ИЗВЕСТИЯ РАН. ТЕОРИЯ И СИСТЕМЫ УПРАВЛЕНИЯ, 2014, № 5, с. 71–83

КОМПЬЮТЕРНЫЕ МЕТОДЫ

УДК 004.031.43

СРАВНЕНИЕ РАЗЛИЧНЫХ ПОДХОДОВ К РАСПРЕДЕЛЕНИЮ РЕСУРСОВ В ЦЕНТРАХ ОБРАБОТКИ ДАННЫХ

© 2014 г. П. М. Вдовин, И. А. Зотов, В. А. Костенко, А. В. Плакунов, Р. Л. Смелянский

Москва, МГУ

Поступила в редакцию 24.01.14 г.

Выбраны абстракции для описания ресурсных запросов и физических ресурсов центров обработки данных. Разработана математическая модель центра обработки данных, которая позволяет описывать широкий класс их архитектур. В рамках этой модели сформулирована математическая постановка задачи распределения ресурсов, которая допускает миграцию виртуальных машин и репликацию элементов хранения данных. Приведены результаты сравнения алгоритмов распределения ресурсов для центров обработки данных с единым планировщиком для всех типов ресурсов и для центров обработки данных с раздельными планировщиками для каждого типа ресурсов, а также результаты сравнения разработанных алгоритмов с аналогичными алгоритмами из платформы OpenStack.

DOI: 10.7868/S0002338814040143

Введение. Качество работы планировщиков ресурсов в центрах обработки данных (ЦОД) определяет загрузку ресурсов ЦОД и возможность гарантированного обеспечения требуемого качество сервиса (SLA). Алгоритмы, используемые в платфороме OpenStack [1]: “назначение запроса на первый подходящий физический ресурс” или “выбор случайным образом физического ресурса из множества подходящих ресурсов” приемлемы по качеству получаемых отображений запросов на физические ресурсы ЦОД при их загрузке до 50%. При загрузке физических ресурсов больше 50% качество отображений, получаемых этими алгоритмами, значительно ухудшается. Это показано в разд. 3 данной статьи. Для отображения ресурсных запросов на физические ресурсы ЦОД требуется решить три взаимосвязанных *NP*-трудных задачи:

отображение виртуальных машин на вычислительные ресурсы ЦОД;

отображение элементов хранения данных (storage-элементов) на хранилища данных ЦОД;

отображение виртуальных каналов на физические ресурсы сети обмена ЦОД.

В случае высокой загрузки физических ресурсов ЦОД алгоритмы из OpenStack неприемлемы по качеству получаемых отображений ресурсных запросов на физические ресурсы ЦОД, а точные алгоритмы неприемлемы из-за высокой вычислительной сложности. Это обуславливает актуальность разработки эвристических алгоритмов, которые позволяют обеспечить нужный баланс между точностью и вычислительной сложностью.

Для обеспечения высокой загрузки в режиме использования ЦОД Infrastructure-as-a-Service (IaaS) [2] требуется решение следующих задач.

1. Устранение сегментации физических ресурсов за счет миграции виртуальных ресурсов в ЦОД.

2. Устранение перегрузки входящих в физическое хранилище данных каналов обмена. Данная задача возникает, когда имеются виртуальные хранилища данных с высокой интенсивностью считывания и низкой интенсивностью записи. Она может быть решена, если допускается репликация хранилищ данных. В этом случае для обеспечения консистентности данных не требуется канал с высокой пропускной способностью, и половина приложений может работать с виртуальным хранилищем данных, а другая – с его копией, которая располагается в другом физическом хранилище данных.

3. Возможность задания SLA для всех типов ресурсов ЦОД. Для этого вычислительные ресурсы, хранилища данных, сетевые ресурсы должны рассматриваться как планируемые типы ресурсов и их планирование должно проходить согласовано в смысле соблюдения соглашений о качестве сервиса.

В таблице приведено сравнение известных алгоритмов [3–16] с точки зрения возможности решать задачи 1–3.

Сравнительная характеристика алгоритмов

Рассмотренные работы	Вычислительные ресурсы как тип ресурсов	Хранилища данных как тип ресурсов	Сетевые ресурсы как тип ресурсов	Возможности		
				отображение виртуальных каналов на физические	репликация	миграция
Application placement on a cluster of servers [3]	+	–	–	–	–	–
Cloud Storage and Online Bin Packing [4]	–	+	–	–	–	–
Efficient Resource Scheduling in Data Centers using MRIS [5]	+	+	–	–	–	–
Quantifying Load Imbalance on Virtualized EnterpriseServers [6]	+	–	–	–	–	+
On Theory of VM Placement: Anomalies in Existing Methodologies and Their Mitigation Using a Novel Vector Based Approach [7]	+	–	–	–	–	+
Optimal Mapping of Virtual Networks with Hidden Hops [8]	–	–	–	+	–	–
Rethinking Virtual Network Embedding: Substrate Support for Path Splitting and Migration [9]	+	–	–	+	–	+
A Virtual Network Mapping Algorithm based on Subgraph Isomorphism [10]	+	–	–	+	–	–
Algorithms for Assigning Substrate Network Resources to Virtualized network Resources [11]	+	–	–	–	–	–
Virtual network embedding with coordinated node and link mapping [12]	+	–	–	+	–	–
Virtual Network Embedding Through Topology-Aware Node Ranking [13]	+	–	–	+	–	–
Coupled placement in modern data centers [14]	+	+	+	±*	–	–
Server-storage virtualization: integration and load balancing in data centers [15]	+	+	+	±*	–	+
Joint VM placement and routing for data center traffic engineering [16]	+	–	+	+	–	+

* Алгоритмы применимы лишь для ЦОД, имеющих иерархическую (древовидную) топологию.

Как видно из таблицы, ни один из алгоритмов не позволяет решать все вышеперечисленные задачи в совокупности.

В данной работе математически сформулирована задача отображений ресурсных запросов на физические ресурсы ЦОД и предложены алгоритмы ее решения, которые позволяют решать все вышеперечисленные задачи в совокупности для модели обслуживания IaaS.

1. Математическая постановка задачи распределения ресурсов ЦОД. *Модель физических ресурсов ЦОД* будем задавать графом $H = (P \cup M \cup K, L)$, где P – множество вычислительных узлов, M – множество хранилищ данных, K – множество коммутационных элементов сети обмена ЦОД, L – множество физических каналов передачи данных.

На множестве P определены функции $vh(p)$ и $qh(p)$, задающие производительность вычислительного узла (операций в секунду) и имеющийся объем всей памяти вычислительного узла (байт).

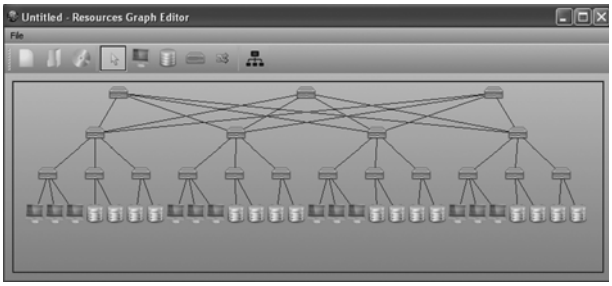

Рис. 1. Стандартная топология ЦОД fat-tree

На множестве M определены функции $uh(m)$ и $type(m)$, задающие объем всей памяти хранилища данных (байт) и его тип.

На множестве K определена функция $th(k)$, задающая пропускную способность коммутационного элемента (байт/с), а на множестве L – функция $rh(l)$, задающая номинальную пропускную способность канала передачи данных (байт/с).

Функции $vh(p)$, $qh(p)$, $uh(m)$, $type(m)$, $th(k)$, $rh(l)$ формируют разметку графа H.

Пример стандартной топологии ЦОД fattree [17] приведен на рис. 1: каждый корневой коммутатор соединен со всеми коммутаторами первого уровня; каждый коммутатор первого уровня соединен с несколькими коммутаторами второго уровня; каждый коммутатор второго уровня объединяет либо несколько вычислительных узлов, либо несколько хранилищ данных.

Ресурсный запрос будем задавать графом $G = (W \cup S, E)$, где W – множество виртуальных машин, используемых приложениями, S – множество storage-элементов, E – множество виртуальных каналов передачи данных между виртуальными машинами и storage-элементами запроса.

На множестве W определены функции $v(w)$ и $q(w)$, задающие требуемую приложениями производительность виртуальных машин (операций в секунду) и объем памяти.

На множестве S определены функции $u(s)$ и $type(s)$, задающие требуемый объем памяти (байт) и тип storage-элемента.

На множестве E определена функция $r(e)$, задающая требуемую пропускную способность виртуального канала (байт/с).

Функции $v(w)$, $q(w)$, $u(s)$, $type(s)$, $r(e)$ формируют разметку графа G.

Ресурсные запросы могут быть двух типов:

слабосвязный: $[(\{W_i\}_{i=1}^{N}, \{S_j\}_{j=1}^{K}),\ E = \varnothing]$;

сильносвязный: $[(\{W_i\}_{i=1}^{N}, \{S_j\}_{j=1}^{K}),\ E = \{(W \subset \{W_i\}_{i=1}^{N}, S \subset \{S_j\}_{j=1}^{K})\}]$.

Назначением ресурсного запроса будем называть отображение $A : G \to H = \{W \to P,\ S \to M,\ E \to \{K, L\}\}$, удовлетворяющее следующим ограничениям.

1. Виртуальная машина w может быть назначена на выполнение на вычислительном узле p, если верны условия

$$\sum_{w \in W_p} v(w) \le vh(p) \quad \text{и} \quad \sum_{w \in W_p} q(w) \le qh(p).$$

Здесь W_p – множество виртуальных машин, назначенных на выполнение на вычислительном узле p. Другими словами, производительность вычислительного узла должна быть не меньше требуемой суммарной производительности назначенных на него виртуальных машин и объем памяти вычислительного узла должен быть не меньше требуемой суммарной памяти назначенных на него виртуальных машин. Планирование выполнения виртуальных машин на вычислительном узле осуществляется локальным планировщиком вычислительного узла. Локальный планировщик должен гарантированно обеспечивать для каждой виртуальной машины требуемую производительность.

2. Виртуальный канал e может быть отображен на физический канал l, если верно условие

$$\sum_{e \in E_l} r(e) \le rh(l).$$

Здесь E_l – множество виртуальных каналов, отображенных на физический канал l.

3. Виртуальный канал e может проходить через коммутационный элемент k, если верно условие

$$\sum_{e \in E_k} r(e) \le \tau h(k).$$

Здесь E_k – множество виртуальных каналов, проходящих через коммутационный элемент k.

4. Storage-элемент s может быть размещен в хранилище данных m, если верно условие

$$\sum_{s \in S_m} u(s) \le uh(m).$$

Здесь S_m – множество storage, размещенных в хранилище данных m, а также тип хранилища данных совпадает с типом storage-элемента.

Репликацией называется отображение $R : H \to H$, дублирующее данные некоторого $m \in M$ и создающее виртуальный канал поддержки консистентности данных $(m', l_1, k_1, \ldots, k_{n-1}, l_n, m)$; $k_i \in K$, $l_i \in L$, $m' \in M$, m' – реплика хранилища m.

Если storage-элемент является репликацией некоторой базы данных s и требуется обеспечение консистентности данных, то в граф запрашиваемых ресурсов G добавляется вершина s'. В граф G также добавляется виртуальный канал между вершинами s и s', пропускная способность которого определяется исходя из требования обеспечения консистентности репликации и базы данных.

Остаточным графом доступных ресурсов называется граф H_{res}, для которого переопределены функции:

$$\nu h_{res}(p) = \nu h(p) - \sum_{w \in W_p} \nu(w), \quad qh_{res}(p) = qh(p) - \sum_{w \in W_p} q(w), \quad uh_{res}(m) = uh(m) - \sum_{s \in S_m} u(s),$$

$$\tau h_{res}(k) = \tau h(k) - \sum_{e \in E_l} r(e), \quad rh_{res}(l) = rh(l) - \sum_{e \in E_k} r(e).$$

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

1) множество запросов $Z = \{(G_i,\ T_i)\}$, поступивших центральному планировщику; здесь T_i – время, на которое запрашиваются ресурсы запросом G_i;

2) остаточный граф доступных ресурсов $H_{res} = (P \cup M \cup K, L)$.

Требуется: из множества Z разместить на выполнение в ЦОД максимальное число запросов. При этом должны быть верны условия:

$$\sum_{w \in W_p} \nu(w) \le \nu h(p), \quad \sum_{w \in W_p} q(w) \le qh(p);$$

$$\sum_{e \in E_l} r(e) \le rh(l);$$

$$\sum_{e \in E_k} \tau(e) \le \tau h(k);$$

$$\sum_{s \in S_m} u(s) \le uh(m); \quad \forall s \in S_m : type(s) = type(m).$$

Входом алгоритма назначения запросов на физические ресурсы являются остаточный граф доступных ресурсов H_{res} и множество ресурсных запросов $\{G_i\}$. Множество $\{G_i\}$ формирует диспетчер задач. В него, кроме вновь поступивших запросов, могут входить и запросы, которые выполняются и для которых допустима миграция. Диспетчер задач также определяет время запуска планировщика.

Выходом алгоритма назначения запросов на физические ресурсы являются множество назначений ресурсных запросов на физические ресурсы $\{A_i : G_i \to H, i = \overline{1,n}\}$ и множество репликаций $\{R_i\}$, $i = 0,1,\ldots$

2. Принципы построения алгоритмов назначения запросов на физические ресурсы ЦОД. Для решения сформулированной задачи были разработаны три алгоритма [18]:

1) алгоритм с общим планировщиком для всех типов ресурсов;

2) алгоритм с использованием независимых планировщиков для каждого типа ресурсов;

3) алгоритм на основе схемы муравьиных колоний.

Алгоритм с общим планировщиком для всех типов ресурсов основан на сочетании жадных стратегий и стратегий ограниченного перебора: в соответствии с тремя жадными критериями выбирается сначала очередной запрос, затем – элементы запроса и физические ресурсы для них. Если на очередном шаге не удалось разместить элемент запроса, то вызывается процедура ограниченного перебора. Если элемент является storage-элементом, то вызывается процедура репликации и только в случае неуспешного завершения процедуры репликации вызывается процедура ограниченного перебора. Для процедуры ограниченного перебора задается параметр, который позволяет регулировать вычислительную сложность и точность алгоритма.

При использовании в ЦОД независимых планировщиков для каждого типа ресурсов алгоритм назначения запросов на физические ресурсы состоит из трех шагов:

1) назначение виртуальных машин в минимальное число вычислительных узлов;

2) назначение storage-элементов в минимальное число физических хранилищ данных;

3) для полученных отображений виртуальных машин и storage-элементов на физические ресурсы построение отображения виртуальных каналов запросов на физические каналы сети обмена.

Алгоритмы назначения виртуальных машин и storage-элементов основаны на сочетании жадных стратегий и стратегий ограниченного перебора. В соответствии с тремя жадными критериями выбирается сначала очередной ресурсный запрос, затем – очередной элемент запроса (виртуальная машина/storage-элемент) и физический ресурс для него (вычислительный узел/хранилище данных). Если на очередном шаге не удалось разместить элемент запроса, то вызывается процедура ограниченного перебора. Для процедуры ограниченного перебора задается параметр, который позволяет регулировать вычислительную сложность и точность алгоритма. Алгоритм отображения виртуальных каналов на физические ресурсы основан на решении задачи о нахождении k кратчайших путей в графе [19]. Процедура репликации storage-элементов вызывается, если не удалось отобразить виртуальный канал на физические каналы.

Алгоритм на базе схемы муравьиных колоний может быть использован вместо алгоритма с общим планировщиком для всех типов ресурсов. Алгоритм основан на схеме муравьиных колоний [20] для отображения виртуальных машин и storage-элементов на вычислительные узлы и хранилища данных. Далее для полученных отображений виртуальных машин и storage-элементов на физические ресурсы строятся отображения виртуальных каналов запросов на физические каналы сети обмена с помощью алгоритма нахождения кратчайшего пути [21]. В отличие от алгоритмов, основанных на сочетании жадных стратегий и стратегий ограниченного перебора, данный алгоритм не требует подбора эвристических процедур для частных задач (заданы ограничения на возможные значения исходных данных), но имеет более высокую вычислительную сложность.

3. Результаты экспериментального исследования свойств алгоритмов. Все эксперименты проводились для ЦОД с архитектурой fat-tree.

3.1. Сравнение предложенных алгоритмов с алгоритмами платформы OpenStack. В данном подразделе приведены результаты сравнения предложенных алгоритмов по качеству получаемых отображений с алгоритмами из платформы OpenStack. Качество отображений будем оценивать процентом размещенных в ЦОД запросов из исходно заданного набора. Все эксперименты проводились для наборов слабосвязных запросов, поскольку алгоритмы из платформы OpenStack не поддерживают назначение сильносвязных запросов.

В платформе OpenStack реализовано два алгоритма назначения ресурсных запросов: First Fit и Random Fit. Для каждого из алгоритмов были сгенерированы наборы исходных данных, которые демонстрируют недостатки функционирования алгоритмов при высокой загрузке физических ресурсов ЦОД.

Для алгоритма First Fit были сгенерированы исходные со следующими характеристиками:

1) количество физических вычислительных узлов – $6n$ (n – натуральное) с производительностью в 210 усл. ед.;

2) общее число виртуальных машин – $18n$; запрос содержит:

2.1) $6n$ виртуальных машин с требуемой производительностью 31 усл. ед.;

2.2) $6n$ виртуальных машин с требуемой производительностью 71 усл. ед.;

2.3) $6n$ виртуальных машин с требуемой производительностью 106 усл. ед.

При неограниченном числе узлов алгоритм First Fit назначает все запросы в $10n$ узлов (рис. 2). Оптимальное назначение всех запросов возможно в $6n$ узлов (рис. 3). При ограничении числа узлов значением $6n$ алгоритм First Fit производит назначение всех виртуальных машин, кроме $4n$

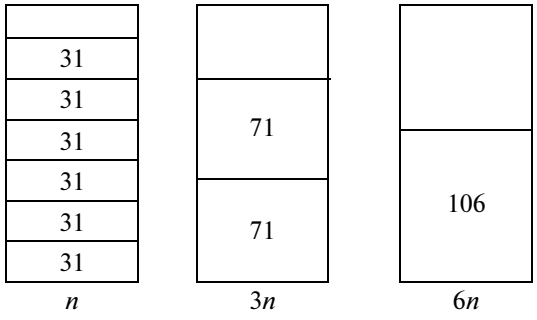

Рис. 2. Назначение ресурсов алгоритмом First Fit

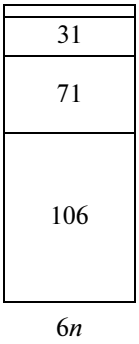

Рис. 3. Оптимальное назначение ресурсов

виртуальных машин с емкостью 106 усл. ед., получая при этом загрузку ресурсов, равную 65%. Предложенные алгоритмы успешно назначали все виртуальные машины с загрузкой физических ресурсов 99%. При проведении экспериментов было выяснено, что качество отображений не зависит от значения n.

Для алгоритма Random Fit были использованы исходные данные со следующими характеристиками:

1) количество вычислительных узлов $n + 1$ – каждый с производительностью 1 млн усл. ед.;

2) общее количество виртуальных машин $n^3 + n -$:

2.1) n^3 виртуальных машин с требуемой производительностью 1 млн/n^3 усл. ед.;

2.2) n виртуальных машин с требуемой производительностью 1 млн усл. ед.;

При проведении назначения запросы с маленькой емкостью заполняли предоставленные ресурсы таким образом, что не оставалось вычислительных узлов, в которых не было бы назначено хотя бы одной виртуальной машины. После этого виртуальные машины, для которых уже требовалось полная емкость вычислительного узла, уже не могли быть назначены. Получаемая загрузка ресурсов для алгоритма Random Fit равнялась в среднем 10%. Разработанные алгоритмы позволяли полностью назначить все виртуальные машины с загрузкой 100%. При проведении экспериментов было выяснено, что качество получаемых отображение не зависит от значения n.

Для демонстрации ухудшения качества работы алгоритмов платформы OpenStack в зависимости от потенциально возможной загрузки физических ресурсов была применена схема исследования с увеличивающейся нагрузкой. Для описанных тестов со значением $n = 10$ производилось удаление ресурсных запросов таким образом, чтобы потенциально возможная загрузка физических ресурсов понижалась с шагом 10%. Таким образом, было получено две серии тестовых данных с возрастающей загрузкой. Результаты работы алгоритмов приведены на рис. 4, 5.

Рис. 4. Результаты работы всех алгоритмов на данных для First Fit: –◆– разработанные алгоритмы, –■– First Fit, –▲– Random Fit

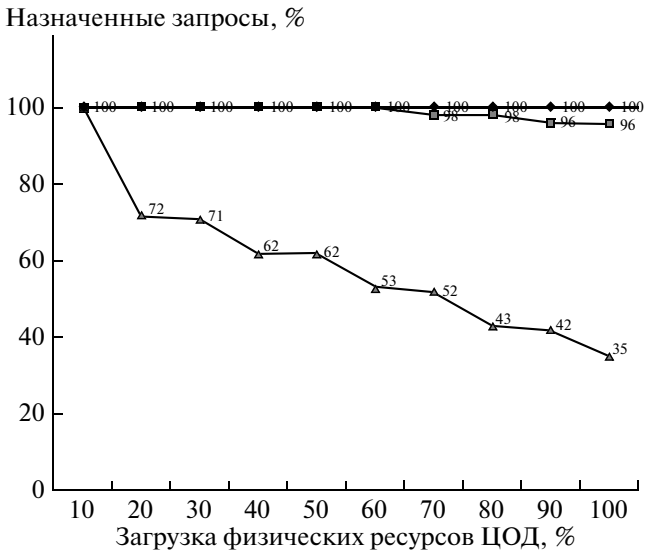

Рис. 5. Результаты работы всех алгоритмов на данных для Random Fit: –◆– разработанные алгоритмы, –■– First Fit, –▲– Random Fit

3.2. Исследование эффективности процедуры репликации. Для демонстрации эффективности процедуры репликации использовались следующие параметры топологии fattree:

1) корневой уровень архитектуры состоял из 10 маршрутизаторов с пропускной способностью 50000 усл. ед. каждый. Каналы передачи данных, соединяющие маршрутизаторы корневого и второго уровней, имели пропускную способность 10000 усл. ед.;

2) второй уровень архитектуры состоял из 15 маршрутизаторов с пропускной способностью 16000 усл. ед. каждый. Каналы передачи данных, соединяющие маршрутизаторы второго и первого уровней, имели пропускную способность 2000 усл. ед.;

3) каждый маршрутизатор второго уровня был соединен с четырьмя маршрутизаторами первого уровня с пропускной способностью 2000 усл. ед. Каждый маршрутизатор первого уровня соединялся с двумя вычислительными узлами или хранилищами данных каналами передачи с пропускной способностью 1000 усл. ед.;

4) всего в ЦОД имелось 60 вычислительных узлов с производительностью 1000 усл. ед. и 60 хранилищ данных с емкостью 1000 усл. ед. Каждый маршрутизатор второго уровня имел доступ к четырем вычислительным узлам и четырем хранилищам данных.

Для генерации набора запросов использовались четыре основных шаблона:

1) шаблон с маленьким числом виртуальных каналов: семь виртуальных машин, пять storage-элементов и 11 виртуальных каналов (в среднем по два виртуальных канала на один storage-элемент);

2) шаблон с большим числом виртуальных каналов: пять виртуальных машин, два storage-элемента и восемь виртуальных каналов (по четыре виртуальных канала на один storage-элемент);

3) шаблон с маленьким числом виртуальных машин: три виртуальных машины и два storage-элемента. Один из storage-элементов был соединен со всеми виртуальными машинами, а второй – с одной из них (всего четыре виртуальных канала);

4) шаблон с большим требованием к пропускной способности сети: две виртуальных машины, один storage-элемент и два виртуальных канала между ними. Значения запрашиваемой пропускной способности виртуальных каналов выбирались так, чтобы их сумма была равна 900 усл. ед. (при пропускной способности канала, ведущего к хранилищу данных, в 1000 усл. ед.).

Наборы запросов делились на два основных класса и генерировались по следующим схемам.

В первом классе варьировалась потенциально возможная загрузка каналов (при оптимальном размещении), ведущих к хранилищам данных, от 0.3 до 1.0. Загрузка вычислительных узлов и хранилищ данных была фиксированной и равнялась 0.75. В данном классе присутствовали только запросы, сгенерированные по первому и второму шаблонам.

Рис. 6. Алгоритм с раздельными планировщиками на первом классе данных: —◆— алгоритм с репликацией, —■— алгоритм без репликации

Рис. 7. Алгоритм с раздельными планировщиками на втором классе данных: —◆— алгоритм с репликацией, —■— алгоритм без репликации

Во втором классе исходных данных использовались только запросы, сгенерированные по третьему и четвертому шаблонам. В данном классе варьировалось количество запросов, сгенерированных по четвертому шаблону, от 0 до 30, при этом потенциально возможная загрузка сети возрастала от 0.5 до 0.8. Загрузка вычислительных узлов и хранилищ данных была фиксированной и равнялась 0.75.

Количество запросов в наборе всегда равнялось 100.

Все алгоритмы запускались на каждом из наборов обоих классов исходных данных один раз с запретом репликации, и один раз с ее разрешением. Алгоритм на основе схемы муравьиных колоний выполнял 70 итераций, параметр глубины перебора в алгоритмах с общими и раздельными планировщиками ресурсов равнялся двум. На следующих рисунках изображена разница в количестве размещенных запросов каждым из алгоритмов с запретом и разрешением операции репликации.

На рис. 6 и 7 представлены результаты работы алгоритма с раздельными планировщиками ресурсов на первом и втором классах данных.

На рис. 8 и 9 показаны результаты работы алгоритма с единым планировщиком ресурсов на первом и втором классах данных.

Рис. 8. Алгоритм с единым планировщиком на первом классе данных: —◆— алгоритм с репликацией, —■— алгоритм без репликации

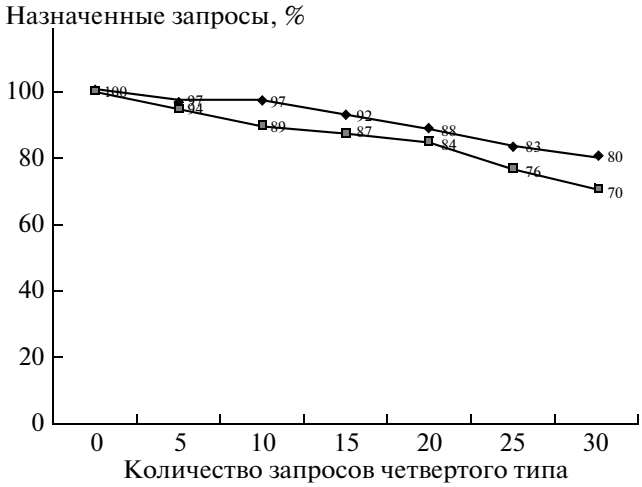

Рис. 9. Алгоритм с единым планировщиком на втором классе данных: —◆— алгоритм с репликацией, —■— алгоритм без репликации

На рис. 10 и 11 приведены результаты работы муравьиного алгоритма на первом и втором классах данных.

Исследования алгоритмов на первом классе исходных данных показали, что процедура репликации позволяет повысить количество размещаемых запросов при потенциально возможной загрузке каналов, ведущих к хранилищам данных, 0.7 и выше. Для второго класса исходных данных аналогичные результаты были получены уже при наличии хотя бы пяти запросов с большим требованием к пропускной способности сети обмена.

Также следует отметить, что на первом классе данных алгоритм с единым планировщиком размещает больше запросов, чем алгоритм с раздельными планировщиками, тогда как на втором классе данных ситуация обратная. Муравьиный алгоритм при этом работает эффективнее обоих алгоритмов как на первом, так и на втором классах данных.

3.3. Области эффективного применения разработанных алгоритмов. Для демонстрации типичных областей применения алгоритмов использовалась сеть ЦОД с топологией fat-tree со следующими характеристиками:

1) корневой уровень архитектуры состоит из пяти маршрутизаторов с пропускной способностью 100000 усл. ед.;

Рис. 10. Муравьиный алгоритм на первом классе данных: –◆– алгоритм с репликацией, –■– алгоритм без репликации

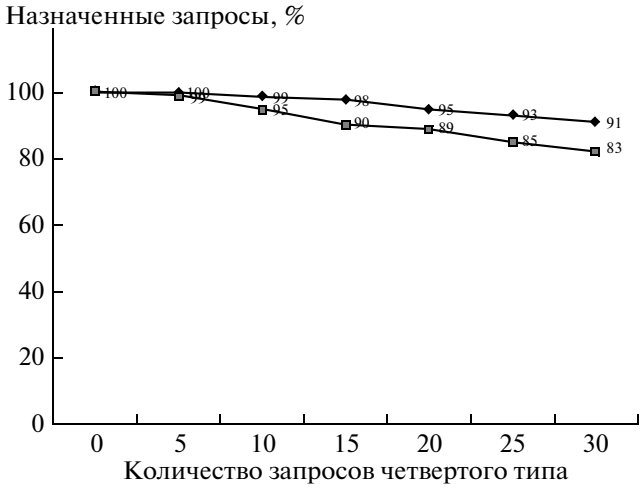

Рис. 11. Муравьиный алгоритм на втором классе данных: –◆– алгоритм с репликацией, –■– алгоритм без репликации

2) второй уровень архитектуры состоит из 10 маршрутизаторов с пропускной способностью 20000 усл. ед. Первый и второй уровни архитектуры соединены в сеть двудольным графом, пропускная способность каналов связи составляет также 20000 усл. ед.;

3) к каждому маршрутизатору второго уровня подключено по четыре маршрутизатора третьего уровня с пропускной способностью 5000 усл. ед. каждый. Пропускная способность каналов связи между вторым и третьим уровнями сети составляет 5000 усл. ед. на канал;

4) совокупность одного маршрутизатора второго уровня и подключенных к нему четырех маршрутизаторов третьего уровня будем называть сегментом топологии fat-tree. Листовой уровень топологии каждого сегмента формируется 10 вычислительными узлами и 10 хранилищами данных, которые подключены к маршрутизаторам третьего уровня из расчета пять машин одного типа на маршрутизатор.

Данная топология отражает применяемую на практике топологию независимых сегментов с дополнительным уровнем доступности. При такой организации работы ЦОД сильно связанные запросы размещаются в одном сегменте для уменьшения задержек передачи данных и повышения производительности межсетевого взаимодействия.

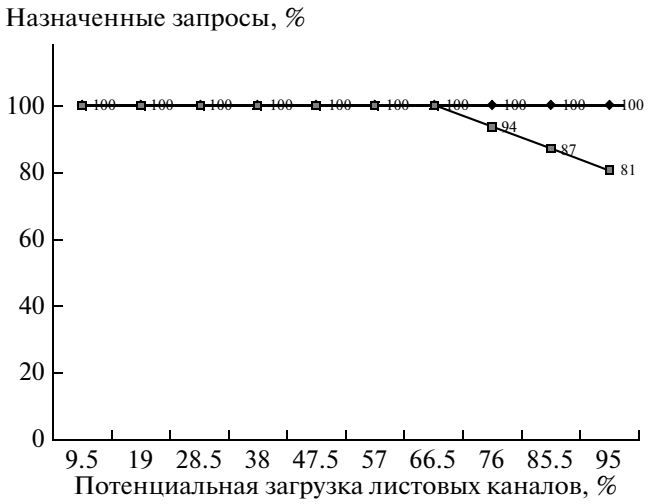

Рис. 12. Результаты исследования на данных первого класса: –◆– алгоритм с единым планировщиком, –■– алгоритм с раздельными планировщиками

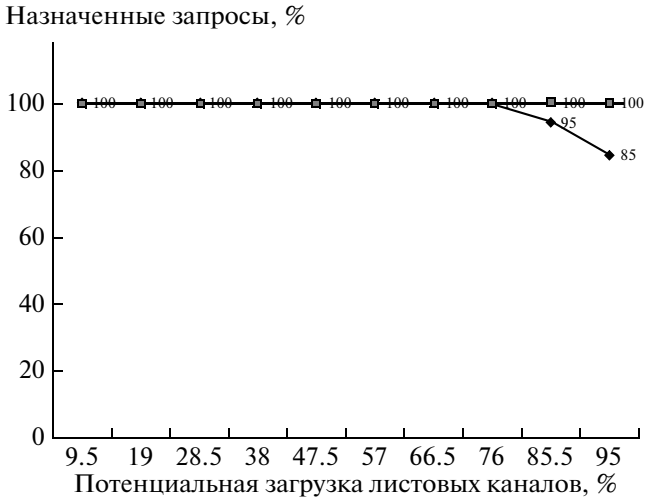

Рис. 13. Результаты исследования на данных второго класса: –◆– алгоритм с единым планировщиком, –■– алгоритм с раздельными планировщиками

Исследования проводились для алгоритмов с единым планировщиком ресурсов и раздельными планировщиками ресурсов. При проведении исследования глубина ограниченного перебора была ограничена значением два. При увеличении значения глубины алгоритмы начинают вести себя асимптотически одинаково, однако время работы полиномиально возрастает.

Исследования проводились на двух классах данных.

1. Наборы запросов из первого класса были получены с помощью генератора оптимальной посегментной загрузки. Для заданной топологии он позволяет получать наборы запросов, полное размещение которых требует загрузки каждого типа ресурсов ЦОД не менее чем на 95%. Набор запросов формируется заданием различного числа загруженных сегментов (от 1 до 10). Соответственно потенциально возможная загрузка ресурсов ЦОД (при оптимальном отображении запросов на физические ресурсы) составляет для этого класса от 9.5 до 95%.

2. Второй класс данных получен теми же методами, что и первый, но потенциальная загрузка сетевых каналов листового уровня для него составляет в среднем 50% с интервалом разброса 5%. Дополнительно введено ограничение на соответствие запросов шаблонам “тяжелого” и “легкого” типов. Математическое ожидание требуемой производительности элементов запроса “тяжелого” типа составляет 60% производительности физического ресурса (вычислительного узла или

хранилища данных). Соответствующее значение для запросов “легкого” типа составляет 20% производительности физических узлов.

Для каждого класса исходных данных генерировалось в среднем по 200 запросов на сегмент, обеспечивающих его загрузку на 95% при оптимальном отображении. Число виртуальных машин и хранилищ данных в запросе – от 10 до 20, отношение числа виртуальных машин и хранилищ данных к числу виртуальных каналов равно 1.

Исследования алгоритмов на первом классе исходных данных показали, что алгоритм с единым планировщиком для всех типов ресурсов позволяет назначать запросы в полном объеме вплоть до 95% загрузки физических ресурсов ЦОД. Алгоритм с раздельными планировщиками для каждого типа ресурсов дает худшие результаты в условиях, когда потенциальная загрузка физических ресурсов ЦОД превышает 70%.

Исследования алгоритмов на втором классе исходных данных показали, что алгоритм с раздельным планировщиком для каждого типа ресурсов позволяет назначать запросы в полном объеме вплоть до 95% загрузки вычислительных узлов и хранилищ данных ЦОД. Алгоритм с единым планировщиком для всех типов ресурсов дает худшие результаты в условиях, когда потенциальная загрузка этих физических ресурсов превышает 80%.

Результаты сравнения алгоритмов приведены на рис. 12 и 13.

Заключение. Предложенные в работе подход и алгоритмы отображения ресурсных запросов на физические ресурсы ЦОД обладают следующими отличительными особенностями:

1) допускают возможность миграции виртуальных ресурсов, что позволяет устранять сегментацию физических ресурсов, которая может возникать в ходе работы ЦОД;

2) допускают возможность репликации хранилищ данных в целях повышения надежности хранения данных и эффективности использования физических ресурсов;

3) допускают возможность рассматривать все типы ресурсов (вычислительные ресурсы, хранилища данных, сетевые ресурсы) как планируемый тип ресурса. Это делает возможным задание SLA для всех типов ресурсов.

Экспериментальные исследования показали, что разработанные алгоритмы позволяют получать отображение ресурсных запросов на физические ресурсы ЦОД, для которых возможно выполнения заданных требований SLA при загрузке физических ресурсов до 95%.

СПИСОК ЛИТЕРАТУРЫ

1. *Pepple K.* Deploying OpenStack. Sebastopol: O’Reilly, 2011.
2. *Amies A, Sluiman H., Tong Q.G. et al.* Developing and Hosting Applications on the Cloud. Boston: IBM Press, 2012.
3. *Urgaonkar B., Rosenberg A.L., Shenoy P.* Application Placement on a Cluster of Servers // Intern. J. Foundations of Computer Science. 2007. T. 18. № 05. P. 1023–1041.
4. *Bein D., Bein W., Venigella S.* Cloud Storage and Online Bin Packing // Proc. of the 5th Intern. Symp. on Intelligent Distributed Computing. Delft: IDC, 2011. P. 63–68.
5. *Nagendram S., Lakshmi J.V., Rao D.V. et al.* Efficient Resource Scheduling in Data Centers Using MRIS // Indian J. Computer Science and Engineering. 2011. V. 2. Issue 5. P. 764–769.
6. *Arzuaga E., Kaeli D.R.* Quantifying Load Imbalance on Virtualized Enterprise Servers // Proc. of the First Joint WOSP/SIPEW Intern. Conf. on Performance Engineering. San Josa, CA: ACM, 2010. P. 235–242.
7. *Mishra M., Sahoo A.* On Theory of VM Placement: Anomalies in Existing Methodologies and Their Mitigation Using a Novel Vector Based Approach // Cloud Computing (CLOUD), IEEE Intern. Conf. Washington: IEEE Press, 2011. P. 275–282.
8. *Botero J.F., Hesselbach X., Fischer A. et al.* Optimal Mapping of Virtual Networks with Hidden Hops // Telecommunication Systems. 2012. V. 51. № 4. P. 273–282.
9. *Yu M., Yi Y., Rexford J. et al.* Rethinking Virtual Network Embedding: Substrate Support for Path Splitting and Migration // ACM SIGCOMM Computer Communication Review. 2008. V. 38. № 2. P. 17–29.
10. *Lischka J., Karl H.* A Virtual Network Mapping Algorithm Based on Subgraph Isomorphism Detection // Proc. of the 1st ACM Workshop on Virtualized Infrastructure Systems and Architectures. Barcelona: ACM, 2009. P. 81–88.
11. *Zhu Y., Ammar M.H.* Algorithms for Assigning Substrate Network Resources to Virtual Network Components // 25th Intern. Conf. on Computer Communications. Barcelona: INFOCOM, 2006. P. 1–12.
12. *Chowdhury N.M.M.K., Rahman M.R., Boutaba R.* Virtual Network Embedding with Coordinated Node and Link Mapping // 28th Intern. Conf. on Computer Communications. Barcelona: INFOCOM, 2009. P. 783–791.

13. *Cheng X., Sen S., Zhongbao Z. et al.* Virtual Network Embedding Through Topology-aware Node Ranking // ACM SIGCOMM Computer Communication Review. 2011. V. 41. № 2. P. 38–47.
14. *Korupolu M., Singh A., Bamba B.* Coupled Placement in Modern Data Centers // IEEE Intern. Symp. on Parallel & Distributed Processing. N.Y.: IPDPS, 2009. P. 1–12.
15. *Singh A., Korupolu M., Mohapatra D.* Server-Storage Virtualization: Integration and Load Balancing in Data Centers // Proc. of the 2008 ACM/IEEE Conf. on Supercomputing. Austin: IEEE Press, 2008. P. 1–12.
16. *Jiang J.W., Tian L., Sangtae H. et al.* Joint VM Placement and Routing for Data Center Traffic Engineering // 31st Intern. Conf. on Computer Communications. Orlando: INFOCOM, 2012. P. 2876–2880.
17. *Al-Fares M., Loukissas A., Vahdat A.A.* Commodity Data Center Network Architecture // ACM SIGCOMM Computer Communication Review. 2008. V. 38. № 4. P. 63–74.
18. *Вдовин П.М., Зотов И.А., Костенко В.А. и др.* Задача распределения ресурсов центров обработки данных и подходы к ее решению // VII Московская междунар. конф. по исследованию операций (ORM2013). М.: ВЦ РАН, 2013. Т. 2. С. 30–32.
19. *Eppstein D.* Finding the *k* Shortest Paths // SIAM J. on Computing. 2006. V. 28. № 2. P. 652–673.
20. *Kostenko V.A., Plakunov A.V.* An Algorithm for Constructing Single Machine Schedules Based on Ant Colony Approach // J. of Computer and Systems Sciences Intern. 2013. V. 52. № 6. P. 928–937.
21. *Ahuja R.K., Magnanti T.L., Orlin J.B.* Network Flows: Theory, Algorithms, and Applications. New Jersey: Prentice Hall, 1993.