Московский государственный университет имени М.В. Ломоносова
Факультет вычислительной математики и кибернетики
Кафедра автоматизации систем вычислительных комплексов

КУРСОВАЯ РАБОТА

Метапланировщик для территориально распределенных ЦОД

Выполнил:
студент 521 группы
Боднарюк Василий Васильевич

Научный руководитель:
с.н.с., к.т.н. Костенко В.А.

Москва, 2018

Аннотация

В курсовой работе исследуется вопрос метапланирования над совокупностью центров обработки данных. Приводятся основные понятия и ставится задача метапланирования, рассматриваются метапланировщики, применяемые в различных распределенных системах.

В работе предложен и описан алгоритм метапланирования для распределенного ЦОД. Некоторые операции представлены в общем виде. Приведены примеры их возможной конкретизации.

В рамках работы алгоритм реализован программно. Проведены экспериментальные исследования и получены результаты для сравнения.

Содержание

1. Введение

В современном мире вычисления консолидируются в центрах обработки данных. Эффективность работы ЦОД зависит в числе прочего от используемых алгоритмов планирования. Развитием алгоритмов планирования в ЦОД могут выступать учет физической архитектуры, учет политик размещения виртуальных машин пользователей, а также метапланирование, организуемое в рамках группы ЦОД.

Курсовая работа посвящена метапланированию над совокупностью локальных ЦОД. Подобный распределенный ЦОД может встречаться как у компаний, занимающихся предоставлением вычислительных ресурсов пользователям (“Ростелеком”), так и у корпораций, использующих ЦОД для обеспечения собственных потребностей в вычислениях (“Сбербанк”). Из заявлений представителей таких компаний, как Facebook, можно заключить, что ЦОД, предназначенный для внутренних нужд корпорации, обслуживает по большей части независимые задачи[1]. В связи с этим планирование вычислений крайне актуально и для корпоративных ЦОД.

Метапланирование находит место в таких системах, как GRID и облачные вычисления. Если в первом случае работа метапланировщика полностью автоматизирована и часто осуществляется на отдельном мастер-узле, то во втором уровень метапланирования возникает в виде статического плана, организуемого пользователем с целью распределить выполнение задачи на независимых виртуальных сущностях, которые можно разместить в том числе на ресурсах различных облачных хостинг-провайдеров.

Метапланирование в контексте ЦОД обладает рядом особенностей, которые не позволяют перенимать подходы GRID и облаков. При формировании плана в GRID учитываются издержки, которые в случае с ЦОД не возникают. Пользовательский план облачных вычислений строится в предположении неограниченности доступных вычислительных ресурсов, в связи с чем такие вопросы, как миграция, на уровне метапланирования не решаются. Однако организация миграции может найти место при метапланировании в рамках группы ЦОД.

Таким образом, задача метапланирования в распределенном ЦОД является актуальной и имеет ряд направлений для исследования.

2. Описание распределенного ЦОД

2.1 Архитектура распределенного ЦОД

Распределенный ЦОД включает в себя некоторое количество локальных ЦОД, обслуживающих запросы на вычисление. Каждый запрос предполагает выделение вычислительных ресурсов под свои нужды. Объединение локальных ЦОД в распределенный производится с целью организации совместного планирования вычислений.

Распределенный ЦОД включает в себя несколько независимых локальных ЦОД. Локальному ЦОД присущ набор серверов и хранилищ данных, а также множество каналов связи между ними. Вышеназванные элементы ЦОД характеризуются физическими характеристиками – ресурсами.

К физическим ресурсам сервера относятся:

1. количество вычислительных ядер;
2. размер оперативной памяти.

Физическим ресурсом хранилища данных является размер хранилища.

Каналы связи характеризуются единственным физическим ресурсом – пропускной способностью. По каналам, с участием коммутаторов, обеспечивается связь между серверами и хранилищами данных.

Для размещения на физических мощностях запросов на вычисление локальные ЦОД имеют собственные планировщики, реализующие локальные алгоритмы планирования, вероятно различные. Получая на вход запросы на размещение, алгоритм планирования пытается разместить эти запросы на вычислительных ресурсах, руководствуясь соображениями оптимальности, которыми могут являться, например, максимизация числа размещенных запросов [2] или снижение энергопотребления [3].

В распределенном ЦОД над уровнем локального планирования вводится уровень метапланирования. Задача метапланировщика – распределить запросы общего пула по пулам локальных планировщиков. В расширенной версии алгоритм метапланирования может допускать перенесение ранее размещенных запросов из одного локального ЦОД в

другие. Подобное имеет смысл, если в результате удастся разместить суммарно большее число запросов.

2.2 Запрос на обслуживание

Запрос на обслуживание представляет из себя задание исполнить на вычислительных ресурсах серверов ЦОД некую вычислительную задачу. Запросы присутствуют в так называемом пуле запросов.

Запросу присущ набор физических требований, сопоставимых с физическими ресурсами серверов ЦОД. Элементами запроса являются:

1. виртуальные машины, характеризующиеся количеством вычислительных ядер и размером оперативной памяти;
2. хранилища данных, характеризующиеся размером памяти;
3. каналы связи между виртуальными машинами и хранилищами данных, характеризующиеся емкостью.

Сопоставление элементов запроса физическим ресурсам серверов может быть реализовано в рамках модели Back-of-tasks [4], то есть в предположении, что планирование осуществляется статически: на момент запуска алгоритм обладает полной информацией о количестве запросов и их структуре, при этом информация о времени поступлении запросов никак не учитывается. Фактически это означает, что пул запросов рассматривается алгоритмом как множество. В подобной модели задача планирования может решаться жадными алгоритмами [2], [5].

3. Постановка задачи метапланирования в ЦОД

3.1 Содержательная постановка задачи

Как отмечено выше, независимые ЦОД собраны в группу с целью организации совместного планирования и обработки единого пула запросов. Схематично роль метапланировщика иллюстрируется на рисунке 1.

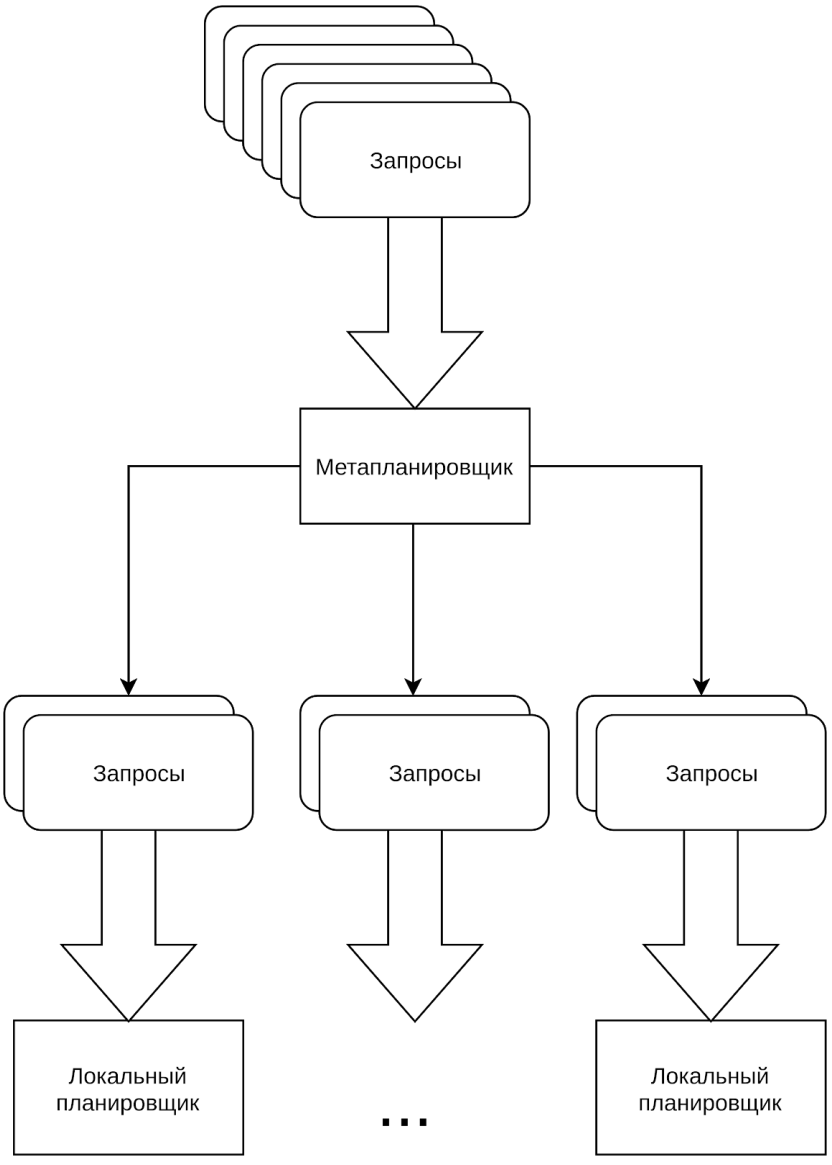

Рисунок 1. Метапланирование над локальными планировщиками

Важно заметить, что планировщики локальных ЦОД между собой никак не взаимодействуют. Логика распределения запросов общего пула реализуется исключительно на уровне метапланировщика.

Метапланировщик в рамках своего алгоритма отправляет поднабор запросов локальному планировщику, получая от последнего обратную связь. Обратная связь в общем случае подразумевает ответ о возможности или невозможности разместить данный поднабор на вычисление. В расширенном случае локальный планировщик сообщает, какие запросы из уже размещенных могут быть сняты с целью разместить более объемный запрос.

Процедура отправки поднабора запросов и получения обратной связи называется итерацией, происходящей в рамках работы алгоритма метапланирования. Число итераций должно ограничиваться изначально. Последнее позволяет производить метапланирование за разумное время.

Цель метапланировщика – распределить запросы наиболее эффективным образом. Эффективность подразумевает соблюдение следующих свойств, в порядке убывания значимости:

1. размещение такого подмножества запросов, которое максимизирует загрузку вычислительных мощностей ЦОД;
2. распределение запросов по локальным ЦОД наиболее равномерным образом;
3. наименьшее число итераций в рамках заданного ограничения.

Вышеназванные требования продиктованы экономическими соображениями. Максимизация загрузки увеличивает доход провайдера ресурсов. Равномерное распределение запросов позволяет сократить издержки, которые при загрузке ЦОД растут нелинейно. Наименьшее число итераций означает меньшие вычислительные и временные затраты на метапланирование.

3.2 Математическая постановка задачи

Математическая постановка задачи метапланирования в ЦОД включает в себя понятия и определения, введенные авторами работы [5] в рамках постановки задачи локального планирования в ЦОД. Приведем введенную постановку задачи ниже.

3.2.1 Задача локального планирования в ЦОД

Модель физических ресурсов ЦОД задается графом $H = (P \cup M \cup K, L)$, где P — множество вычислительных узлов, M — множество хранилищ данных, K — множество коммутационных элементов сети обмена локального ЦОД, L — множество физических каналов передачи данных. На множествах P, M, K и L определены векторные функции скалярного аргумента, задающие соответственно характеристики вычислительных узлов, хранилищ данных, коммутационных элементов и каналов передачи данных:

$ph(p) = (ph_1(p), ph_2(p), \ldots, ph_{n1}(p))$,
$mh(m) = (mh_1(m), mh_2(m), \ldots, mh_{n2}(m))$,
$kh(k) = (kh_1(k), kh_2(k), \ldots, kh_{n3}(k))$,
$lh(l) = (lh_1(l), lh_2(l), \ldots, lh_{n4}(l))$.

Здесь $p \in P$, $m \in M$, $k \in K$, $l \in L$. Компоненты векторной функции могут принимать целочисленные или вещественные значения. Предполагается, что для коммутационных элементов и каналов передачи данных определяются одинаковые характеристики.

Ресурсный запрос задается графом $G = (w \cup S, E)$, где W — множество виртуальных машин, используемых приложениями, S — множество виртуальных хранилищ данных (storage-элементов),

E — множество виртуальных каналов передачи данных между виртуальными машинами и storage-элементами запроса. На множествах W, S и E определены векторные функции скалярного аргумента, задающие характеристики запрашиваемого виртуального элемента (требуемое качество сервиса, SLA):

$wg(w) = (wg_1(w), wg_2(w), \ldots, wg_{n1}(w))$,
$sg(s) = (sg_1(s), sg_2(s), \ldots, sg_{n2}(s))$,

$eg(e) = (eg_1(e), eg_2(e), \ldots, eg_{n4}(e))$.

Здесь $w \in W$, $s \in S$, $e \in E$. Компоненты векторной функции также принимают целочисленные или вещественные значения. Характеристики SLA элемента запроса совпадают с характеристиками соответствующего ему физического ресурса.

Назначением ресурсного запроса называется отображение:

$A : G \to H = \{W \to P, S \to M, E \to \{K, L\}\}$.

Выделяется три типа отношений между характеристиками элементов запросов и соответствующих характеристик физических ресурсов. Через x_i обозначается характеристика i-го элемента запроса, через y_j — соответствующая ей характеристика j-го физического ресурса. Эти отношения могут быть записаны следующим образом:

1. Недопустимость перегрузки емкости физического ресурса:

$$\sum_{i \in R_j} x_i \le y_j,$$

здесь R_j — множество элементов запросов, назначенных на выполнение на физическом ресурсе y_j .

2. Соответствие типа запрашиваемого ресурса типу физического ресурса:

$x_i = y_i$.

3. Наличие требуемых характеристик у физического ресурса:

$x_i \le y_i$.

Отображение A называется *корректным*, если для всех физических ресурсов и всех их характеристик выполняются отношения 1-3.

Остаточным графом доступных ресурсов называется граф H_{res} , для которого переопределены значения функций по характеристикам, которые должны удовлетворять соотношениям:

$$\begin{aligned}
ph_{res}(p) &= ph(p) - \sum_{w \in W_p} wg(w), \\
mh_{res}(m) &= mh(m) - \sum_{s \in S_m} sg(s), \\
kh_{res}(k) &= kh(k) - \sum_{e \in E_k} eg(e), \\
lh_{res}(l) &= lh(l) - \sum_{e \in E_l} eg(e).
\end{aligned}$$

Здесь W_p – множество виртуальных машин, назначенных на выполнение на вычислительном узле p, S_m — множество storage-элементов, размещенных в хранилище данных m, E_k — множество виртуальных каналов, проходящих через коммутационный элемент k, E_l — множество виртуальных каналов, отображенных на физический канал l.

В качестве *исходных данных* задачи назначения ресурсных запросов на физические ресурсы локального ЦОД заданы:

1. множество запросов $Z = \{G_i\}$, поступивших локальному планировщику;
2. остаточный граф доступных ресурсов $H_{res} = (P \cup M \cup K, L)$.

Требуется: из множества Z разместить на выполнение в локальном ЦОД максимальное число запросов, таких, что отображение A является корректным.

Входом алгоритма назначения запросов на физические ресурсы является остаточный граф доступных ресурсов H_{res} и множество ресурсных запросов $\{G_i\}$.

Выходом алгоритма назначения запросов на физические ресурсы является множество назначений ресурсных запросов на физические ресурсы $\{A_i : G_i \rightarrow H, i \in \{1, \ldots, n\}, i = 0, 1, \ldots .$

3.2.2 Задача метапланирования над локальными алгоритмами

Задача метапланирования ставится для множества локальных ЦОД $\{D_i\}$. Описание локального ЦОД может даваться в различных форматах, например, в виде введенного ранее графа остаточных ресурсов:

$D_i = H_{res\ i}$

Иным форматом описания является совокупность остаточных ресурсов ЦОД:

$D_i = \sum y_{ji}$

На вход алгоритму метапланирования, аналогично алгоритму локального планирования, поступает:

1. множество запросов $Z_{meta} = \{G_i\}$ того же вида, что и поступающие локальным планировщикам;
2. описания локальных ЦОД $\{D_i\}$ в выбранном формате.

Требуется разбить множество Z_{meta} на подмножества Z_i, $i \in \{1, \ldots, m\} \cup \{None\}$. где каждое подмножество, кроме Z_{None} , соответствует i-му локальному ЦОД. Совокупность запросов такого подмножества полностью и корректно отображается на ресурсы соответствующего ЦОД. Z_{None} включает в себя неназначенные запросы. При этом максимальное число ресурсов локальных ЦОД должны утилизироваться, то есть

$||Z_{None}|| \rightarrow \min$

Выходом алгоритма является совокупность множеств назначений ресурсных запросов на физические ресурсы локальных ЦОД, то есть для j-го локального ЦОД:

$\{Aj_{ji} : G_{ji} \rightarrow H, i \in \{1, \ldots, n\}, i = 0, 1, \ldots .$

4. Метапланирование вычислений в распределенных системах

4.1 GRID-системы

Примером систем, в которых находит применение метапланирование, являются GRID-системы.

GRID-система представляет из себя физическую и программную инфраструктуру, обеспечивающую доступ к высокопроизводительным вычислительным ресурсам [6]. Задачи на вычисление регистрируются специальным компонентом GRID – Grid Resource Allocation and Management (GRAM). Выделением вычислительных мощностей в GRID заведуют локальные менеджеры, назначающие ресурсы задачам в режиме конкуренции. Последнее осуществляется с целью обеспечения общей эффективности и высокой производительности [7]. Таким образом, GRAM является интегрирующим средством, отвечающим за прием и рассылку задач на свободные вычислительные ресурсы, в то время как непосредственным размещением задач на физических ресурсах заведуют локальные менеджеры.

Глобальный уровень планирования в GRID может предполагать различные стратегии планирования – статическое и динамическое. В рамках статического планирования на выбор ресурса для выполнения задачи влияет информация о состоянии ресурсов на начальный момент времени. При динамическом планировании учитывается появление новых ресурсов, а также изменение состояний заданий, находящихся на исполнении [8].

Цель статического метапланирования – уменьшить время выполнения задач. План определяется до этапа запуска задач и такие характеристики, как время обработки, требования синхронизации, известны заранее [9].

Метапланирование может осуществляться на основе экономических моделей [10]. Владельцы ресурсов (производители) и потенциальные пользователи ресурсов (потребители) преследуют разные экономические цели. Владельцы ресурсов желают увеличить прибыль, в то время как

пользователи пытаются снизить расходы. Таким образом, возникают торги. Данная модель является примером динамического планирования.

При планировании в GRID всегда учитывается непостоянный состав вычислителей, в связи с чем предусмотрены создание контрольных точек и миграция задач.

4.2 Облачные платформы

Исследования в области оптимизации облачных вычислений производятся в контексте двух точек зрения: оптимизация с точки зрения провайдеров ресурсов и оптимизация с точки зрения пользователя [1]. Если снижение затрат провайдера как правило сводится к компактному размещению виртуальных машин на серверах, то пользовательская оптимизация затрат предполагает сравнение вариантов размещения задачи на различных комбинациях виртуальных машин.

Физическая инфраструктура облаков абстрагирована от пользователя и в исследованиях на тему пользовательской оптимизации предполагается неограниченность доступных физических ресурсов. Задавая ограничения на материальные затраты или на время выполнения задачи, пользователь с помощью профилировщика может оптимальным образом распределить элементы задачи по вычислителям, находящимся в том числе на облачных платформах различных провайдеров [1].

Таким образом, возникает уровень метапланирования: элементы задачи исполняются независимо на различных платформах в рамках выбранного пользователем плана. Главная особенность приведенного выше подхода состоит в том, что метапланирование осуществляется полностью на стороне пользователя, а провайдеры ресурсов не имеют никакого представления о целостной задаче.

4.3 Распределенный ЦОД. Непременимость подходов GRID и пользовательского метапланирования

GRID

Реализация метапланирования в распределенных ЦОД отличается от метапланирования в системах GRID за счет ряда особенностей.

Подходы, применяемые при планировании в GRID, учитывают следующие свойства вычислителей:

1. гетерогенность по характеристикам;
2. непостоянный состав вычислителей – новые узлы могут как подключаться к GRID, так и выбывать.

Метапланировщики GRID проектируются с учетом приведенных выше свойств, что означает дополнительные затраты при планировании: учитывается создание контрольных точек, миграция задач в связи с выбыванием вычислителя. Подобные затраты не возникают в распределенном ЦОД.

Пользовательское метапланирование

Пользовательское метапланирование для облачных платформ осуществляется вне учета ограниченности ресурсов последних, так как ресурсные запросы отдельного пользователя почти всегда несопоставимо меньше физических ресурсов в распоряжении провайдера.

В ситуации с распределенным ЦОД совокупность запросов не всегда может быть назначена на исполнение в локальном ЦОД. В подобном случае важно предусматривать возможность обратной связи со стороны локальных планировщиков. Механизмы миграции задач с одной платформы на другую, аналогичные миграции запросов из одного локального ЦОД в другой, также остаются недоступными.

Из вышесказанного следует, что проектирование алгоритма метапланирования в ЦОД должно осуществляться на основе принципов, отличных от принципов GRID и идей, закладываемых в пользовательское метапланирование для облачных платформ.

5. Алгоритм работы метапланировщика

5.1 Идеи и принцип построения

Алгоритм работы метапланировщика, отрабатывая за конечное число шагов, разбивает входной пул запросов на подпулы локальных планировщиков.

На вход алгоритм метапланирования получает ограничение на число итераций обращений к локальным планировщикам, описание локальных ЦОД, которое может даваться с разной степенью детализации, а также множество запросов для разбиения на подпулы.

Описание локальных ЦОД может даваться с разной степенью детализации. В математической постановке задачи метапланирования приведены варианты описания локальных ЦОД, в которых предполагается либо осведомленность метапланировщика о суммарных остаточных ресурсах ЦОД, либо наличие у него детализированной информации о графах остаточных ресурсов локальных центров обработки данных. В зависимости от выбранного варианта описания в алгоритме допускаются различные реализации функции оценки возможности размещения запросов в некотором ЦОД. Необходимо также иметь функцию оценки запроса и функцию, которая позволит пересчитывать оценку описания ЦОД с учетом оценки размещаемого запроса.

Функция оценки описания ЦОД будет обозначаться **d(x)**. Она определена для некоторого описания ЦОД, ее возвращаемое значение в общем случае – набор пар ключ-значение.

Обозначением функции оценки запроса является **g(x)**. Определенная для запроса **x**, функция возвращает набора пар ключ-значение.

Функция **d_decrease(x, y)** принимает в качестве аргументов значения функций **d**, **g** соответственно. Она не определена в случаях, когда размещение запроса с оценкой **y** невозможно в ЦОД с оценкой **x**. В остальных случаях ее значением является пересчитанная оценка **x** для описания ЦОД, то есть набор пар ключ-значение.

Вариативность возникает и на этапе выбора локального ЦОД, подпул которого будет пополняться на очередном шаге. Алгоритм допускает

разные стратегии выбора подпула, две из которых – выбор подпула максимального ЦОД по числу ресурсов и случайный выбор ЦОД – рассматриваются ниже. Будем обозначать функцию выбора очередного подпула **datacenter_choice(x)**. Функция определена для множеств описаний ЦОД. Ее возвращаемое значение – элемент этого множества.

Также существует возможность определять стратегию выбора очередного запроса из множества допустимых к помещению в подпул ЦОД. Аналогично, рассматриваются две стратегии выбора – жадная и рандомизированная. Функцию выбора очередного запроса обозначим **tenant_choice(x)**. Функция определена для множеств запросов. Ее возвращаемое значение – элемент этого множества.

Введенные операции позволяют представить описание алгоритма в общем виде. В последующем будут приведены варианты определений функций и оценок, а также обоснована корректность.

5.2 Описание общего алгоритма

Запуск алгоритма имеет смысл, когда элементы входных данных – множество описаний локальных ЦОД $\{D_i\}$ и множество запросов $\{G_i\}$ – не пусты. Основной алгоритм выглядит следующим образом.

1. Установить значение переменной iter_count, хранящей число проделанных итераций обращения к локальным планировщикам, равным 0.
2. Поставить в соответствие каждому описанию D_i множество rejects_i , положить его равным пустому множеству. Множество будет хранить запросы, которые локальный планировщик i-го ЦОД отказался размещать по результатам некоторой итерации.
3. Поставить в соответствие каждому запросу G_i метку mark_i , положить ее значение неопределенным. Метка сигнализирует о том, что запрос определен в подпул некоторого ЦОД.
4. Поставить в соответствие каждому описанию D_i значение функции оценки d(x), обозначаемое d_i .
5. Поставить в соответствие каждому запросу G_i значение функции оценки g(x), обозначаемое g_i .

6. Поставить в соответствие каждому запросу G_i метку try_mark_i , положить ее значение неопределенным. Метка сигнализирует о том, что запрос является кандидатом в подпул некоторого ЦОД.
7. Поставить в соответствие каждому описанию D_i булеву метку $in\_iteration_i$, положить ее значение равным *True*. Метка сигнализирует о том, что *i*-й ЦОД участвует в процессе назначения запросов на некоторой итерации.
8. Построить множество $\{D_i\}'$, состоящее из элементов множества запросов $\{D_i\}$, соответствующая метка $in\_iteration_i$ которых имеет значение *True*.
9. Если множество $\{D_i\}'$ – пустое, то перейти на шаг 17.
10. Положить *k* равным индексу элемента множества $\{D_i\}$, полученного согласно стратегии выбора datacenter_choice($\{D_i\}'$).
11. Построить множество $\{G_i\}'$, состоящее из элементов множества запросов $\{G_i\}$, не содержащихся в $rejects_k$ и для которых метки $mark_i$, try_mark_i не определены. Исключить из этого множества те запросы G_j, для которых значение $d\_decrease(d_i, g_j)$ является неопределенным.
12. Если множество $\{G_i\}'$ – пустое, то установить $in\_iteration_k$ равным *False* и перейти на шаг 8.
13. Положить *l* равным индексу элемента множества $\{G_i\}$, полученного согласно стратегии выбора tenant_choice($\{G_i\}'$).
14. Положить значение метки try_mark_l равным *k*.
15. Рассчитать новую оценку d_k на основе ее предыдущего значения и оценки g_l при помощи функции d_decrease(x, y).
16. Перейти на шаг 8.
17. Элементы множества запросов $\{G_i\}$, имеющие соответствующую метку try_mark_i равной *j*, формируют число кандидатов в подпул *j*-го локального планировщика. Увеличить значение переменной iter_count на единицу, разослать кандидатов соответствующим планировщикам, получить обратную связь. Обратная связь содержит информацию о размещенных запросах и физических ресурсах, на которых запросы были размещены.
18. Для каждого размещенного запроса из отправленных установить соответствующее значение метки $mark_l$ равным try_mark_l.

19. Для каждого неразмещенного запроса G_i из отправленных добавить его во множество $rejects_j$ соответствующего описания ЦОД.
20. Изменить описания D_i в соответствии с информацией о размещенных запросах и занимаемых ими ресурсах.
21. Если число итераций iter_count меньше заданного ограничения, перейти к шагу 4.
22. Считать запросы, имеющие соответствующую метку $mark_i$ определенной и равной некоторому числу j, приписанными к подпулу локального ЦОД, индекс описания которого во множестве $\{D_i\}$ равен j.

5.3 Описание вариантов ключевых операций

Ключевыми операциями, которые требуют определения, являются следующие:

1. функция оценки описания локального ЦОД **d(x)**;
2. функция оценки запроса **g(x)**;
3. функция **d_decrease(x, y)**, возвращающая новое значение оценки описания ЦОД по старой оценке и по значению функции **g(x)** для запроса, являющегося кандидатом на размещение в данном ЦОД;
4. функция **datacenter_choice(X)**, задающая правило выбора описания ЦОД из множества описаний;
5. функция **tenant_choice(X)**, определяющая стратегию выбора запроса из множества запросов.

5.3.1 Оценка описания локального ЦОД и ее изменение

Не являясь планировщиком, алгоритм метапланирования потенциально не способен иметь достаточные критерии размещения запроса на исполнение в определенном ЦОД. Однако необходимый критерий возможно получить на основании информации, имеющейся у метапланировщика о ЦОД и запросах.

В частности очевидно, что если метапланировщику известен граф остаточных ресурсов ЦОД, то должна быть исключена ситуация, в которой в список кандидатов на размещение попадает запрос,

виртуальных элемент которого превышает по характеристикам любой из имеющихся в ЦОД физических элементов.

Если же алгоритм метапланирования обладает информацией только о суммарных ресурсах остаточного графа, то должна исключаться ситуация суммарного превосходства виртуальных мощностей элементов запроса над суммой физических мощностей.

В связи с вышесказанным требуется механизм получения необходимых критериев размещения на этапе формирования списков кандидатов в подпулы локальных ЦОД. Этот механизм организуется с участием функций **d(x)**, **g(x)** и **d_decrease(x, y)**.

Функция **d(x)** предоставляет оценку некоторого ЦОД, функция **g(x)** – оценку некоторого запроса. Данные оценки поступают на вход **d_decrease(x, y)**. В случае несоблюдения необходимых критериев размещения запроса с оценкой **y** в ЦОД с оценкой **x** данная функция возвращает неопределенное значение. Иначе возвращается значение, которое считается целесообразной оценкой ЦОД при условии, что в нем размещен запрос с оценкой **y**.

Связка функций **d(x)**, **g(x)** и **d_decrease(x, y)** должна обладать определенным свойством, которое мы будем называть свойством уменьшения оценки. Оно заключается в следующем: если запустить итерационный процесс

d_{i+1} = **d_decrease(**d_i **,** y_i **)**,

в котором y_i – оценка произвольного непустого запроса, полученная с помощью **g(x)**, а d_0 – оценка ЦОД, полученная с помощью **d(x)**, то на конечном шаге n значение d_n перестанет быть определено. Данное свойство выражает идею того, что на ресурсах ЦОД возможно разместить только конечное число запросов. Свойство используется в обосновании корректности общего алгоритма метапланирования.

5.3.2 Пример оценки в случае осведомленности метапланировщика о графах остаточных ресурсов ЦОД

Рассмотрим пример связки функций **d(x)**, **g(x)** и **d_decrease(x, y)**, подходящий в случае, когда описанием ЦОД является граф остаточных ресурсов.

Пусть D – граф остаточных ресурсов некоторого ЦОД, тогда значением **d**(D) будет являться множество пар ключ-значение, в которых ключ состоит из типа ресурса и его уникального идентификатора, а значением является набор характеристик.

Предположим, в ЦОД имеется 3 ресурса – сервер “computer1” (2 ядра, 1024 RAM), сервер “computer2” (1 ядро, 512 RAM) и хранилище данных “controller1” (5 Гб). Тогда значением функции **d(x)** будет {(server, computer1):(2, 1024), (server, computer2):(1, 512), (storage, controller1):(5)}

Функция **g(x)** приписывает оценки запросам G аналогичным образом, оперируя типами и характеристиками виртуальных элементов запроса.

Если запрос включает в себя 2 виртуальных ресурса, виртуальную машину “vm1” (1 ядро, 512 RAM) и хранилище “st1” (1 Гб), то значением **g(x)** является {(vm, vm1):(1, 512), (st, st1):(1)}.

Функция **d_decrease(x, y)** возвращает новую оценку для описания ЦОД D, вычитая из характеристик физических ресурсов соответствующие характеристики виртуальных. Вычитание производится из наименьшего физического ресурса из числа доступных для этого действия, то есть при условии, что после вычитания все значения физических ресурсов остаются неотрицательными. Если произвести вычитание, сохранив неотрицательность, не удается ни из одного из ресурсов, возвращаемое значение считается неопределенным.

Например, для рассмотренных выше оценок

d_decrease(

{(server, computer1):(2, 1024), (server, computer2):(1, 512), (storage, controller1):(5)}**,**

{(vm, vm1):(1, 512), (st, st1):(1)}

)

вернет значение {(server, computer1):(2, 1024), (server, computer2):(0, 0), (storage, controller1):(4)}.

При этом значение

d_decrease(

{(server, computer1):(2, 1024), (server, computer2):(1, 512), (storage, controller1):(5)}**,**

{(vm, vm1):(1, 1025), (st, st1):(1)}

)

было бы неопределенным, что говорит о невозможности разместить виртуальную машину “vm1” ни на одном из серверов ЦОД.

Очевидно, что определенная подобным образом функция **d_decrease(x, y)** удовлетворяет свойством уменьшения оценки в итерационном процессе.

5.3.3 Пример оценки в случае осведомленности метапланировщика о суммарных остаточных ресурсах ЦОД

Являясь более простым вариантом описания, сумма остаточных ресурсов определяет меньшие возможности задания необходимых условий размещения запросов.

Определим **d(x)** как множество пар ключ-значение, в которых ключ – тип ресурса, значение – сумма ресурсов данного типа в ЦОД.

Как и в предыдущем случае, предположим, в ЦОД имеется 3 ресурса – сервер “computer1” (2 ядра, 1024 RAM), сервер “computer2” (1 ядро, 512 RAM) и хранилище данных “controller1” (5 Гб). Тогда значение функции **d(x)** – это {server:(3, 1536), storage:(5)}.

Функция **g(x)** оценки запроса G аналогичным образом суммирует характеристики виртуальных элементов запроса.

Если запрос включает в себя 3 виртуальных ресурса, виртуальную машину “vm1” (1 ядро, 512 RAM), хранилище “st1” (1 Гб) и хранилище “st2” (2 Гб), то значением **g(x)** является {vm:(1, 512), st:(3)}.

Функция **d_decrease(x, y)** возвращает новую оценку, вычитая из сумм физических характеристик ЦОД суммы виртуальных характеристик элементов запроса. Если произвести вычитание по какой-либо из пар, сохранив неотрицательность, не удается, возвращаемое значение считается неопределенным.

Например, для рассмотренных выше оценок

d_decrease(
{server:(3, 1536), storage:(5)}**,**
{vm:(1, 512), st:(3)}
)

вернет значение {server:(2, 1024), storage:(2)}.

Аналогично заметим, что значение

d_decrease(

{server:(3, 1536), storage:(5)},
{vm:(4, 512), st:(3)}
)

не определено, то есть необходимое условие планирования не выполнено.

Примером ситуации, когда необходимое условие выполнено, однако планирование не будет успешным, является рассмотренная структура ЦОД и запрос, состоящий из виртуальной машины с требованием выделения трех ядер. Таким образом, суммарная оценка в ряде случаев уступает детализированной.

5.3.4 Функция выбора ЦОД из множества

На шаге 10 основного алгоритма предполагается выбор ЦОД из числа доступных для размещения виртуальных ресурсов с помощью функции **datacenter_choice(X)**. Подобный выбор определяет стратегию загрузки локальных ЦОД, и в качестве примера можно рассмотреть две стратегии.

Рандомизированный выбор предполагает случайное определение ЦОД. Преимуществом такого выбора является равномерность загрузки запросами различных центров обработки данных.

Жадный выбор отдает предпочтение тому ЦОД, оценка которого достигает максимального значения по множеству доступных для выбора ЦОД. Такой оценкой может служить кортеж (суммарное число ядер, суммарное число оперативной памяти, суммарное число памяти хранилищ данных), а сравнение производится поэлементно слева направо. В данном случае предполагается достигнуть равномерного распределения свободных ресурсов по ЦОД.

5.3.5 Функция выбора запроса из множества

Выбор запроса из множества удовлетворяющих необходимому условию размещения производится на шаге 13 основного алгоритма. Аналогично выбору ЦОД, приведем две возможные стратегии работы функции **tenant_choice(X)**.

Рандомизированный выбор, при котором запрос из множества выбирается случайно, предполагает равномерность распределения “тяжелых” и “легких” запросов по различным ЦОД. Эта стратегия представляется эффективной для ситуации, в которой все ЦОД похожи по структуре, и утилизация ресурсов таким образом будет происходить равномерно.

Жадный выбор отдает первенство при метапланировании более тяжелым запросам. Преимуществом является потенциальная возможность избежать фрагментации физических ресурсов. Однако стоит учитывать, что, например, для случая осведомленности метапланировщика лишь о суммарных остаточных ресурсах ЦОД, попытка назначать “тяжелые” запросы в первую очередь обернется большим числом неэффективных итераций обращения к локальным планировщикам.

Таким образом, разнообразие вариантов реализации ключевых операций алгоритма метапланирования позволяет подстраивать его под различные нужды, при этом варьируя такими характеристиками, как формат входных данных, сложность реализации, ограничение на число итераций.

5.4 Обоснование корректности алгоритма

Корректность работы алгоритма метапланирования требует соблюдения следующих свойств:

1. завершаемость;
2. непересечение подпулов локальных ЦОД.

Исследуем соблюдение данных свойств ниже.

5.4.1 Завершаемость

Общий алгоритм содержит два цикла – внешний (шаги 4-21) и внутренний (шаги 8-16).

Внешний цикл общего алгоритма всегда имеет конечное число итераций, поскольку шагу 21, где значение переменной iter_count сравнивается с константой и принимается решение о новой итерации,

всегда предшествует шаг 17, на котором значение iter_count увеличивается.

Выход из внутреннего цикла происходит на шаге 9. Условием выхода является отсутствие элементов во множестве $\{D_i\}'$, что достигается в том случае, когда каждому элементу $\{D_i\}$ поставлено значение метки in_iteration_i, равное *False*. Значение метки обращается в *False* для некоторого D_k тогда и только тогда, когда множество $\{G_i\}'$, соответствующее D_k – пустое.

Докажем, что для произвольного D_k в процессе работы внутреннего цикла соответствующее множество станет пустым.

На очередной итерации множество $\{G_i\}'$ не увеличивается. Данный факт имеет место потому, что множества rejects_k , метки mark_i , не изменяются в процессе работы цикла, а try_mark_i изменяются только на определенные значения.

Исключение элемента G_j из множества в процессе итераций цикла происходит, когда d_decrease(d_k, g_j) принимает неопределенное значение. Заметим, что оценка d_k переписывается каждую итерацию, меняя значение на d_decrease(d_k, g_j), то есть имеет место итерационный процесс, расмотренный выше. Для предложенных вариантов определения функция **d_decrease(x, y)** уменьшает оценку для непустых запросов, которыми и оперирует алгоритм метапланирования. То есть на одной из очередных итерации d_k станет неопределенным.

Таким образом, произвольный G_j окажется удален из $\{G_i\}'$ за конечное число шагов. $\{G_i\}'$ станет пустым, вследствие чего произойдет выход из цикла.

5.4.2 Непересечение подпулов локальных ЦОД

Попадание *i*-го запроса из $\{G_i\}$ в подпул локального ЦОД с описанием D_j из $\{D_j\}$ происходит если и только если mark_i имеет значение *j*.

Значение метки mark_i всегда устанавливается по метке try_mark_i. Последняя, в свою очередь, может получить значение лишь на шаге 14. Это происходит исключительно для элементов множества $\{G_i\}'$, которое формируется на шаге 11 и не включает в себя элементы с уже определенным значением mark_i.

Очевидно, что mark_i для произвольного запроса из $\{G_i\}$ не может быть установлена в определенное значение более одного раза.

5.5 Расширение алгоритма на случай миграции запросов между ЦОД

Частой является ситуация, в которой размещение “тяжелого” запроса в ЦОД невозможно, однако такая возможность может появиться в случае миграции более “легких” запросов за пределы этого ЦОД. Если возлагать на локальный планировщик функцию обнаружения и уведомления о таких ситуациях алгоритм метапланирования, то допустимо организовывать миграцию запросов между локальными ЦОД.

Реализация подобной идеи на стороне метапланировщика означает учет им целесообразности миграции. В данной работе не исследуются способы оценки целесообразности, однако интуитивно понятно, что, например, снятие запроса, состоящего из виртуальной машины с одним ядром, в пользу запроса, включающего в себя виртуальную машину с четырьмя ядрами, может быть целесообразно.

Тем не менее, миграция порождает новый запрос для метапланирования, который должен быть обслужен в первоочередном порядке и в рамках существующего ограничения на максимальное число итераций. Для нового запроса, в свою очередь, могут возникнуть новые возможности миграций. Предложенный алгоритм для подобной рекурсивной цепочки в общем случае может исчерпать ограничение на число итераций, исследуя лишь возможность миграции и не производя разбиения первоначального пула запросов на подпулы.

В связи с вышесказанным способ оценки целесообразности требует отдельного исследования.

Предположим, этот способ был некоторым образом выбран, приведем модификацию общего алгоритма для случая сообщения о возможности миграции.

Модифицируем шаг 17 основного алгоритма.

17*. Элементы множества запросов $\{G_i\}$, имеющие соответствующую метку try_mark_i равной j, формируют число кандидатов в подпул j-го локального планировщика. Увеличить значение переменной iter_count на единицу, разослать кандидатов соответствующим планировщикам, получить обратную связь. Обратная связь содержит информацию о размещенных запросах и физических ресурсах, на которых запросы были размещены, а также о возможностях миграций. Выбрать те миграции,

которые оценены как целесообразные. Рекурсивно запустить основной алгоритм для каждой из миграций, при этом установив число ограничений на итерации равным разности ограничения для внешнего вызова алоритма и текущего значение iter_count этого вызова. После каждого рекурсивного вызова прибавлять число выполненных им итераций к iter_count внешнего вызова. В случае достижения ограничения на число итераций выполнить шаг 18, затем 22, затем завершить работу внешнего вызова.

В данном случае требование корректности алгоритма сохраняется: ограничение на число итераций убывает для рекурсивных вызовов и алгоритм всегда завершается.

6. Программная реализация алгоритмов метапланирования

6.1 Программная архитектура

Программная реализация алгоритма метапланирования предполагает запуск локальных планировщиков и взаимодействие с форматом их входных-выходных данных, генерацию примеров ЦОД и запросов, а также предоставление различных реализаций для ключевых операций, в соответствии с рассмотренными вариантами.

В связи с этим можно выделить следующие программные модули:

1. Генератор данных
2. Ввод-вывод xml
3. Вызов локального алгоритма планирования
4. Варианты реализации связок оценочных функций
5. Основной алгоритм со стратегиями выбора запросов и ЦОД
6. Экспериментальные исследования

Генератор данных

Генератор данных содержит функции, позволяющие получать элементы ЦОД и запросов для экспериментальных исследований. Данные генерируются с учетом некоторых эвристик, таких как стандартные характеристики серверов, виртуальных машин, а также предполагаемое распределение последних по классам. Например, более “легкие” виртуальные машины ожидается встречать на практике чаще.

Ввод-вывод xml

Данный модуль подготавливает перевод внутреннего представления запросов и ЦОД в формат xml, принимаемый алгоритмом локального планирования.

Вызов локального алгоритма планирования

Реализация вызова локального алгоритма планирования с определенными входными данными и считывание результатов его работы содержится в данном модуле.

Варианты реализации связок оценочных функций

В модуле приведен интерфейс, а также его реализация на случай оценочных функций для различных вариантов описаний ЦОД: через граф остаточных ресурсов и через суммарное количество ресурсов.

Основной алгоритм со стратегиями выбора запросов и ЦОД

Основной алгоритм, принимающий на вход в том числе стратегии выбора ЦОД и запроса из числа возможных. Для обоих случаев реализованы жадная и рандомизированная стратегия. Алгоритм возвращает суммарное число размещенных запросов, число итераций и количество утилизированных ядер.

Экспериментальные исследования

Модуль, в котором сравниваются различные варианты планирования.

Программа написана на языке Python версии 3 с использованием Jupyter Notebook. Для работы с xml задействована библиотека BeautifulSoup, данные анализируются с применением библиотек numpy и matplotlib.

Исходный код доступен по ссылке [11].

6.2 Классы и структуры данных

Ниже приведены структуры данных программы:

запрос

{

'name': ***название запроса***,

'links': ***список соединений между портами элементов***,

'sts': ***список элементов-хранилищ данных***,

'vms': ***список элементов-виртуальных машин***

};

хранилище данных

{

'name': ***название хранилища***,

'size': ***размер хранилища***,

'ports': ***список портов***

};

виртуальная машина

{

'name': ***название виртуальной машины***,

'VCPUs': ***число ядер***,

'RAM': ***размер оперативной памяти***,

'ports': ***список портов***

};

ЦОД

{

'name': ***название ЦОД***,

'links': ***список соединений между портами элементов***,

'storages': ***список элементов-хранилищ данных***,

'servers': ***список элементов-серверов,***

'netelements': ***список коммутаторов***,

};

сервер

{

'name': ***название сервера***,

'VCPUs': ***число ядер***,

'RAM': ***размер оперативной памяти***,
'ports': ***список портов***
};
коммутатор
{
'name': ***название коммутатора***,
'ports': ***список портов***
}.

Классами в программной реализации являются следующие.

Tenant

Объекты класса агрегируют запрос, соответствующий ему в рамках алгоритма try_mark, mark, rejects, значение оценки g.

DC

Объекты этого класса содержат данные ЦОД и значение его оценки d.

Evaluator_base

Базовый класс для связок функций расчета оценкок ЦОД, запроса, понижения оценки ЦОД.

Evaluator_summed

Реализация функций оценок для случая осведомленности метапланировщика о суммарных остаточных ресурсах ЦОД.

Evaluator_details

Реализация функций оценок для случая осведомленности метапланировщика о графах остаточных ресурсов ЦОД.

7. Исследование свойств алгоритма метапланирования

7.1 Цель и методика исследования

Основной целью исследования является сравнение эффективности работы алгоритма для случаев различной реализации его ключевых операций.

Также интерес представляет вариация формата входных данных, которая в случае с ЦОД может быть как детализированной, так и нет.

Практический смысл имеет и выявление среднего числа итераций метапланирования, изучение поведения метапланировщика при намеренном ограничении максимального числа итераций относительно небольшим числом.

Методика исследования предполагает запуск алгоритма как на специально подготовленных данных, так и на сгенерированных. В первом случае проверяются предложенные гипотезы, во втором – изучается поведение на статистически значимых выборках.

При исследовании сравниваются такие показатели метапланирования, как число размещенных запросов, количество итераций алгоритма, число утилизированных процессорных ядер. Первый показатель дает возможность оценить, насколько хорошо алгоритм размещает запросы в среднем, второй показатель показывает эффективность производимых итераций, третье значение позволяет понять, как обстоят дела с размещением “тяжелых” запросов.

При исследовании на генерируемых данных алгоритм метапланирования запускается 100 раз, после чего результаты запусков агрегируются: суммируется число размещенных запросов, число проделанных итераций, число утилизированных ядер процессора.

7.2 Результаты экспериментов

В результате проведения экспериментов не удалось установить вариант алгоритма, который превосходил бы все остальные по эффективности в контексте трех изучаемых параметров.

В процессе проверки гипотез и работы с генерируемыми данными подтвержден или выявлен ряд особенностей. Ниже приведены некоторые из них.

1. Алгоритм в случае детализированного описания ЦОД осуществляет метапланирование за меньшее число итераций, почти всегда хватает одной итерации. Для описания в виде совокупности ресурсов число итераций иногда в среднем может достигать 3-х. Очевидно, сказывается меньшая осведомленность алгоритма метапланирования.
2. Большое число итераций для алгоритма с суммарной оценкой ресурсов появляется в случае жадных стратегий выбора. Это означает, что ограничение в таких случаях числа итераций меньшим значением, например, 2, сильно скажется на эффективности алгоритма. Таким образом, в случае использования представления в виде суммарной оценки ресурсов, необходимо обеспечивать достаточное число итераций, иначе алгоритм деградирует.
3. Применение в обоих случаях рандомизированной стратегии выбора элемента увеличивает число обслуженных запросов в 1.6 раза относительно обеих жадных стратегий. Однако число утилизированных ядер снижается на 6%. Число итераций в случае детализированного представления растет почти незначительно.
4. Рандомизация только выбора запроса увеличивает число обслуженных запросов в 1.5 раза, рандомизация только выбора ЦОД почти не увеличивает. Очевидно, первое ведет к обслуживанию запросов, которые в среднем легче.

Таким образом, выявлены принципы, позволяющие подстраивать алгоритм метапланирования под определенные стратегии и нужды.

8. Заключение

Курсовая работа посвящена метапланированию в распределенном ЦОД. К основным результатам относится следующее.

1. Произведено знакомство с предметной областью: изучены особенности планирования в центрах обработки данных.
2. Рассмотрены дисциплины метапланирования в распределенных системах GRID и облачных платформах.
3. Предложен алгоритм работы метапланировщика. Ключевые операции описаны в общем виде, а также имеют варианты конкретизации.
4. Алгоритм реализован программно в нескольких вариантах конкретизации ключевых операций.
5. Произведено исследование работы алгоритма. Приведены результаты экспериментов.

9. Список источников

1. Long Thai, Blesson Varghese, Adam Barker. A Survey and Taxonomy of Resource Optimisation for Executing Bag-of-Task Applications on Public Clouds, 2017
2. Зотов И. А., Костенко В. А. Алгоритм распределения ресурсов в центрах обработки данных с единым планировщиком для различных типов ресурсов, 2015
3. Kashif Bilal, Saif Ur Rehman Malik, Osman Khalid, Abdul Hameed,Enrique Alvarez,Vidura Wijaysekara, Rizwana Irfan, Sarjan Shrestha, Debjyoti Dwivedy, Mazhar Ali,Usman Shahid Khan, Assad Abbas, Nauman Jalil, Samee U. Khan. A taxonomy and survey on Green Data Center Networks, 2012
4. Marco A. S. Netto, Rajkumar Buyya. Coordinated Rescheduling of Bag-of-Tasks for Executions on Multiple Resource Providers, 2010
5. Вдовин П. М., Костенко В. А. Алгоритм распределения ресурсов в центрах обработки данных с раздельными планировщиками для различных типов ресурсов, 2014
6. Ian Foster, Computational Grids, 1998
7. Прихожий А. А., Фролов О. М. Исследование планировщиков задач в GRID, 2015
8. Дема К.В., Волк М.А., Филимончук М.А. Анализ архитектур и характеристик современных планировщиков заданий в GRID-системах, 2011
9. Голубев И. А. Планирование задач в распределённых вычислительных системах на основе метаданных, 2014
10. Rajkumar Buyya, David Abramson, And Srikumar Venugopal, The Grid Economy, 2005
11. Программная реализация метапланировщика https://github.com/Vasilesk/c5work

10. Приложения

Приложение 1. Исходные данные для экспериментов

В качестве исходных данных случайно генерируется набор из 3-х ЦОД и 30-ти запросов [11].

Пример сгенерированных данных для ЦОД

```
[{'links': [],
  'name': 'dc1',
  'netelements': [],
  'servers': [{'RAM': 16384, 'VCPUs': 4, 'name': 'computer1', 'ports': []},
   {'RAM': 8192, 'VCPUs': 2, 'name': 'computer2', 'ports': []}],
  'storages': [{'name': 'storage1', 'ports': [], 'size': 5},
   {'name': 'storage2', 'ports': [], 'size': 100}]},
 {'links': [],
  'name': 'dc2',
  'netelements': [],
  'servers': [{'RAM': 32768, 'VCPUs': 6, 'name': 'computer1', 'ports': []},
   {'RAM': 32768, 'VCPUs': 6, 'name': 'computer2', 'ports': []},
   {'RAM': 32768, 'VCPUs': 6, 'name': 'computer3', 'ports': []},
   {'RAM': 16384, 'VCPUs': 4, 'name': 'computer4', 'ports': []}],
  'storages': [{'name': 'storage1', 'ports': [], 'size': 250},
   {'name': 'storage2', 'ports': [], 'size': 5}]},
 {'links': [],
  'name': 'dc3',
  'netelements': [],
  'servers': [{'RAM': 8192, 'VCPUs': 2, 'name': 'computer1', 'ports': []},
   {'RAM': 8192, 'VCPUs': 2, 'name': 'computer2', 'ports': []},
   {'RAM': 32768, 'VCPUs': 6, 'name': 'computer3', 'ports': []},
   {'RAM': 16384, 'VCPUs': 4, 'name': 'computer4', 'ports': []}],
  'storages': [{'name': 'storage1', 'ports': [], 'size': 25},
   {'name': 'storage2', 'ports': [], 'size': 500}]}]
```

Пример сгенерированных данных для запросов (первые 5 из 30)

```
[{'links': [],
  'name': 'request1',
  'sts': [{'name': 'storage1', 'ports': [], 'size': 10}],
  'vms': [{'RAM': 2048, 'VCPUs': 2, 'name': 'vm1', 'ports': []},
   {'RAM': 1024, 'VCPUs': 1, 'name': 'vm2', 'ports': []},
   {'RAM': 2048, 'VCPUs': 3, 'name': 'vm3', 'ports': []},
   {'RAM': 8192, 'VCPUs': 4, 'name': 'vm4', 'ports': []},
   {'RAM': 2048, 'VCPUs': 2, 'name': 'vm5', 'ports': []}]},
 {'links': [],
  'name': 'request2',
  'sts': [{'name': 'storage1', 'ports': [], 'size': 50}],
  'vms': [{'RAM': 2048, 'VCPUs': 2, 'name': 'vm1', 'ports': []},
   {'RAM': 8192, 'VCPUs': 4, 'name': 'vm2', 'ports': []}]},
 {'links': [],
```

```
 'name': 'request3',
 'sts': [{'name': 'storage1', 'ports': [], 'size': 1},
  {'name': 'storage2', 'ports': [], 'size': 10}],
 'vms': [{'RAM': 1024, 'VCPUs': 1, 'name': 'vm1', 'ports': []}]},
{'links': [],
 'name': 'request4',
 'sts': [{'name': 'storage1', 'ports': [], 'size': 5},
  {'name': 'storage2', 'ports': [], 'size': 1}],
 'vms': [{'RAM': 8192, 'VCPUs': 4, 'name': 'vm1', 'ports': []},
  {'RAM': 2048, 'VCPUs': 3, 'name': 'vm2', 'ports': []},
  {'RAM': 2048, 'VCPUs': 3, 'name': 'vm3', 'ports': []},
  {'RAM': 2048, 'VCPUs': 3, 'name': 'vm4', 'ports': []}]},
{'links': [],
 'name': 'request5',
 'sts': [{'name': 'storage1', 'ports': [], 'size': 25}],
 'vms': [{'RAM': 2048, 'VCPUs': 3, 'name': 'vm1', 'ports': []},
  {'RAM': 8192, 'VCPUs': 4, 'name': 'vm2', 'ports': []}]},
]
```

Приложение 2. Результаты экспериментов

Интегральные результаты по 100 экземплярам случайных исходных данных приводятся ниже.

Осведомленность о графе остаточных ресурсов

1. стратегия выбора ЦОД: random, стратегия выбора запроса: random, запросов 766.0, ядер 3426.0, итераций 102.0
2. стратегия выбора ЦОД: random, стратегия выбора запроса: max, запросов 488.0, ядер 3326.0, итераций 101.0
3. стратегия выбора ЦОД: max, стратегия выбора запроса: random, запросов 753.0, ядер 3241.0, итераций 107.0
4. стратегия выбора ЦОД: max, стратегия выбора запроса: max, запросов 478.0, ядер 3323.0, итераций 104.0

Осведомленность о суммарных ресурсах ЦОД

1. стратегия выбора ЦОД: random, стратегия выбора запроса: random, запросов 732.0, ядер 3234.0, итераций 160.0
2. стратегия выбора ЦОД: random, стратегия выбора запроса: max, запросов 477.0, ядер 3369.0, итераций 311.0
3. стратегия выбора ЦОД: max, стратегия выбора запроса: random, запросов 763.0, ядер 3336.0, итераций 159.0
4. стратегия выбора ЦОД: max, стратегия выбора запроса: max, запросов 470.0, ядер 3455.0, итераций 274.0

По времени выполнения кода второй вариант в разы уступает первому, что обусловлено большим количеством вызовов локального планировщика.