и скачивание данных запросов в подмножество этого множества различных граничных вычислительных устройств для обработки, каковое подмножество включает в себя упомянутое одно граничное вычислительное устройство.

2. Машиночитаемый носитель информации по п. 1, при этом сравнение использования ресурсов содержит сравнение, по меньшей мере, использования полосы пропускания, ассоциированное с упомянутой выгрузкой по первому сценарию и с упомянутыми выгрузкой и скачиванием по второму сценарию.

3. Машиночитаемый носитель информации по п. 2, при этом при сравнении использования полосы пропускания учитываются асимметричные затраты на выгрузку и скачивание данных между отдельными граничными вычислительными устройствами и облаком.

4. Машиночитаемый носитель информации по п. 1, в котором упомянутые действия дополнительно содержат, в случае когда использование ресурсов меньше во втором сценарии, предписание оставшейся части из упомянутого множества граничных вычислительных устройств выгрузить данные запросов в облачные ресурсы и затем предписание облачным ресурсам скачать данные запросов на упомянутое одно граничное вычислительное устройство.

5. Машиночитаемый носитель информации по п. 1, в котором упомянутые действия дополнительно содержат, в случае когда использование ресурсов больше во втором сценарии, предписание упомянутому множеству граничных вычислительных устройств, включая упомянутое одно граничное вычислительное устройство, выгрузить данные запросов в облачные ресурсы и предписание выполнения упомянутой обработки в облачных ресурсах.

6. Машиночитаемый носитель информации по п. 1, при этом сравнение использования ресурсов выполняется динамически таким образом, что учитываются параметры, относящиеся к облачным ресурсам, упомянутому множеству различных граничных вычислительных устройств и параметрам связи между облачными ресурсами и данным множеством различных граничных вычислительных устройств, при этом упомянутое сравнение повторяется итерационным образом для отражения изменений параметров.

7. Машиночитаемый носитель информации по п. 1, при этом упомянутые оценивание и сравнение выполняются отдельным граничным вычислительным устройством, которым сгенерирован упомянутый запрос потоковой передачи данных реального времени, либо упомянутые оценивание и сравнение выполняются облачными ресурсами, либо упомянутые оценивание и сравнение выполняются облачными ресурсами и каждым из упомянутого множества различных граничных вычислительных устройств.

8. Машиночитаемый носитель информации по п. 1, при этом упомянутое оценивание содержит перезапись упомянутого запроса потоковой передачи данных реального времени в виде направленного ациклического графа временных операторов, который ссылается на схемы множества потоков.

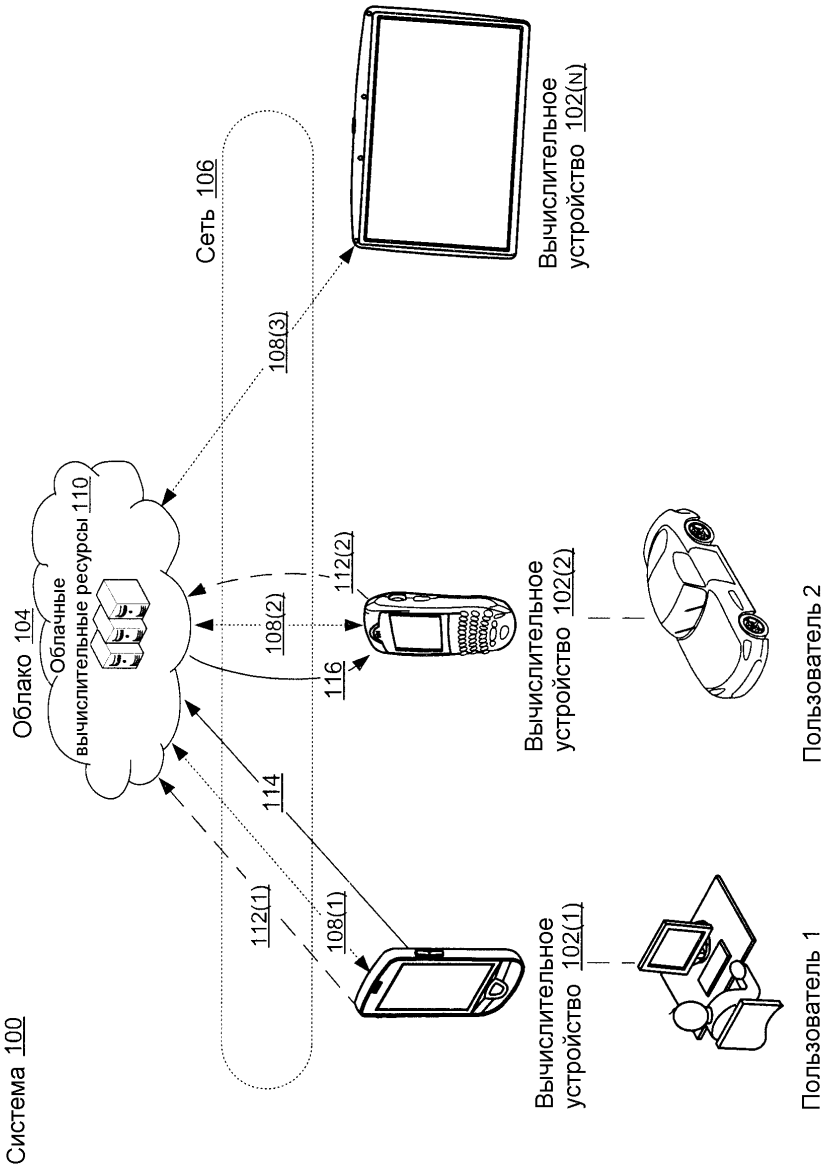
9. Машиночитаемый носитель информации по п. 1, при этом упомянутое оценивание запроса потоковой передачи данных реального времени содержит компиляцию запроса потоковой передачи данных реального времени в объектное представление.

10. Машиночитаемый носитель информации по п. 9, при этом оценивание объектного представления содержит граф запросов, где ребра графа запросов определены между упомянутым множеством граничных вычислительных устройств и облачными ресурсами.

1

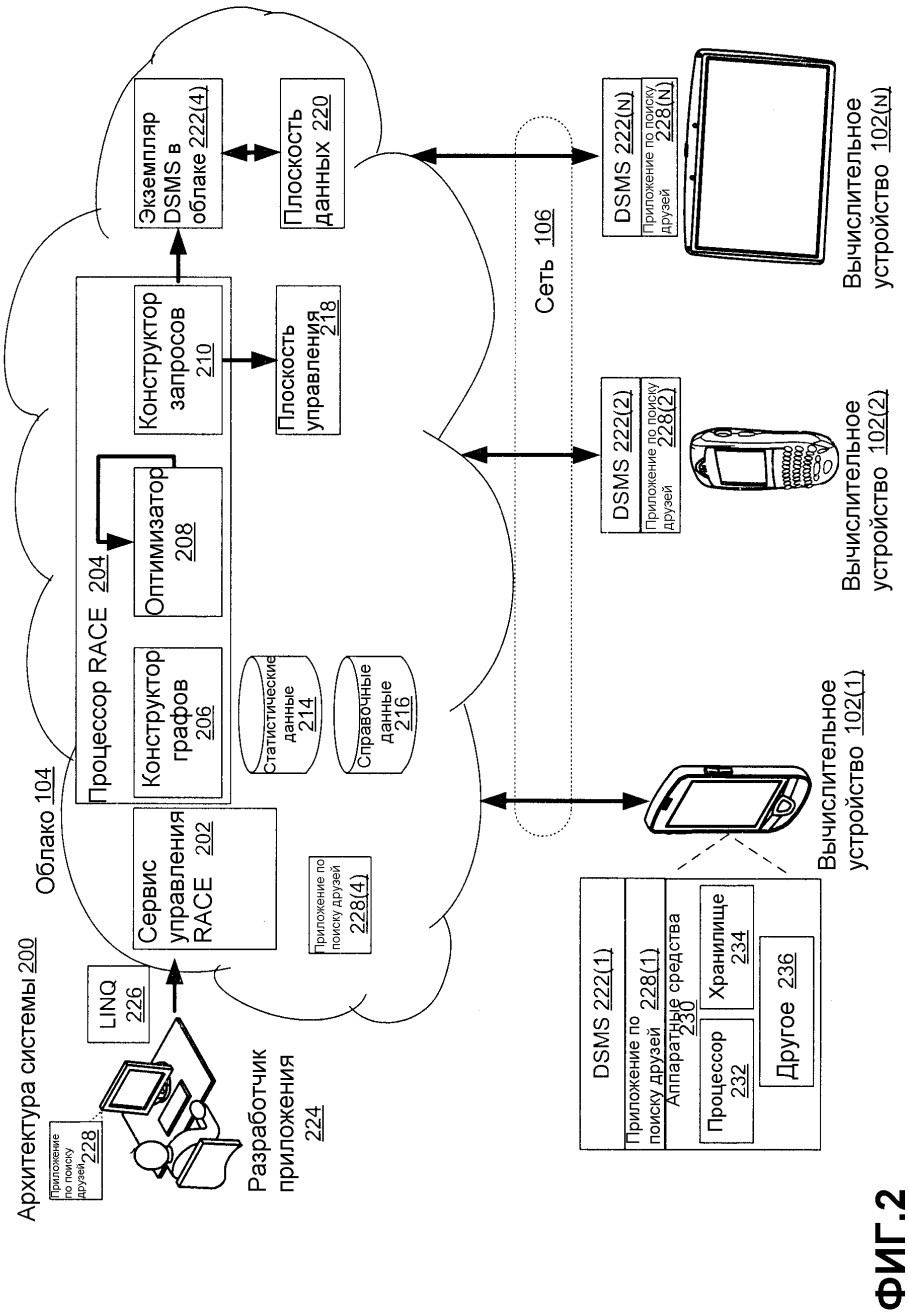
516284

1/7

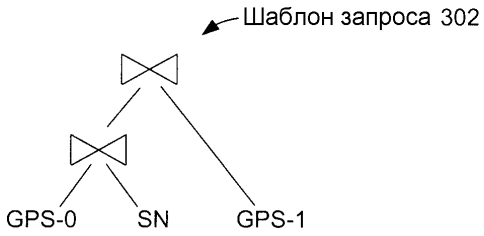


ФИГ.1

2

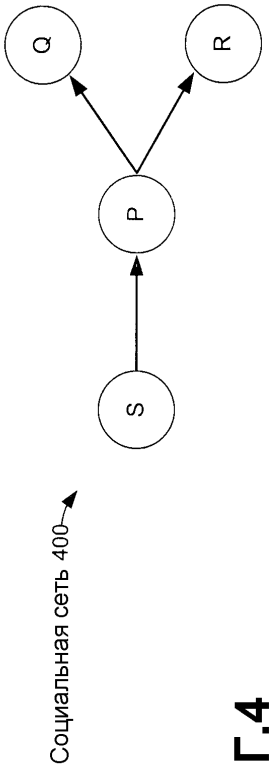

Архитектура системы 200
Облако 104
Приложение по поиску друзей 228
LINQ 226
Разработчик приложения 224
Сервис управления RACE 202
Процессор RACE 204
Конструктор графов 206
Оптимизатор 208
Конструктор запросов 210
Экземпляр DSMS в облаке 222(4)
Плоскость управления 218
Плоскость данных 220
Статистические данные 214
Справочные данные 216
Приложение по поиску друзей 228(4)
Сеть 106
DSMS 222(1)
Приложение по поиску друзей 228(1)
Аппаратные средства 230
Процессор 232
Хранилище 234
Другое 236
Вычислительное устройство 102(1)
DSMS 222(2)
Приложение по поиску друзей 228(2)
Вычислительное устройство 102(2)
DSMS 222(N)
Приложение по поиску друзей 228(N)
Вычислительное устройство 102(N)

ФИГ.2

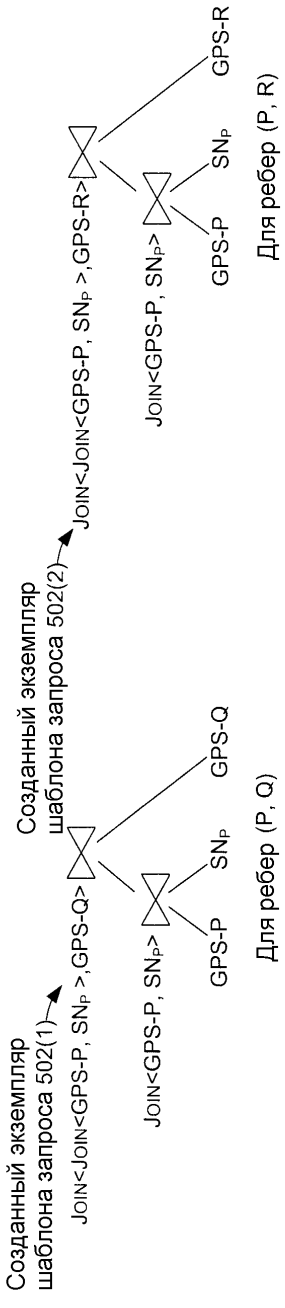


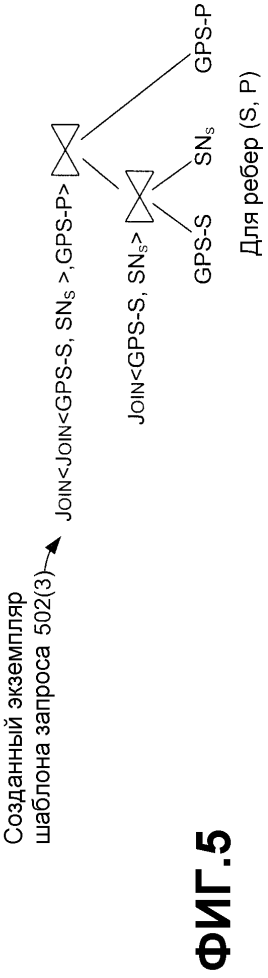
ФИГ.3

4

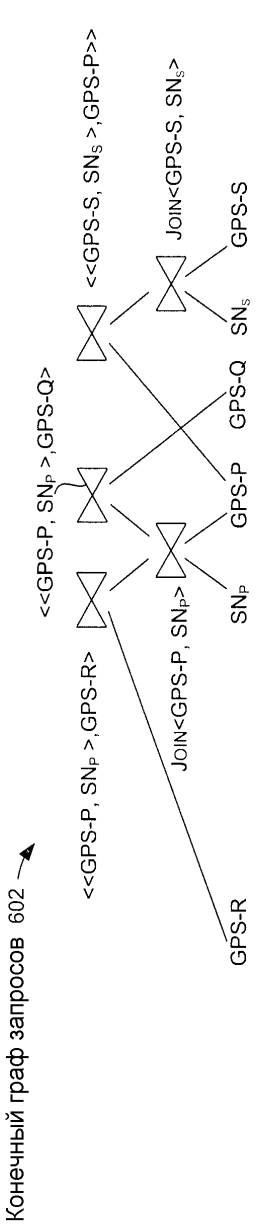

Социальная сеть 400
Q
P
R
S

ФИГ.4

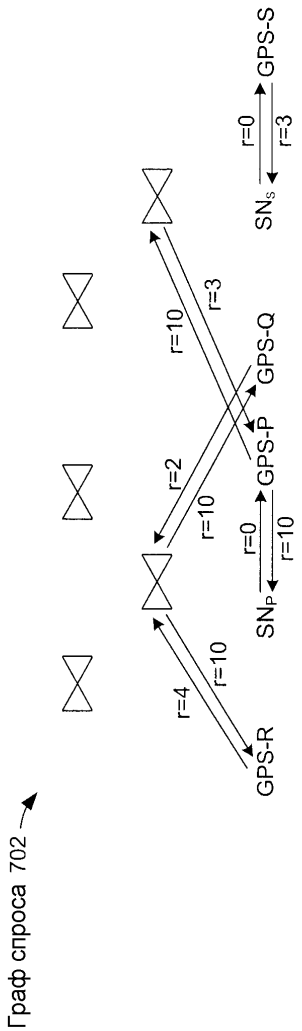

Созданный экземпляр шаблона запроса 502(1)
JOIN<JOIN<GPS-P, SNP >,GPS-Q>
JOIN<GPS-P, SNP>
GPS-P
SNP
GPS-Q
Для ребер (P, Q)
Созданный экземпляр шаблона запроса 502(2)
JOIN<JOIN<GPS-P, SNP >,GPS-R>
JOIN<GPS-P, SNP>
GPS-P
SNP
GPS-R
Для ребер (P, R)


Созданный экземпляр шаблона запроса 502(3)
JOIN<JOIN<GPS-S, SNS >,GPS-P>
JOIN<GPS-S, SNs>
GPS-S
SNS
GPS-P
Для ребер (S, P)

ФИГ.5

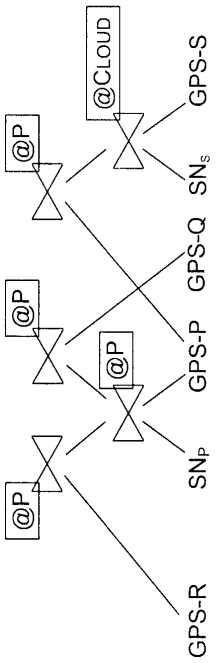

Конечный граф запросов 602
<<GPS-P, SNP >,GPS-R>
<<GPS-P, SNP >,GPS-Q>
JOIN<GPS-P, SNP>
<<GPS-S, SNS >,GPS-P>>
JOIN<GPS-S, SNS>
GPS-R
SNP
GPS-P
GPS-Q
SNS
GPS-S

ФИГ.6

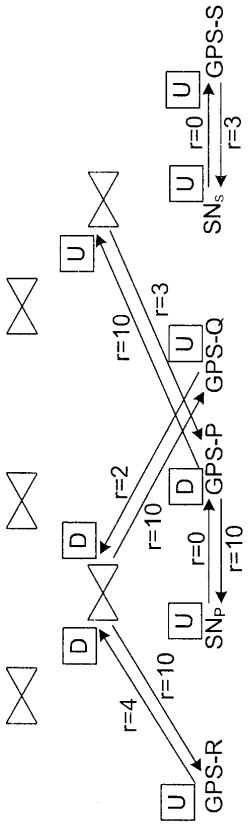

Граф спроса 702
r=4
r=10
r=2
r=10
r=10
r=3
r=0
r=10
r=0
r=3
GPS-R
SNP
GPS-P
GPS-Q
SNS
GPS-S

ФИГ.7

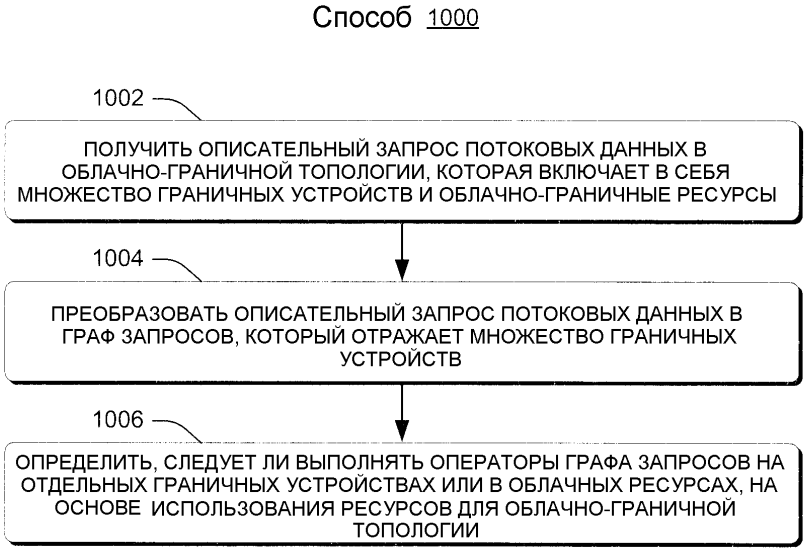
6


@CLOUD
@P
@P
@P
@P
GPS-S
SNS
GPS-Q
GPS-P
SNP
GPS-R

ФИГ.8


U
U
U
U
U
D
D
D
r=10
r=3
r=2
r=10
r=0
r=10
r=4
r=10
r=0
r=3
GPS-S
GPS-Q
GPS-P
SNS
SNP
GPS-R

ФИГ.9


Способ 1000
1002
ПОЛУЧИТЬ ОПИСАТЕЛЬНЫЙ ЗАПРОС ПОТОКОВЫХ ДАННЫХ В ОБЛАЧНО-ГРАНИЧНОЙ ТОПОЛОГИИ, КОТОРАЯ ВКЛЮЧАЕТ В СЕБЯ МНОЖЕСТВО ГРАНИЧНЫХ УСТРОЙСТВ И ОБЛАЧНО-ГРАНИЧНЫЕ РЕСУРСЫ
1004
ПРЕОБРАЗОВАТЬ ОПИСАТЕЛЬНЫЙ ЗАПРОС ПОТОКОВЫХ ДАННЫХ В ГРАФ ЗАПРОСОВ, КОТОРЫЙ ОТРАЖАЕТ МНОЖЕСТВО ГРАНИЧНЫХ УСТРОЙСТВ
1006
ОПРЕДЕЛИТЬ, СЛЕДУЕТ ЛИ ВЫПОЛНЯТЬ ОПЕРАТОРЫ ГРАФА ЗАПРОСОВ НА ОТДЕЛЬНЫХ ГРАНИЧНЫХ УСТРОЙСТВАХ ИЛИ В ОБЛАЧНЫХ РЕСУРСАХ, НА ОСНОВЕ ИСПОЛЬЗОВАНИЯ РЕСУРСОВ ДЛЯ ОБЛАЧНО-ГРАНИЧНОЙ ТОПОЛОГИИ

ФИГ.10