Федеральное государственное автономное образовательное учреждение
высшего образования
«Национальный исследовательский университет «Высшая школа экономики»

Факультет Санкт-Петербургская школа физико-математических и компьютерных наук
Департамент информатики

Основная профессиональная образовательная программа
«Прикладная математика и информатика»

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

на тему

Уменьшение фрагментации ресурсов при честном планировании на распределенных вычислительных кластерах

Выполнил студент группы БПМ151, 4 курса,

Тонких Андрей Александрович

Руководитель:

приглашенный преподаватель ФКН НИУ ВШЭ,

Колесниченко Игнатий Игоревич

Консультант(ы):

д-р физ.-мат. наук, профессор департамента информатики,

Омельченко Александр Владимирович

Санкт-Петербург 2019 г.

Оглавление

Введение

Современный подход к обработке больших объемов данных заключается в запуске вычислений на кластере из множества серверов, управляемых вычислительным фреймворком. Примером такого фреймворка является MapReduce [1]. Вычисления в MapReduce разбиваются на индивидуальные процессы, которые называются задачами. Каждая задача должна быть выполнена на одном из серверов кластера, и при этом она потребляет часть ресурсов сервера, таких как процессорное время и оперативная память. Если на сервере одновременно запустить слишком много задач, то эффективность может упасть из-за высокой конкуренции процессов за ресурсы. Если же, наоборот, на сервере запустить недостаточно задач, чтобы полностью утилизировать хотя бы один из его ресурсов, это естественным образом приводит к падению общей пропускной способности вычислительного кластера. Таким образом, эффективные стратегии планирования задач на кластере обычно стараются поддерживать утилизацию хотя бы одного ресурса на каждом сервере близкой к 100%, при условии наличия достаточного запроса от пользователей. Аллокация ресурсов, при которой не может быть запущенно новых задач без вытеснения уже работающих и превышения лимитов, называется Парето-оптимальной.

Однако даже при Парето-оптимальной аллокации, утилизация ресурсов кластера может быть далека от максимально возможной. Например, предположим, что в системе есть 2 одинаковых сервера и 2 типа задач: задачи первого типа используют 10% CPU и 5% памяти одного сервера, в то время как задачи второго типа используют 5% CPU и 10% памяти. При Парето-оптимальной аллокации возможна ситуация, при которой на одном сервере запущены 10 задач первого типа, а на другом -- 10 задач второго типа. Однако более эффективная стратегия планирования могла бы запустить 7 задач первого типа и 6 задач второго типа на каждом из серверов. При этом утилизация CPU на серверах была бы равна 100%, а утилизация памяти -- 95%. В данном примере первая аллокация оказалась значительно менее эффективной, чем вторая из-за фрагментации.

Борьба с фрагментацией может увеличить утилизацию ресурсов класте-

ра, что означает, что компании будет необходимо закупать меньше оборудования для выполнения тех же задач. С учетом того, что закупка оборудования может составлять существенную часть бюджета компании, даже небольшая экономия ресурсов в масштабах вычислительного фреймворка может сэкономить значительное количество денег для компании. Кроме того, высокая фрагментация также может негативно влиять на предсказуемость поведения системы.

Помимо высокой утилизации ресурсов, алгоритм планирования задач также часто должен отвечать другим свойствам. Одно из наиболее важных свойств -- честность. Интуитивно, честный алгоритм планирования старается поделить ресурсы между пользователями поровну или с учетом определенных весов. Так, например, если в системе есть 2 пользователя, и у первого вес 1, а у второго 0.5, то большинство честных алгоритмов будут стремиться как-то поделить все ресурсы на 3 части и отдать 2 части первому пользователю, и одну часть -- второму. На практике, помимо весов, часто используются другие дополнения к честному алгоритму. Например, можно гарантировать какому-то пользователю минимальное количество ресурсов, которое будет выделено его вычислениям. К сожалению, когда при планировании учитывается несколько видов ресурсов, определение честности становится неоднозначным. Подробнее об этом мы поговорим в главе 1. В общем случае, большинство честных алгоритмов определяют для каждого пользователя некоторую честную долю ресурсов, и, когда появляется возможность запустить новую задачу, запускают задачу пользователя с наименьшим отношением уже выделенных ему ресурсов к честной доле ресурсов для этого пользователя.

Существующие подходы к борьбе с фрагментацией в рамках честного планирования являются негибкими и либо слишком сильно жертвуют честностью, либо наоборот не предоставляют никакого способа улучшить упаковку в обмен на незначительное ухудшение честности.

Целью данной работы является разработка нового более гибкого подхода к борьбе с фрагментацией ресурсов на вычислительных кластерах для честных алгоритмов планирования задач. Для достижения цели были поставлены следующие задачи:

- Анализ различных высокоуровневых подходов к комбинированию

упаковки задач и честного планирования.

- Выбор наиболее гибкой архитектуры, которая в перспективе позволит улучшить упаковку и тонко регулировать влияние на честность планирования.
- Построение и анализ свойств конкретного алгоритма в рамках выбранной архитектуры.
- Реализация прототипа.
- Тестирование и проведение экспериментов.

В результате, был предложен новый подход для борьбы с фрагментацией, который заключается в том, чтобы позволить вычислениям отказываться от аллокаций в случаях, когда задачи вычисления плохо упаковываются на сервер. Данный подход был реализован в рамках MapReduce-системы YT [2] и протестирован на кластере из 73 вычислительных серверов. По результатам экспериментов, удалось показать, что предложенный подход действительно может значительно увеличивать утилизацию ресурсов за счет борьбы с фрагментацией. Анализ решения показал, что порождаемая им аллокация сходится к аллокации, порождаемой честным алгоритмом на кластерах, разделяемых между многими вычислениями. В рамках выбранной архитектуры, честность планирования может быть очень точно отрегулирована на уровне отдельных вычислений.

Последующие главы устроены следующим образом. В главе 1 приводится краткое описание наиболее релевантных работ в области планирования задач. В главе 2 подробно описывается архитектура предложенного решения. Глава 3 содержит описание и анализ используемого алгоритма принятия решений об отказах к аллокации. В главе 4 приводятся описание и результаты проведенных экспериментов на тестовом кластере. Заключение содержит краткое описание полученных результатов, а также потенциальные направления для развития.

1. Обзор литературы

1.1. Планирование на основе слотов

Исторически, планировщики разбивали ресурсы каждого сервера на куски фиксированного размера, называемые слотами. Затем они выделяли эти слоты под задачи, основываясь на некотором определении честности. Этот подход к планированию обычно называется планированием на основе слотов, и он раньше использовался в Hadoop [3, 4] и Dryad [5, 6]. Планирование на основе слотов обычно не учитывает, или слабо учитывает, как то, что на разных серверах ресурсы могут быть доступны в разном соотношении, так и то, что разные задачи могут требовать для запуска различное количество ресурсов. Это часто приводит к неэффективной утилизации ресурсов [7, 8].

1.2. DRF

Ghodsi и др.[7] показали, что эффективность MapReduce-фреймворка можно значительно увеличить, учитывая при планировании несколько типов ресурсов, в том числе процессорное время, оперативную память, и ширину сетевого канала. Они предложили Dominant Resource Fairness (DRF) -- определение честности для нескольких типов ресурсов. DRF основан на понятии доминантного ресурса. Доминантный ресурс -- тот ресурс, наибольшую долю которого потребляет пользователь. Например, если пользователю выделено $1/10$ всех CPU на кластере, $1/15$ всей оперативной памяти, и $1/15$ от общей ширины сетевого канала, то доминантным ресурсом для этого пользователя является CPU. Доминантная доля пользователя -- это доля потребления пользователем его доминантного ресурса. В приведенном примере доминантная доля пользователя равна $1/10$. DRF стремится уравнять доминантные доли всех пользователей, или, иначе говоря, максимизировать минимальную из доминантных долей среди всех пользователей.

Планировщик, основанный на DRF значительно эффективнее утилизирует ресурсы, чем планировщик, выделяющий слоты, в случаях, когда на кластере запускаются вычисления с различными требованиями к ресурсам. DRF также гарантирует выполение нескольких свойств, крайне важных в контек-

сте облачных вычислений. Среди прочих, DRF предоставляет:

- Sharing incentive (Стимул делиться): каждый пользователь получает не меньше ресурсов, чем если бы все ресурсы заранее были бы поровну (или с учетом весов) поделены между пользователями. Тривиальным следствием является отсутствие голодания -- состояния, когда пользователь совсем не получает ресурсов;
- Strategy proofness (Устойчивость к обману): пользователь не может получить больше ресурсов, если соврет о своих потребностях;
- Envy freeness (Отсутствие зависти): ни один пользователь не завидует аллокации другого пользователя с таким же весом;
- Pareto-efficiency (Парето-оптимальность): нельзя запустить новые задачи без вытеснения уже запущенных.

Существует множество работ, изучающих ограничения, возможные расширения, и применения DRF [9, 10, 11]. Также были предложены и другие определения честности для случая нескольких типов ресурсов [12].

1.3. Теоретическая модель

Несмотря на большой объем исследований в области честных стратегий планирования, большинство работ ограничиваются рассмотрением упрощенной модели, где все ресурсы объединены в единый пул, а также допускается выделение ресурсов под нецелое количество задач. На практике, кластер обычно состоит из набора отдельных вычислительных серверов с разнообразным количеством доступных ресурсов, а задачи неделимы, что делает выделение ресурсов под дробное количество задач непрактичным. Прямолинейное применение стратегий планирования, неадаптированных к реальному окружению, может привести к низкой утилизации ресурсов, и к невозможности выполнить заявленные свойства. Предметом исследования данной работы является одна из проблем, возникающих при переходе от теоретической модели к реальному миру, -- фрагментация ресурсов.

1.4. DRFH

Wang и др. [11] предложили простую эвристику для борьбы с фрагментацией. В каждый момент времени, их честный алгоритм диктует то, какая задача должна быть запущена следующей. Вместо того, чтобы запускать данную задачу на произвольном сервере, Wang и др. предложили выбирать сервер с наименьшим значением эвристической функции:

$$H_{drfh}(t, s) = \left\| D_t/D_{t1} - F_s/F_{s1} \right\|_1 , \tag{1.1}$$

где t -- идентификатор задачи; s -- идентификатор сервера; D_t -- вектор запроса ресурсов (demand) задачи t; F_s -- вектор свободных ресурсов на сервере s; D_{t1} и F_{s1} -- первые компоненты соответствующих векторов; $\|\cdot\|_1$ -- L^1-норма, т.е. сумма компонент вектора. Недостатком данной эвристики является то, что она влияет на планирование только в тех случаях, когда у планировщика есть выбор, на каком сервере запустить задачу. В реальности, вычислительные кластера часто находятся под высокой нагрузкой, и ресурсы большинства серверов полностью утилизированы большую часть времени, что следует из Парето-оптимальности стратегии планирования. Когда на каком-то сервере кластера какая-то задача заканчивает своё выполнение и освобождает ресурсы, цель планировщика состоит в том, чтобы запустить новые задачи и утилизировать освободившиеся ресурсы с минимальной задержкой, чтобы избежать простаивания ресурсов. Таким образом, на практике, планировщик обычно не имеет широкого выбора серверов, на которых можно запустить фиксированную задачу, что делает данную эвристику менее эффективной.

1.5. Tetris

Grandl и др. [8] описали Tetris -- планировщик, который стремится достичь максимальной эффективности за счет хорошей упаковки задач на сервера и ряда других оптимизаций. Подобно DRFH, Tetris использует эвристическую функцию для оценки того, как хорошо задача упаковывается на сервер:

$$H_{tetris\_max}(t, s) = \langle D_t, F_s \rangle, \tag{1.2}$$

где за $\langle \cdot, \cdot \rangle$ обозначено скалярное произведение векторов. Считается, что чем больше значение скалярного произведения, тем лучше задача упаковывается на сервер. Интересное свойство данной эвристической функции состоит в том, что её значение увеличивается линейно в зависимости от размера задачи. Для удобства, введем следующую функцию:

$$H_{tetris\_min}(t, s) = \frac{1}{H_{tetris\_max}(t, s)} = \frac{1}{\langle D_t, F_s \rangle}. \quad (1.3)$$

Grandl и др. исследовали компромисс между честностью и эффективностью упаковки и предоставили гибкий способ их комбинировать в Tetris. Каждый раз, когда сервер s информирует планировщик о доступных ресурсах, планировщик старается запустить новые задачи на данном сервере. Большинство честных стратегий в данном случае выбрали бы задачу наиболее обделенного пользователя. Tetris вместо этого сортирует всех пользователей по степени их удовлетворенности, определяемой честным алгоритмом, и аллоцирует задачу с наименьшим значением $H_{tetris\_min}(t, s)$, среди $(1 - f)$ доли списка, где f -- это произвольное число от 0 до 1, и является способом регулировать уровень честности. Например, при f, близком к 1, алгоритм сводится к честному планированию и не улучшает упаковку, в то время как при f, равном 0, алгоритм полностью игнорирует честность. Таким образом, разумные значения f лежат где-то между 0 и 1. Путем подбора значения f, Grandl и др. смогли достичь разумной честности и хорошей упаковки одновременно.

К сожалению, при значении f меньше 1, данный подход не может гарантировать sharing incentive. Более того, он не гарантирует отсутствие голодания. Например, в случае, если кластер состоит из набора одинаковых серверов, а доминантным ресурсом подавляющего большинства вычислений является CPU, то пользователь, запустивший вычисление с особенно низким потреблением памяти, рискует никогда не получить ресурсов, т.к. значение скалярного произведения для его задач будет всегда меньше, чем для задач других вычислений. Несложно придумать способ для борьбы с голоданием, однако фундаментальным свойством данного подхода остаётся то, что он предпочитает вычисления, которые лучше упаковываются на серве-

ры кластера, тем самым нарушая strategy proofness, envy freeness, и sharing incentive. Авторы статьи сознательно пошли на такие жертвы, т.к. Tetris разрабатывался для частного облака, где данные свойства менее критичны, чем в публичном облаке. Однако, с учетом того, что зачастую промышленные кластера имеют стабильную нагрузку, данный подход стимулирует пользователей неэффективно использовать непопулярные ресурсы, а также может приводить к плохому пользовательскому опыту для тех пользователей, чьи задачи плохо упаковываются на серверы кластера.

1.6. Выводы

Долгое время в области алгоритмов планирования задач исследования проводились в упрощенной модели, не учитывающей особенности реального устройства вычислительных кластеров и промышленной нагрузки. В такой модели был получен ряд важных результатов, на основе которых устроено большинство современных планировщиков. Однако при переходе от теоретической модели к практике возникают такие проблемы, как фрагментация ресурсов на серверах. Фрагментация может приводить к значительной недоутилизации ресурсов, что на больших масштабах может заметно повысить расходы компании на закупку вычислительных серверов, а также ухудшить предсказуемость поведения системы: то, сколько задач получится запустить на кластере начинает зависеть от того, как эти задачи распределятся по серверам. Интересной работой в области борьбы с фрагментацией является Tetris. Tetris значительно улучшает упаковку задач и предоставляет способ регулировать уровень честности на кластере, однако авторы Tetris сознательно пожертвовали гарантиями честности, что делает их решение менее универсальным. Другим примечательным результатом является DRFH. Авторы DRFH предлагают простую эвристику, улучшающую упаковку, совсем не влияя на честность. К сожалению, предложенный подход не предоставляет никакого способа регулировать качество упаковки. Целью моей работы является разработка более гибкой и универсальной эвристики, которая позволит улучшить упаковку с минимальным и легкоуправляемым влиянием на честность.

2. Архитектура решения

В данной главе описывается общая архитектура предложенного решения, а также известные альтернативные подходы.

2.1. Известные подходы

Для начала, давайте рассмотрим устройство честного планировщика, не оптимизирующего упаковку, на примере планировщика YT. Мы сфокусируемся на сценарии, при котором кластер полностью нагружен, и суммарный запрос пользователей значительно превышает количество ресурсов на кластере, так как это очень распространенный случай для промышленных кластеров, и в нём упаковка является релевантной задачей. В таком контексте, если средняя продолжительность задач не слишком маленькая, задержка, с которой планировщик запускает задачи, не слишком большая, а стратегия планирования является Парето-оптимальной, на большинстве серверов большую часть времени хотя бы один ресурс должен быть полностью утилизирован, т.е. планировщик не может запустить на них новых задач. Когда на каком-то сервере заканчиваются задачи, и освобождаются ресурсы, этот сервер информирует об этом событии планировщик. Чтобы избежать простаивания освобожденных ресурсов, планировщик старается запустить там новые задачи с минимальной задержкой. Таким образом, распространенной является реактивная модель, при которой планировщик выбирает, какие задачи нужно запустить в ответ на запрос сервера. При таком подходе запросы каждого сервера обрабатываются независимо от других запросов других серверов. Для ответа на запрос, честный планировщик выбирает наиболее обделенного пользователя, чьи задачи могут запуститься на данном сервере, запускает одну его задачу, после чего повторяет процесс для оставшихся ресурсов сервера, пока все ресурсы не будут распределены. Итого, мы получаем довольно ограниченную модель, где строго зафиксированы один пользователь и один сервер. В такой модели у планировщика почти нет гибкости, которая позволяла бы бороться с фрагментацией.

Естественный и, возможно, наиболее простой способ ослабить данную

модель -- это фиксировать один сервер и несколько наиболее обделенных пользователей. Подробно такой подход мы уже обсудили в контексте планировщика Tetris в разделе 1.5. Вкратце, он может приводить к хорошей упаковке, но ведет к существенному ухудшению честности и отсутствию таких свойств, как strategy proofness и sharing incentive.

Другой способ -- это фиксировать одно наиболее обделенное вычисление и выбирать из нескольких серверов тот, на который задачи этого вычисления лучше всего упаковываются. Похожий подход использовался в DRFH, однако, как описано в разделе 1.4, там он был реализован только в наиболее базовом варианте. Если пытаться адаптировать данный подход к описанной выше реактивной архитектуре планировщика, естественным способом было бы собирать запросы серверов в небольшие группы перед запуском алгоритма планирования, однако этот способ чреват увеличением задержки планирования, что естественным образом приводит к уменьшению утилизации, и может плохо работать для кластеров с небольшим количеством вычислительных серверов, т.к. там частота, с которой планировщик получает сообщения о свободных ресурсах, может быть недостаточной.

2.2. Предлагаемый подход

Вместо явного планирования задачи на группу серверов, предлагается использовать архитектуру, при которой контроллер вычисления имеет право отказаться от аллокации, предложенной планировщиком. В таком случае, планировщик предлагает ресурсы следующему вычислению в порядке, определяемом честностью, и так далее, пока какое-то вычисление не согласится запустить свою задачу на данном сервере. Если это произошло, планировщик запускает задачу данного вычисления и начинает процесс сначала, с оставшимися ресурсами сервера. Так происходит пока на сервере достаточно ресурсов, чтобы запустить новые задачи. Случай, когда все вычисления отказываются от аллокации, может обрабатываться по-разному, мы его подробно обсудим в разделе 2.3.

Контроллер вычисления отказывается от аллокации если считает (в соответствии с некоторых эвристическим алгоритмом), что задачи этого вы-

числения слишком плохо упаковываются на этот конкретный сервер относительно остальных серверов кластера. Для вычисления данного эвристического алгоритма, контроллер использует статистику о предыдущих предложениях. Подробнее про то, какая стратегия была выбрана для контроллера, мы поговорим в разделе 3. В общем же случае, каждое вычисление может иметь свою стратегию принятия решений. Это позволяет максимально гибко регулировать влияние упаковки на планирование. Например, на кластерах, где важные промышленные вычисления соседствуют с менее критичными аналитическими запросами, можно полностью или почти полностью отключить упаковку для важных вычисление и всё ещё получить низкую фрагментацию за счет аналитических запросов.

Похожая архитектура была использовано в [4] для решения проблемы локальности данных при планировании MapReduce задач. Важной отличительной чертой, однако, является тот факт, что в случае локальности данных, вычисления отказываются от запуска задач для собственной выгоды (локальность данных ускоряет выполнение задач), в то время как при борьбе с фрагментацией, вычисления отказываются от аллокаций в ущерб себе, ради общего блага: повышения пропускной способности кластера. Следствием данного факта является, например, то, что в контексте облачных вычислений, нельзя позволить пользователям свободно выбирать или настраивать поведения контроллера их вычислений. Пользователям почти всегда будет выгодно отключать упаковку.

Благодаря тому, что процесс отказа контроллером от аллокации в YT уже налажен и используется как для локальности данных, так и для ряда других целей, предложенное решение оказалось более простым в реализации, чем, например, решение с накоплением запросов от серверов, которое потребовало бы более радикального переписывания кода планировщика.

2.3. Крайний случай

Различные подходы могут быть использованы в случае, когда все вычисления отказываются от аллокации на сервере. Такая ситуация может легко возникнуть для сервера с сильно скошенными ресурсами. Например, можно

распределить все ресурсы сервера при помощи честного алгоритма, однако это чревато слишком плохой упаковкой. Можно сделать второй раунд аллокации со включенными отказами по упаковке, но ослабить некий барьер, чтобы увеличить шанс того, что одно из вычислений согласится на аллокацию. При таком подходе, если не гарантировать успешность при втором проходе, есть риск, что один запрос будет обрабатываться неограниченное количество времени. В текущей реализации, в данном случае алгоритм запустит одну задачу наиболее обделенного вычисления, игнорируя упаковку, после чего закончит обрабатывать запрос данного сервера даже если остались нераспределенные ресурсы. Поскольку каждый сервер периодически отправляет запросы к планировщику, свободные ресурсы будут заняты позже. Данный подход не претендует на оптимальность. В дальнейшем имеет смысл разработать более эффективный способ или научиться вовсе избегать подобных ситуаций.

2.4. Выводы и результаты по главе

В данной главе были рассмотрены различные подходы к комбинированию упаковки задач и честного планирования, и предложен новый гибкий подход, основанный на механизме отказов от аллокации.

3. Стратегия контроллера вычисления

В данной главе описывается пример стратегии принятия решений для контроллера вычисления.

Общая идея состоит в том, чтобы посмотреть на предлагаемые ресурсы как на некоторую случайную величину. Если предположить, что распределение этой случайной величины имеет продолжительные промежутки стабильности, то можно собрать статистику и соглашаться только на относительно хорошие предложения.

3.1. Эвристическая функция упаковки

Подобно DRFH и Tetris, было принято решение использовать эвристическую функцию для оценки качества упаковки задачи на сервер. Для оценки эффективности различных эвристических функций, был написан симулятор упрощенной модели планировщика. В рамках данного симулятора были реализованны прототипы всех подходов, описанных в главе 2. Симулировалась упрощенная модель, в которой задачи никогда не заканчиваются, а все вычисления начинались в один момент времени. Для симуляции использовались образы реальных промышленных кластеров Яндекса, а также случайным образом сгенерированные синтетические образы кластеров. Образ кластера -- это информация про ресурсы каждого из серверов кластера. В рамках данного симулятора это 2 числа на сервер: количество CPU и RAM. Также были собраны данные о нагрузке на промышленных серверах Яндекса и сгенерированы синтетические виды нагрузок.

Следующая эвристическая функция давала хорошие результаты на большинстве тестов:

$$H_{angle}(t, s) = 1 - \cos(\measuredangle(D_t, F_s)). \tag{3.1}$$

Данная функция ведет себя почти неотличимо от H_{drfh}, однако проще интерпретируется.

Для экспериментов также дорабатывался и использовался полноценный симулятор планировщика, задача которого просимулировать планировщик YT так, как если бы он использовался на реальном промышленном кластере.

Введем вспомогательную функцию:

$$H_{length}(t, s) = \frac{\|F_s\|}{\|D_t\|}, \tag{3.2}$$

где $\|u\|$ -- длина вектора.

В результате более поздних экспериментов на полноценном симуляторе планировщика, для прототипа была выбрана следующая эвристическая функция:

$$H_{angle\_length}(t, s) = H_{angle}(t, s) \cdot H_{length}(t, s) = \frac{(1 - \cos(\measuredangle(D_t, F_s))) \cdot \|F_s\|}{\|D_t\|}. \tag{3.3}$$

3.2. Принятие решений

Предлагается контроллеру каждого вычисления отказаться от какого-то количества s (например, 5) первых предложений, чтобы накопить минимальную статистику, а затем поддерживать окно из не более чем n последних предложений. После этого контроллер соглашается тогда и только тогда, когда среди предложений в окне находится не более k предложений с меньшим значением эвристической функции. Данный подход предполагает, что распределение ресурсов предлагаемых аллокаций будет меняться редко. Например, предполагается, что не будет ситуации, при которой каждое следующее предложение будет хуже предыдущего на протяжении продолжительного периода времени. Поддержание небольшого окна позволяет контроллеру быстро адаптироваться к изменениям в распределении предложений, также уменьшая вычислительную сложность одной проверки. Например, при $n = 15$ и $k = 3$, контроллер соглашается тогда и только тогда, когда текущее предложение оказалось среди 4 лучших в выборке размера 16 из случайной величины. Если предположить, что предложения независимы, то шанс, что контроллер согласится равен 25%.

Помимо ограничения на размер окна, выставляется также ограничение на то, насколько старые предложения будут находиться в окне. Это также нужно, чтобы гарантировать актуальность информации контроллера о распределении. Ограничение выставляется достаточно большим (например, 20

минут), чтобы если вычисление активно участвует в планировании, это ограничение не срабатывало. Оно важно, например, когда планирование вычисления было по каким-то причинам приостановлено и возобновлено через какой-то промежуток времени.

Чтобы избежать ненужных отказов, когда задачи какого-то вычисления хорошо упаковываются на большинство серверов кластера, было введено две дополнительные величины A и R и функция

$$significantlyBetterThan(lhs, rhs) = (lhs < rhs - A)\ \&\ (lhs < rhs/R), \tag{3.4}$$

где знаком & обозначено логическое ``И". Разумные значения для этих двух величин сильно зависят от выбранной эвристической функции. Например, для H_{angle_length} разумными могут быть значения $A = 0.05$ и $B = 1.5$. Алгоритм принятия решений был модифицирован таким образом, что контроллер отказывается тогда и только тогда, когда существуют больше k из n последних предложений, для которых выполняется $significantlyBetterThan(oldV, currentV)$, где за $oldV$ обозначено значение метрики предложения из окна, а за $currentV$ -- значение метрики текущего предложения. Интуитивно, контроллер отказывается тогда, когда может доказать, что в текущем распределении предлагаемых ресурсов (которое считается стабильным) есть достаточно много предложений, которые значительно лучше текущего. Это позволяет избежать отказов, например, когда задачи вычисления достаточно хорошо упаковываются на большинство серверов. Данная оптимизация опциональна. Она значительно уменьшает количество отказов контроллера и улучшает упаковку в некоторых случаях, однако при этом теряется свойство strategy proofness. Подробнее об этом мы поговорим в следующем разделе.

В качестве страховки от голодания, вызванного алгоритмом упаковки, имеет смысл добавить ограничение на количество подряд идущих отказов.

3.3. Настройка параметров

Предложенная стратегия имеет набор параметров (числа s, n, k, A, R).

При помощи регулировки параметра s можно обменивать качество упа-

ковки на задержку, с которой вычисление начинает участвовать в планировании. Для интерактивных вычислений имеет смысл установить $s = 0$.

Параметр n влияет на время принятия решения контроллером, поскольку для этого контроллер просматривает все предложения в окне. Есть технические трудности, которые не позволяют этот алгоритм легко оптимизировать. Например, связанные с тем, что требования задач вычисления к ресурсам могут иногда меняться. Также большие значения параметра n уменьшают адаптируемость контроллера к изменению распределения ресурсов предложений. Однако чем больше значение n, тем больше выборка контроллера, и тем точнее его информация о распределении.

Крайне важен параметр k. Он позволяет непосредственно регулировать частоту отказов контроллера от аллокации. В частности, если не использовать оптимизацию с функцией $significantlyBetterThan$, то ожидаемая доля аллокаций, на которые согласится контроллер, равна $\frac{k+1}{n+1}$. При $k = n$, контроллер будет всегда соглашаться, а при $k = 0$ будет соглашаться с вероятностью $\frac{1}{n+1}$.

Параметры A и R крайне чувствительны к тому, какая эвристическая функция используется. Они предоставляют менее очевидный способ регулировать частоту отказов и качество упаковки.

3.4. Сбор статистики

К сожалению, существует много причин, по которым задачи какого-то вычисления нельзя запустить на каком-то сервере. Помимо того, что на сервере может быть недостаточно ресурсов, сервер также может не подходить из соображений локальности данных, или по причине специфичных требований к расположению, характеристикам или оборудованию сервера, на котором вычисление готово запускать свои задачи. В стратегии контроллера важно отказываться от аллокации ради упаковки, основываясь только на данных о тех серверах, на которых данное вычисление действительно могло бы запустить свои задачи. Иначе легко можно оказаться в ситуации, когда вычисление будет отказываться от всех аллокаций.

Для обеспечения такой гарантии, контроллер каждого из вычислений ве-

дет собственную статистику о предлагаемых ресурсах (вместо одной общей статистики), и учитывает там только те предложения, на которых действительно была запущена задача, либо от которых контроллер отказался из соображений упаковки, после проведения некоторых других проверок. К сожалению, некоторые проверки требуют удаленного вызова и, соответственно, занимают слишком много времени. Такие проверки приходится делать уже после проверки на упаковку.

3.5. Анализ

При предположении о равномерности распределения предлагаемых ресурсов и отсутствии оптимизации с функцией $significantlyBetterThan$, рассматриваемая стратегия является strategy proof, т.к. распределение отказов одинаково для всех вычислений. К сожалению, при наличии вышеупомянутой оптимизации, возможен такой случай, что вычисление никогда не будет отказываться от аллокаций. Например, если оно везде упаковывается очень хорошо или очень плохо, что будет приводить к тому, что функция $significantlyBetterThan$ никогда не будет принимать значения $true$. У таких вычислений будет преимущество перед теми вычислениями, которые иногда будут отказываться.

В общем случае, для задачи динамической аллокации дискретных задач на набор серверов, свойства, такие как sharing incentive, envy freeness, и strategy proofness, можно гарантировать только в некотором ослабленном смысле. А именно, обычно доказывается сходимость к желаемой аллокации. Например, если в системе было запущенно новое вычисление, а все ресурсы уже заняты, у планировщика нет способа мгновенно удовлетворить данное вычисление без активного вытеснения задач других вычислений.

Если предлагаемая стратегия будет сходиться к честной аллокации, как если бы алгоритма упаковки не было, то алгоритм с упаковкой будет обладать теми же свойствами (в смысле сходимости), что и честный алгоритм, поверх которого он построен. Предположим, что в системе не появляется новых вычислений, а все уже запущенные вычисления состоят из бесконечного числа задач, причем все задачи каждого из вычислений требуют одинакового

количества ресурсов и конечны по времени. Это стандартные предположения для сходимости динамического алгоритма аллокации. При таких условиях, если какое-то обделенное вычисление C будет соглашаться на аллокации чаще, чем его задачи будут заканчиваться, то его потребление ресурсов будет постепенно увеличиваться, и со временем оно достигнет своей честной аллокации. Для этого необходимо, чтобы предложения аллокации поступали достаточно часто, хотя бы в 4 раза чаще, чем заканчиваются задачи этого вычисления для значения параметров $n = 15$, $k = 3$. Это обычно выполняется на больших кластерах, разделяемых между многими пользователями. К сожалению, для кластеров, где одно вычисление может, например, претендовать более, чем на половину всех ресурсов кластера, могут понадобиться доработки данного алгоритма принятия решений. В моем случае, данный сценарий был неприоритетным.

Для того, чтобы данная стратегия сходилась к честной аллокации в произвольном случае, требуются дополнительные модификации. А именно, нужно проконтролировать, чтобы контроллер соглашался на аллокации с неменьшей частотой, чем задачи данного вычисления заканчиваются. Для этого можно, например, отслеживать обе величины и увеличивать значение параметра k, когда контроллер соглашается недостаточно часто. Более глубокое изучение и тестирование данного вопроса оставлено для будущих исследований.

Интересным направлением для развития было бы использование алгоритмов машинного обучения для принятия решений. В частности, данная задача хорошо подходит для решения при помощи обучения с подкреплением.

3.6. Выводы и результаты по главе

В данном разделе была предложена несложная стратегия принятия решений для контроллера вычислений, а также определены направления для дальнейших исследований.

Предложенный алгоритм имеет несколько параметров, благодаря которым покрывает широкий спектр возможных применений. Также параметры могут регулироваться динамически, во время работы алгоритма.

Предложенная стратегия сходится к честной аллокации на кластерах, разделяемых между многими вычислениями.

4. Эксперименты на реальном кластере

Решение, описанное в главах 2 и 3 было реализовано в рамках планировщика MapReduce-системы YT [2]. В данной главе приводится описание экспериментов, проведенных на тестовом кластере для оценки эффективности упаковки.

В проведенных экспериментах при планировании учитывались только 2 ресурса: CPU и RAM, поскольку на практике именно эти ресурсы чаще всего заканчиваются первыми на серверах и приводят к фрагментации.

4.1. Устройство планировщика YT

В YT используется свой динамический алгоритм, который сходится к аллокации DRF. Все вычислительные серверы сообщают планировщику о свободных ресурсах раз в 5 секунд, а также когда на сервере освобождаются ресурсы за счет того, что какие-то задачи заканчивают своё исполнение.

В YT реализован алгоритм вытеснения, подробное описание которого выходит за пределы данной работы. Вкратце, если вычислению выделено меньше ресурсов, чем его честная доля, на протяжении слишком большого промежутка времени, планировщик YT разрешает данному вычислению вытеснять задачи тех вычислений, которым наоборот выделено слишком много ресурсов. К сожалению, эксперименты с выключенным вытеснением были бы нерепрезентативными, поскольку для одновременного обеспечения честности и высокой утилизации в условиях, когда требования задач к ресурсам могут сильно отличаться, вытеснение почти необходимо. Подробнее про вытеснение при планировании MapReduce-задач можно почитать, например, в [13].

4.2. Описание нагрузки

Тестовая нагрузка состояла из 24 пользователей с равными весами. Каждый пользователь в цикле запускал одно и то же вычисление. Параметры запускаемых вычислений можно видеть в таблице 4.1. Каждое вычисление состояло из job_count задач, каждая из которых потребляла cpu ядер про-

Таблица 4.1: Параметры запускаемых вычислений в эксперименте.

Пользователи	cpu	ram	job_count	$seconds\_mean$	$seconds\_deviation$
1,2	1	1	200	177	30
3,4	1	1	500	176	30
5,6	1	2	300	134	20
7,8	1	2	300	140	20
9,10	1	3	450	220	20
11,12	1	4	300	113	10
13,14	1	10	60	872	120
15,16	1	11	40	679	40
17,18	1	17	50	341	20
19,20	1	20	150	78	14
21,22	5	3	300	317	60
23,34	6	4	150	113	60

цессора и ram гибибайт оперативной памяти. Продолжительность задач генерировалась из нормального распределения с матожиданием $seconds\_mean$ и стандартным отклонением $seconds\_deviation$.

4.3. Описание кластера

Эксперименты проводились на кластере из 73 вычислительных серверов. Поскольку, по нашим наблюдениям, значительная фрагментация проявляется чаще, когда ресурсы на серверах в кластере наиболее разнообразны, для данных экспериментов ресурсы серверов были специальным образом рандомизированы. На графике 4.1 можно видеть распределение доступных ресурсов на серверах после рандомизации.

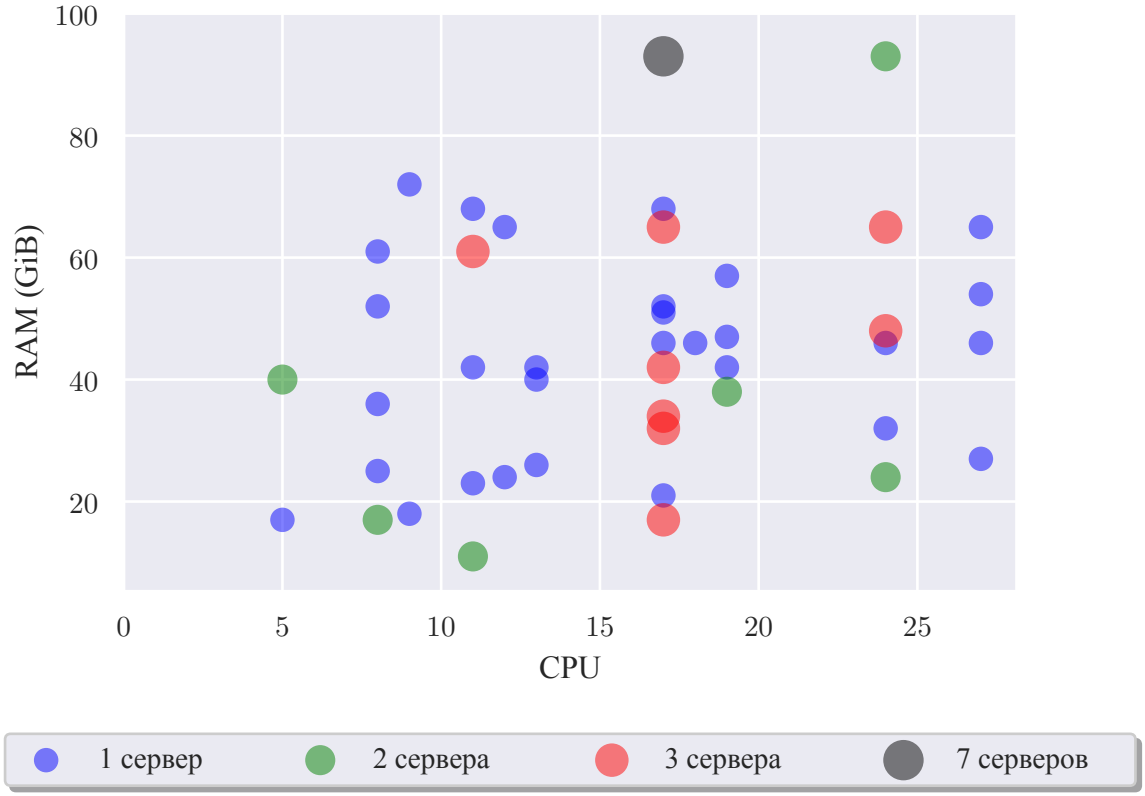

Рис. 4.1: Ресурсы на серверах тестового кластера.

4.4. Настройки контроллеров

Во всех контроллерах вычислений в данном эксперименте использовалась стратегия принятия решений, описанная в главе 3 с эвристической функцией H_{angle_length} и параметрами из таблицы 4.2. Подробное описание семантики каждого из параметров было приведено в разделе 3.2.

4.5. Результаты экспериментов

В таблице 4.3 можно видеть результаты проведенных экспериментов. В колонке ``Без упаковки" приведены результаты запуска тестовой нагрузки на кластере YT с выключенным алгоритмом упаковки. В колонке ``С упаковкой" приведены результаты запуска со включенным алгоритмом упаковки. Все величины выражены в процентах от общего количества ресурсов на кластере. Для оценки утилизации CPU без учета вытесненных задач, использовалась статистика о потреблении задачами CPU и информация о том, какие задачи не закончили свое исполнение из-за вытеснения. Статистика может

Таблица 4.2: Значения параметров контроллеров вычислений в эксперименте с упаковкой.

Количество отказов в начале (s)	5
Размер окна статистики (n)	15
Максимальное количество лучших предложений в окне (k)	2
Максимальная разница между значениями эвристической функции (A)	0.05
Максимальное отношение значений эвристической функции (R)	1.5

Таблица 4.3: Результаты экспериментов на тестовом кластере YT.

	Без упаковки	**С упаковкой**
Средняя утилизация CPU	92.2%	96.7%
Средняя утилизация CPU без учета вытесненных задач, приблизительно	90.6%	93.9%
Средняя утилизация RAM	91.5%	96.4%

быть не совсем точной, из-за чего данная величина является приблизительной.

На графиках 4.2 и 4.3 можно видеть сравнение утилизации CPU и RAM в проведенных экспериментах.

4.6. Выводы и результаты по главе

Эксперименты показали, что в рамках предложенной архитектуры действительно можно уменьшить фрагментацию и увеличить утилизацию ресурсов на кластере.

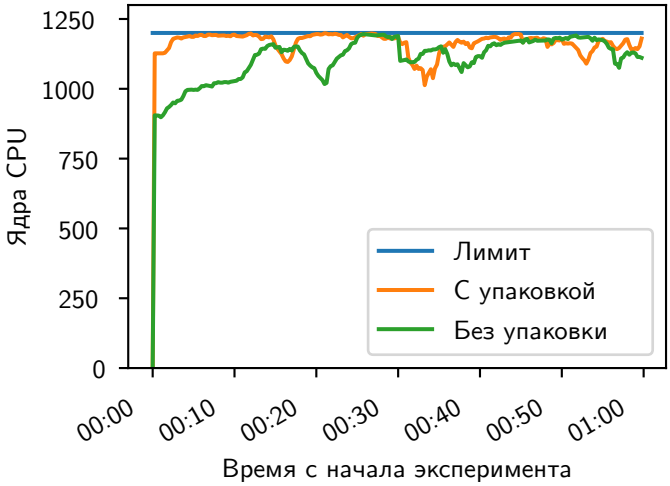

Рис. 4.2: Сравнение утилизации CPU.

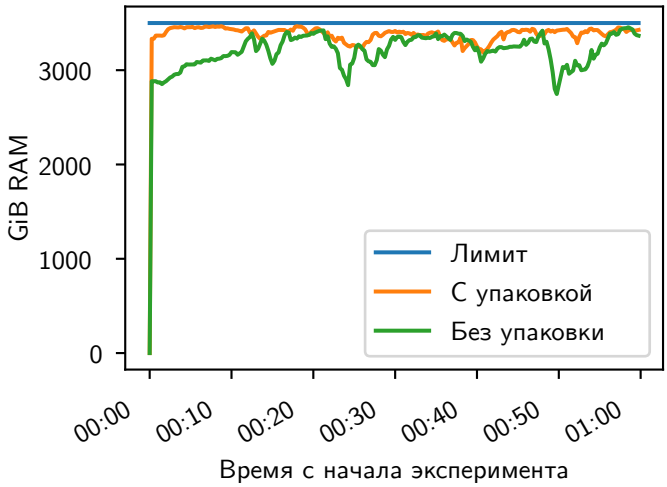

Рис. 4.3: Сравнение утилизации RAM.

Заключение

В данной работе был предложен новый подход к борьбе с фрагментацией при честном планировании задач на вычислительном кластере, отличающийся гибкостью, хорошей конфигурируемостью, и простотой в реализации. Предложенный подход был реализован в рамках MapReduce-системы YT и протестирован на кластере из 73 серверов с синтетической нагрузкой.

В результате получилось значительно уменьшить фрагментацию и, соответственно, увеличить утилизацию ресурсов на кластере, что, в перспективе, может сэкономить для компании значительное количество денег. В дальнейшем, код предложенного решения предстоит подготовить к использованию на промышленных кластерах, состоящих из десятков тысяч вычислительных серверов.

Два наиболее интересных направления для дальнейшего развития -- это обеспечение строгих гарантий честности и хорошей упаковки в наиболее общем случае путем улучшения стратегии принятия решений контроллером вычислений, а также использование алгоритмов машинного обучения для принятия решений. Также полезным может оказаться изучение возможных стратегий для крайнего случая, когда все вычисления отказываются от аллокации, или подходов, которые позволят полностью избежать таких ситуаций.

Список литературы

[1] Dean Jeffrey, Ghemawat Sanjay. MapReduce: simplified data processing on large clusters // Communications of the ACM. — 2008. — Vol. 51, no. 1. — P. 107--113.

[2] YT: зачем Яндексу своя MapReduce-система и как она устроена. — 2016. — URL: `https://habr.com/ru/company/yandex/blog/311104` (дата обращения: 23.05.2019).

[3] Apache Hadoop. — URL: `https://hadoop.apache.org` (дата обращения: 23.05.2019).

[4] Zaharia Matei, Borthakur Dhruba, Sen Sarma Joydeep et al. Delay scheduling: a simple technique for achieving locality and fairness in cluster scheduling // Proceedings of the 5th European conference on Computer systems / ACM. — 2010. — P. 265--278.

[5] Isard Michael, Budiu Mihai, Yu Yuan et al. Dryad: distributed data-parallel programs from sequential building blocks // ACM SIGOPS operating systems review / ACM. — Vol. 41. — 2007. — P. 59--72.

[6] Isard Michael, Prabhakaran Vijayan, Currey Jon et al. Quincy: fair scheduling for distributed computing clusters // Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles / ACM. — 2009. — P. 261--276.

[7] Ghodsi Ali, Zaharia Matei, Hindman Benjamin et al. Dominant Resource Fairness: Fair Allocation of Multiple Resource Types. // Nsdi. — Vol. 11. — 2011. — P. 24--24.

[8] Grandl Robert, Ananthanarayanan Ganesh, Kandula Srikanth et al. Multi-resource packing for cluster schedulers // ACM SIGCOMM Computer Communication Review. — 2015. — Vol. 44, no. 4. — P. 455--466.

[9] Parkes David C, Procaccia Ariel D, Shah Nisarg. Beyond dominant resource fairness: Extensions, limitations, and indivisibilities // ACM Transactions on Economics and Computation (TEAC). — 2015. — Vol. 3, no. 1. — P. 3.

[10] Bhattacharya Arka A, Culler David, Friedman Eric et al. Hierarchical scheduling for diverse datacenter workloads // Proceedings of the 4th annual Symposium on Cloud Computing / ACM. — 2013. — P. 4.

[11] Wang Wei, Li Baochun, Liang Ben. Dominant resource fairness in cloud computing systems with heterogeneous servers // IEEE INFOCOM 2014-IEEE Conference on Computer Communications / IEEE. — 2014. — P. 583--591.

[12] Dolev Danny, Feitelson Dror G, Halpern Joseph Y et al. No justified complaints: On fair sharing of multiple resources // proceedings of the 3rd Innovations in Theoretical Computer Science Conference / ACM. — 2012. — P. 68--75.

[13] Ananthanarayanan Ganesh, Douglas Christopher, Ramakrishnan Raghu et al. True elasticity in multi-tenant data-intensive compute clusters // Proceedings of the Third ACM Symposium on Cloud Computing / ACM. — 2012. — P. 24.