УДК 004.75

В. Гайдамако, Институт машиноведения и автоматики Национальной академии наук, Бишкек

МОДЕЛИРОВАНИЕ СЕРВЕРА В ИНФРАСТРУКТУРЕ ОБЛАЧНОГО ЦЕНТРА ОБРАБОТКИ ДАННЫХ НА БАЗЕ ПЛАТФОРМЫ SIMGRID

В статье рассматривается моделирование сервера облачного центра обработки данных с использованием фреймворка моделирования распределенных и облачных систем Simgrid на примере трехуровневого веб-приложения. Реализация унифицированного представления сервера упрощает как описание моделируемой платформы, позволяя легко изменять состав и количество виртуальных серверов, физических машин и машин пользователей, так и написание кода сервера.

Ключевые слова: облачные вычисления, центр обработки данных, балансировка нагрузки, сервер, моделирование, Simgrid.

Введение. Имитационное моделирование распределенных и облачных систем позволяет провести оценку производительности системы с различным набором и параметрами компонентов в условиях, когда постановка натурного эксперимента или слишком дорога, или затруднительна, или вообще невозможна. Моделирование сервера центра обработки данных (ЦОД) является частью комплекса работ по созданию модели облачной информационно-измерительной системы (ОИИС) [1], включающей ЦОДы, беспроводные сенсорные сети (БСС) и отдельные датчики, группы пользователей и приложения пользователей. Для подключения приложений пользователей создаются виртуальные датчики. Доступ к измерительной информации, виртуальным и физическим датчикам осуществляется через веб-сайт. ЦОД включает серверы разного типа, работающие на виртуальных машинах, причем количество их может меняться во время работы приложения в зависимости от нагрузки, а виртуальные машины могут перемещаться от одной физической машины к другой. Модель ОИИС создается для исследования работы системы при проектировании или изменении состава оборудования/программного обеспечения, оценки производительности системы, а также для выбора оптимальной политики и алгоритма балансировки нагрузки. Балансировка нагрузки играет важнейшую роль в обеспечении производительности облачных и распределенных вычислений, ее функциями являются распределение вычислительных ресурсов и планирование задач. Для создания имитационной модели была выбрана платформа (фреймворк) моделирования распределенных и облачных приложений Simgrid [2, 3, 4], позволяющая моделировать физические хосты, устройства и линии связи, энергопотребление, виртуальные машины и их перемещение между серверами, различную топологию сетей, различную нагрузку, в том числе меняющуюся во времени.

Инфраструктура облачного центра обработки данных (ЦОД)

Основой облачных систем являются центры обработки данных, в которых собраны тысячи блейд-серверов, связанных скоростной локальной сетью. ЦОДы соединяются между собой через скоростные каналы связи. Пользователи подключаются к ЦОД через интернет или глобальную сеть организации, обычно к “ближайшему” доступному серверу. Для удовлетворения запроса пользователя в ЦОД создается или экземпляр приложения/сервиса, работающий на виртуальной машине (модель SaaS – программное обеспечение как услуга), или виртуальная машина/контейнер, содержащий требуемую «платформу» − рабочую среду (PaaS – платформа как услуга), или целая виртуальная инфраструктура, включающая виртуальные машины и связывающую их локальную сеть (IaaS – инфраструктура как услуга). Физические и виртуальные машины или контейнеры являются ресурсами облака.

В рассматриваемой модели в начале работы ЦОД представляет собой один работающий сервер – контроллер (брокер) облака и набор физических серверов, ожидающих заданий на выполнение. Задания выполняются на виртуальных ресурсах (машинах или

контейнерах). Брокер облака принимает или отклоняет поступающие задания и запускает инфраструктуру для реальных приложений.

В состав любого ЦОД обязательно входят серверы, обеспечивающие работу ЦОД и облака в целом – контроллер (брокер) ЦОД, балансировщик нагрузки, сервер мониторинга. Другие серверы, например, серверы для создания инфраструктуры приложений клиентов запускаются, копируются и удаляются по необходимости. Выбирается доступная физическая машина (ФМ, хост), на ней создается виртуальная машина (ВМ) с программным обеспечением нужного сервера. При увеличении нагрузки создаются дополнительные копии сервера, при уменьшении нагрузки лишние копии удаляются. Если запросы приходят от отдаленных групп пользователей, копии серверов могут перемещаться (мигрировать) в другой ЦОД, более “близкий” к пользователю. Такой алгоритм используется для масштабирования как серверов облака, так и серверов приложений клиентов (арендаторов) облака. На рисунке 1 представлена инфраструктура для работы трехуровневого веб-приложения.

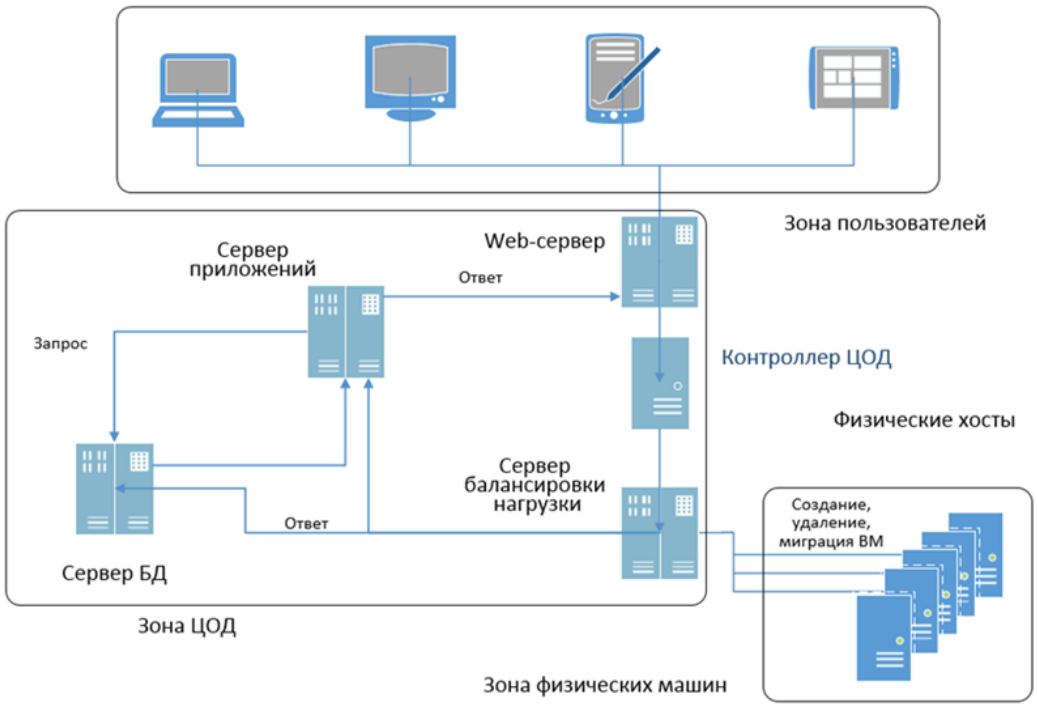

Рисунок 1– Инфраструктура ЦОД для трехуровневого веб-приложения

Серверы ЦОД:

- контроллер ЦОД (брокер облака) – cb_xx, координирует всю работу ЦОД и принимает решение о приеме или отклонения запроса на обработку
- сервер-балансировки нагрузки (loadbalancer) lb_xx, в соответствии с алгоритмом балансировки нагрузки выбирает экземпляр (ВМ) сервера запрашиваемого типа или физической машины для обслуживания запроса пользователя или другого сервера. Доступная физическая машина определяется по алгоритму Round Robin, то есть это первая машина в циклической очереди;
- сервер хранения db_xx – сервер баз данных, там хранится информация, необходимая для обработки запроса пользователя;
- сервер приложений app_xx обрабатывает запросы пользователей, формирует запросы к серверам хранения (базам данных), принимает ответ и формирует ответ на запрос пользователя;
- веб-сервер, web_xx, принимает запросы пользователей и высылает ответы. При получении запроса пользователя формируется соответствующий запрос к серверу

приложений (трехуровневая архитектура), который также выбирается сервером балансировки;

- физические хосты (Phisical Mashine) pm_xx, на которых, собственно, размещаются виртуальные ресурсы (виртуальные машины и контейнеры), представляющие виртуальные серверы, которые являются экземплярами сервисов. Физическая машина описывается как не самый производительный Intel(R) Core(TM) i5 с 4 ядрами, скорость около 20 Гфлопс (операций с плавающей точкой) для каждого ядра, эти характеристики могут быть изменены в файле платформы [2].

Группа пользователей:

- группа пользовательских машин UZ представлена в виде кластера, каждая машина через случайные интервалы времени посылает запрос контроллеру облака. В идеале для имитации нагрузки от пользователей необходимо создать базу данных пользователей, которая содержит информацию о расположении и характере активности групп пользователей. Пользователи объединяются в группы по признаку, определяющему профили активности пользователей, – например, по территории, от этого зависят часы активности. Для группы указывается также объем входящего и исходящего трафика на запрос. В целях отладки нагрузка от пользователей может генерировать запросы с постоянным или меняющимся периодом.

Для линий связи указывается пропускная способность (bandwitdth) – максимальное количество единиц информации в секунду и задержка (latency), которая может возникать, например, из-за необходимости кодирования или упаковки данных. Будем рассматривать два типа линий связи:

- линия связи между облаком и внешней средой – интернет-соединение, скорость может быть самой разной, предположим, что это не самые быстрые соединения – 10 – 30 Мб/с. Это линии связи между между пользователями и веб-сервером;
- линии связи внутри облака – внутри ЦОД. Линии связи между ЦОД более скоростные – 1Gб/с [2].

Simgrid представляет различные классы сетевых зон, в данной модели используется Full Zone для зоны всей платформы и Cluster Zone[4] для описания зоны физических машин внутри ЦОД и зоны пользовательских машин. При задании топологии с отдельными зонами обязательно должны быть заданы маршруты между зонами с тегом <zoneRoute>.

Трехуровневое веб-приложение. Рассмотрим работу облачного трехуровневого веб-приложения – SaaS (рис.2). В таком приложении запрос клиента направляется на веб-сервер (уровень презентации P, здесь – сервер nginx), веб-сервер пересылает запрос серверу приложений (уровень приложения B, сервер TomCat), который для формирования ответа на запрос делает запрос к серверу базы данных (уровень данных D, MySQL)[5]. Сервер для обслуживания запросов назначается сервером балансировки исходя из используемого алгоритма балансировки нагрузки [6]. При получении ответа от сервера базы данных (БД) все происходит в обратном порядке – сервер приложений формирует страницу ответа, передает ее веб-серверу, веб-сервер доставляет новую страницу клиенту. Все серверы являются виртуальными, работают на физических серверах, их количество может увеличиваться и уменьшаться в зависимости от количества запросов. Для любых веб-приложений время ответа на запрос должно составлять менее 2 секунд, а в идеале – менее 1 секунды, будем считать время отклика веб-приложения главным критерием, или главной метрикой производительности системы.

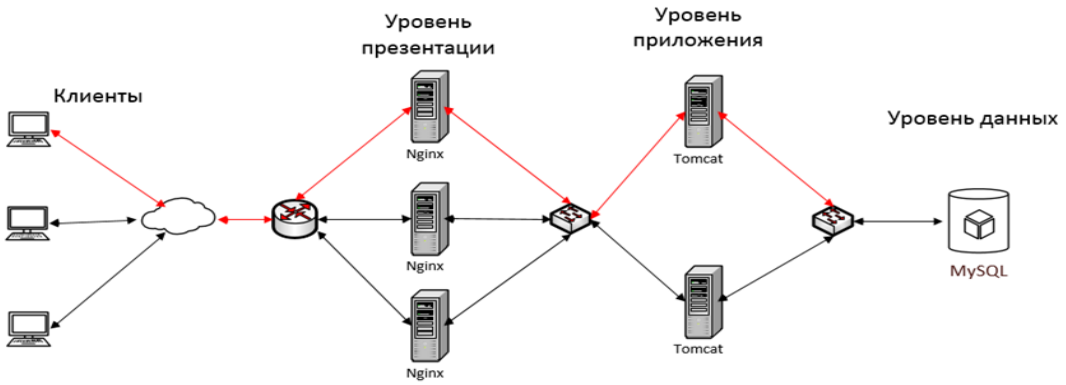

Рисунок 2 − Инфраструктура для трехуровневого веб-приложения (без учета балансировки нагрузки) [4]

Предположим, что все серверы, кроме сервера балансировки, работают с образованием канала связи с клиентом (statefull), то есть запросы от пользователя и презентация результатов обрабатываются одной виртуальной машиной, представляющей веб-сервер, запросы к базе данных и обработка результатов – подготовка веб-страницы (формы) осуществляется тем же экземпляром сервера приложений, который получил первоначальный запрос от веб-сервера. Для учета балансировки нагрузки (служебный уровень S) добавляются серверы координатор облака и балансировщик нагрузки (рис. 1), и запрос будет проходить уже 5 ВМ, причем некоторые более чем по одному разу. Последовательность обработки запроса разными серверами представлена на рисунке 3, как видно, запрос обслуживается уже в 8 этапов.

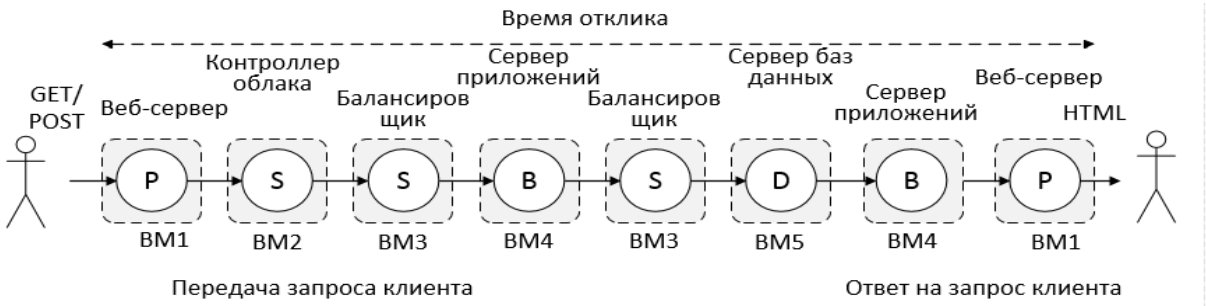

Рисунок 3 − Последовательность обработки запроса пользователя с учетом балансировки нагрузки

Моделирование ЦОД с использованием SimGrid

Модель SimGrid состоит из нескольких акторов, выполняющих заданные пользователем функции. Действия акторов (активности) выполняются на ресурсах (хосты, линии связи и диски). SimGrid прогнозирует время, затрачиваемое на каждое действие, и соответствующим образом организует работу акторов[2, 4].

Каждый актор выполняется на физическом хосте или виртуальной машине. Хосты находятся в сетевых зонах, внутри зоны явно или неявно описываются связи между всеми хостами. Для каждой сетевой зоны описываются маршруты к другим зонам. Сетевые зоны могут включаться в другие зоны, образуя дерево сетевых зон. Корневая зона содержит всю платформу. Для коммуникации между акторами используются почтовые ящики.

Для проведения эксперимента по моделированию необходимо представить файл платформы и файл развертывания. В файле платформы содержится описание ресурсов – физических машин, устройства хранения, сетевых устройств и линий связи в формате XML.Ресурсы могут также программно создаваться и удаляться во время работы модели, то есть во время моделирующего эксперимента. Эксперимент описывается в файле развертывания (deploymentfile), задается последовательность запуска активностей на хостах и параметры функций для активностей. [2,3,4].

Если прямо описывать структуру, представленную на рисунке 1, будет ясно, что файл платформы даже для такой структуры довольно громоздок, так как, помимо перечисления серверов, нужно описать все линии связи и маршруты между ними[3]. Добавление нового сервера приводит к необходимости описать множество новых связей и маршрутов, с увеличением количества серверов задача становится слишком трудоемкой и плохо контролируемой. Поэтому для сложных моделей более логично было бы представлять работу модели так же, как и работу ЦОД, – сначала работает только контроллер облака, и он запускает все нужные серверы (рис. 4).

Такая организация работы ЦОД предполагает постоянное создание новых серверов с описанием взаимодействия между ними, причем, если серверы находятся в кластере, связи и маршруты между ними описывать не нужно, нужно только создать ил назначить соответствующие почтовые ящики. Благодаря унифицированному представлению сервера и правилам именования почтовых ящиков задача создания новых серверов существенно упрощается. В результате файл платформы содержит описание зоны ЦОД из одного сервера (впрочем, и его можно было бы запускать программно, но так удобнее передавать аргументы), и кластеры для физических машин ЦОД и машин группы пользователей, три линии связи между ними и три маршрута между зонами. Сравним описания платформы.

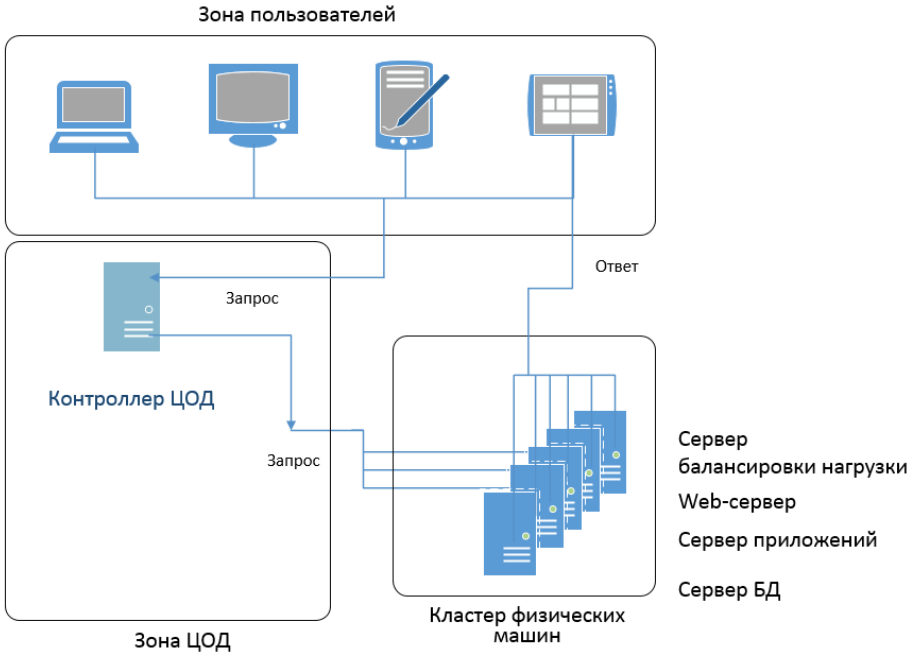

Рисунок 4 – Инфраструктура ЦОД для трехуровневого веб-приложения с одним сервером в начале работы

Описание зоны ЦОД с жестко заданной инфраструктурой [3]:

```
<zoneid="DC01" routing="Full">
<host id="ST01" speed="20Gf" core="4" />
<host id="CS01" speed="20Gf" core="4" />
<host id="LB01" speed="20Gf" core="4" />
<host id="WS01" speed="20Gf" core="4" />
<host id="AS01" speed="20Gf" core="4" />
<link id="LC1-01" bandwidth="1GBps" latency="100us"/>
<link id="LC1-02" bandwidth="1GBps" latency="100us"/>
<link id="LC1-03" bandwidth="1GBps" latency="100us"/>
<link id="LC1-04" bandwidth="1GBps" latency="100us"/>
<link id="LC1-05" bandwidth="1GBps" latency="100us"/>
<link id="LC1-06" bandwidth="1GBps" latency="100us"/>
<link id="LCS1-1" bandwidth="30MBps" latency="350us"/>[2]
<link id="LDCPM1-01" bandwidth="1GBps" latency="350us"/>
<link id="LDCUZ1-01" bandwidth="30MBps" latency="350us"/>
<route src="CS01" dst="ST01"><link_ctn id="LC1-01"/>-01"/></route>
```

```
<route src="CS01" dst="LB01"><link_ctn id="LC1-02</route>
<route src="LB01" dst="ST01"><link_ctn id="LC1-03</route>
<route src="LB01" dst="WS01"><link_ctn id="LC1-04</route>
<route src="LB01" dst="AS01"><link_ctn id="LC1-05</route>
<route src="WS01" dst="AS01"><link_ctn id="LC1-05</route>
</zone>
```

При последовательном запуске серверов:

```
<zone id="datacenter" routing="None">
<host id="cs1" speed="20Gf" core="4" />
</zone>
```

Описание кластера физических машин и зоны пользователей остается без изменений:

```
<zone id="blades" routing="Full">
<cluster id="pm"  prefix="pm1-" radical="0-10" suffix=".pm"
            speed="20Gf" core="4"bw="1000MBps" lat="100us"
bb_bw="1GBps" bb_lat="100us" router_id="pm1-pm_router.pm" />
</zone>
```

```
<cluster id="users" prefix="u-" suffix=".uz" radical="0-10" speed="1Gf" bw="125MBps" lat="50us"
       router_id="u-users_router.uz"/>
```

Маршруты между зонами:

```
<zoneRoutesrc="datacenter" dst="users" gw_src="cs1" gw_dst="u-users_router.uz">       <link_ctn
id="link1"/>
</zoneRoute>
<zoneRoutesrc="datacenter" dst="blades" gw_src="cs1" gw_dst="pm1-pm_router.pm">
       <link_ctn id="link2"/>
</zoneRoute>
<zoneRoutesrc="blades" dst="users" gw_src="pm1-pm_router.pm" gw_dst="u-users_router.uz">
       <link_ctn id="link3"/>
</zoneRoute>
```

Линии связи: множество линий связи и маршрутов между всеми элементами в описании зоны ЦОД с начальной инфраструктурой и три линии связи в новом описании:

```
<link id="link1" bandwidth="100kBps" latency="100us"/>
<link id="link2" bandwidth="1000MBps" latency="10us"/>
<link id="link3" bandwidth="100kBps" latency="100us"/>
```

Как видим, описание зоны ЦОД значительно упростилось, при добавлении новых серверов не нужно описывать их связи и маршруты ко всем другим серверам.

Файл развертывания описывает запуск активности (действия, осуществляемые акторами). Для каждой активности создается соответствующая функция, в файле задаются значения аргументов. В таблице 1 приводится описание акторов с соответствующими активностями в файле развертывания.

Таблица 1 – Файл развертывания

“Жесткое” описание зоны ЦОД (6 акторов)	Последовательный запуск серверов (1 актор)
<actor host="ST01" function="storage"/> <actor host="CS01" function="cloudstation"> <argument value="200"/> <argument value="500000"/> <argument value="10000"/> <argument value="5000" /> </actor> <actor host="WS01" function="webServer"> <argument value="S01"/> </actor> <actor host="LB01" function="loadBalancer"> <argument value="500000"/> </actor> <actor host="AP01" function="appServer">	</platform> <platform version="4.1"> <actor host="cs1" function="cloudBroker"> <argument value="1"/><!—cloud brokers > <argument value="1"/><!-- load balancers > <argument value="2"/><!-- data bases --> <argument value="2"/><!-- apps --> <argument value="2"/><!-- webs \ --> <argument value="10"/><!-- user cycles --> <argument value="1000"/> !-- request_size --> <argument value="10000"/><!--response_size --> <argument value="100000000"/><!--calculation_cost--> </actor> </platform>

<argument value="500000"/> </actor> <actor host="ST01" function="Storage"> <argument value="500000"/> </actor>[3]	

В функции main() регистрируются функции активностей, считываются из файлов конфигурация платформы и файл развертывания, и на хостах в заданном порядке запускаются активности [2,4]. В данном случае из функции main() при старте моделирования запускаются активности пользователя на физической машине из зоны пользователей. Эта активность генерирует запросы к веб-серверу, имитируется работа с приложением – пользователь своим действием, например, нажатием кнопки, запускает запрос к веб-серверу, после получения ответа через случайный интервал (в пределах 5 – 10 с) снова запускает запрос, и так 10 раз (количество циклов можно изменить в файле развертывания).

Все серверы запускаются брокером облака и работают по одному унифицированному алгоритму (рис. 5).Упрощается как описание платформы, можно легко добавлять другие ЦОДы и группы пользователей, так и написание кода серверов, нужно отдельно описывать только блок обработки запроса, к тому значительно упрощается работа с почтовыми ящиками.

Унифицированное представление сервера

В рассматриваемой модели ЦОД первоначально существует только контроллер (брокер) облака и кластер физических машин. Виды и начальное количество серверов, необходимых для развертывания приложения, задаются в файле развертывания и запускаются контроллером при старте модели. Каждый сервер выполняет унифицированную функцию сервера: ожидать (слушать) запросы клиента (пользователя или другого сервера) и запускать виртуальную машину, выполняющую функцию обслуживания, на физической машине, указанной балансировщиком нагрузки (рис.5). Примерно так работают серверы в приложениях клиент-сервер с взаимодействием через сокеты (sockets) [7] – они ждут клиентов («слушают») и при установке соединения создают свою копию, на которую переключают клиента. Копия сервера обслуживает клиента, а главный сервер продолжает “слушать”.

Для удобства серверы, запускаемые координатором облака, будем называть базовыми серверами, или серверами нулевого уровня. Их назначение – слушать и принимать запросы от клиентов (пользователи и другие серверы), создавать виртуальную машину на физическом сервере, указанном балансировщиком нагрузки. На созданном сервере (сервер первого уровня) запускается функция обслуживания запроса клиента, которая может потребовать обращения к другому серверу (сервер второго уровня). В этом случае сервер первого уровня обращается к базовому серверу нужного типа, тот создаст нужный сервер первого уровня и передаст его адрес клиенту для дальнейшего взаимодействия в процессе обслуживания начального запроса от клиента.

Контроллер облака после запуска необходимых серверов также начинает работать как сервер, именно через контроллер облака проходит все запросы, которые или принимаются и перенаправляются на сервер обработки, или отклоняются и перенаправляются в другой ЦОД. На данном этапе принимаются все запросы, рассматривается работа одного ЦОД. Принятый запрос передается на базовый веб-сервер, указанный балансировщиком нагрузки.

Основная сложность – описать взаимодействие серверов, которое моделируется через почтовые ящики (MailBox). В рассматриваемой модели каждый физический или виртуальный хост имеет почтовый ящик для чтения, его имя совпадает с именем машины. Таким образом, чтобы передать что-то другому хосту, надо знать его имя. Для упрощения были введены правила именования серверов и виртуальных машин. Каждый тип сервера имеет свой префикс, с которого будет начинаться его имя – "web_", "app_","db_","lb_". Имена серверов нулевого уровня (базовых) строятся из префикса и порядкового номера в группе серверов данного типа, например "web_0", "web_1", "app_0","db_0","lb_0". При

создании сервера первого уровня к префиксу добавляется имя клиента – web_user0 и так далее. По таким правилам клиент всегда может построить имя сервера и работать с ним, но только после того, как получит сообщение от сервера более высокого уровня, это будет гарантией того, что сервер для обслуживания запроса существует.

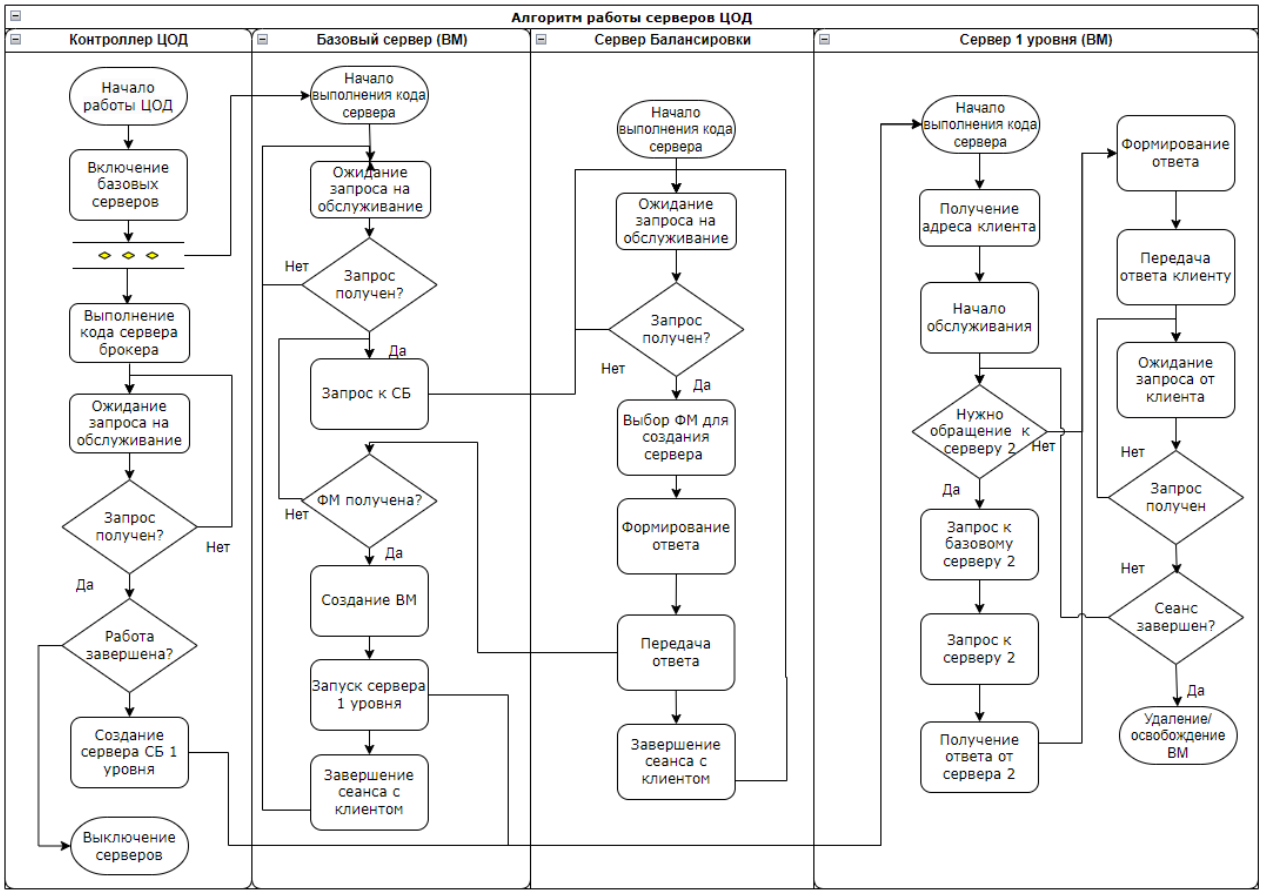

Рисунок 5 – Алгоритм работы унифицированного сервера

При такой организации мы получаем ограничение – так как ящик для входящих сообщений один, взаимодействие должно быть организовано таким образом, чтобы всегда было однозначно ясно, откуда получено сообщение. Для синхронной работы с установкой соединения (statefull) такой способ вполне подходит, но, если возможны асинхронные сообщения или запросы от других серверов, сообщение должно содержать имя или тип сервера, а в коде сервера должна быть соответствующая проверка Можно было бы поступить по-другому, для связи с новым сервером создавать новый ящик для чтения, но по опыту это приводит к крайнему усложнению кода. Поэтому решение с одним входным почтовым ящиком и в последовательном обращении к необходимым серверам на данный момент является вполне удовлетворительным.

Для связи с серверами, не участвующими в обработке запроса, например с сервером мониторинга, собирающим данные о состоянии виртуальной или физической машины, лучше создавать ящики с именем сервера и номером порта, подобно URL-адресам, как рекомендуют разработчики Simgrid[4]. Для обслуживания запроса сервера мониторинга (pull-запросы, когда соединение инициирует сервер мониторинга) или для отправки данных по инициативе рабочего сервера (push-запросы) будет создаваться дополнительный актор на том же физическом сервере, так как для моделирования параллельного исполнения на машинах с количеством ядер более одного рекомендуется создавать виртуальные машины с одним ядром[4], и для этой виртуальной машины в любом случае будет создаваться отдельный почтовый ящик.

Проведение эксперимента и анализ результатов моделирования

Целью эксперимента является проверка работоспособности модели ЦОД с унифицированным сервером при разном количестве пользователей и серверов. Функция main(), функции активностей, вспомогательные функции разрабатывались на языке C++. Для отладки использовались средства журналирования SimGrid – APIXBT_LOG [2, 4]. Отладочные сообщения позволяют проследить последовательность обработки запроса, и особенно они важны на этапе отладки взаимодействия серверов, когда их количество сравнительно невелико. Например, последовательность выполнения запроса от пользовательской машины №6:

```
[u-6.uz:u-6.uz:(9) 2290.846041] [s4u_app_cl/INFO] UHUMAN 'u-6.uz' sendingdata 'request  19'
toserver 'web_u-6.uz'
[pm1-5.pm:web_u-6.uz:(41) 2292.170115] [s4u_app_cl/INFO] VM WS id = web_u-6.uz recived message
'request  19' from userbox  u-6.uz
[u-6.uz:u-6.uz:(9) 2292.170115] [s4u_app_cl/INFO] UHUMAN 'u-6.uz' waiting for response from
server 'web_u-6.uz'
[pm1-7.pm:app_web_u-6.uz:(43) 2292.174126] [s4u_app_cl/INFO] VM APP id = app_web_u-6.uz, received
data 'app1_get' from web_u-6.uz
[pm1-5.pm:web_u-6.uz:(41) 2292.174126] [s4u_app_cl/INFO] VM WS id = web_u-6.uz sent message
'request  19' to app app_0
[pm1-5.pm:web_u-6.uz:(41) 2292.188626] [s4u_app_cl/INFO] VM WS id = web_u-6.uz, mailbox is u-6.uz
sending data is DB respose from db_app_web_u-6.uz20 APP respose from  20 20
[u-6.uz:u-6.uz:(9) 2387.644928] [s4u_app_cl/INFO] UHUMAN 'u-6.uz'  received data 'DB respose from
db_app_web_u-6.uz20 APP respose from  20 20' from server 'web_u-6.uz'
```

Так, можно проследить похождение всех запросов, но при сложной топологии сети и большом количестве серверов это уже не имеет смысла.

При запуске модели создается файл трассировки simgrid.trace в формате Paje [8], в котором отражается использование вычислительной мощности хостов и пропускной способности линий связи во время симуляции. Этот файл можно разобрать на трассировки для отдельных компонентов и вычислить, какая часть времени была затрачена на прием, передачу данных и вычисления, построить временные диаграммы и даже визуализировать топологию сети, например, с помощью интерпретатора R[9]. Анализ временных диаграмм с помощью Vite [10] также позволяет выявлять ошибки и неточности в топологии модели, коде и коммуникациях между акторами.

В данной модели не представлен мониторинг ресурсов, который необходим для разработки и тестирования динамических алгоритмов балансировки нагрузки. В проводимых экспериментах использовался только алгоритм RoundRobin (RR). При определении физического хоста для запуска очередной виртуальной машины для каждого запроса на создание ВМ использовался первый в циклической очереди физический хост. При выборе базового сервера – первый сервер в очереди серверов заданного типа. Для обслуживания запроса пользователя создавались новые экземпляры серверов на новых виртуальных машинах, после завершения сессии с пользователем виртуальные машины уничтожались. Сессия пользователя включает заданное количество запросов, последовательно пересылаемых на веб-сервер через случайный интервал времени в диапазоне 5–10 с, что имитирует работу пользователя с веб-страницей, в данном случае пользовательское приложение посылает 10 запросов. Размер запроса 1000 байт, размер ответа и от базы данных, и от сервера приложений – 10кБ (типичный размер веб-страницы на сентябрь 2022 года 2.2 МБ [11]). Так как главным критерием оценки работы системы является время обработки запроса (оно не должно превышать 2 с), рассмотрим этот показатель для разного количества пользователей и разного стартового количества базовых серверов.

Пользователь отправляет запрос в почтовый ящик и сразу же начинает ожидать ответа. После получения ответа выполняется simgrid::s4u::this_actor::sleep_for(..). В файле состояний ресурсов для машины пользователя отображаются строки в формате "Type", "Actor", "Type1", "Start", "End", "Duration", "Level", "State":

```
"Type", "Actor", "Type1", "Start", "End", "Duration", "Level", "State"
State, u-1.uz-3, ACTOR_STATE, 0.000000, 44.000000, 44.000000, 0.000000, sleep
State, u-1.uz-3, ACTOR_STATE, 44.000000, 187.000657, 143.000657, 0.000000, send
State, u-1.uz-3, ACTOR_STATE, 44.130338, 187.000657, 142.870319, 1.000000, receive
State, u-1.uz-3, ACTOR_STATE, 44.296852, 58.296852, 14.000000, 2.000000, sleep
State, u-1.uz-3, ACTOR_STATE, 58.296852, 187.000657, 128.703805, 2.000000, send
State, u-1.uz-3, ACTOR_STATE, 58.428622, 187.000657, 128.572035, 3.000000, receive
State, u-1.uz-3, ACTOR_STATE, 58.582634, 72.582634, 14.000000, 4.000000, sleep
State, u-1.uz-3, ACTOR_STATE, 72.582634, 187.000657, 114.418023, 4.000000, send
State, u-1.uz-3, ACTOR_STATE, 72.714403, 187.000657, 114.286254, 5.000000, receive
State, u-1.uz-3, ACTOR_STATE, 72.868416, 86.868416, 14.000000, 6.000000, sleep
```

Перед началом работы пользователь выдерживает паузу, чтобы все серверы успели запуститься. Далее посылается первый запрос, который придет к контроллеру облака, контроллер перенаправит его на доступный базовый веб-сервер (указанный планировщиком), базовый веб-сервер создаст новую ВМ (сервер 1 уровня) для обработки запроса. Первый ответ придет уже от обслуживающего сервера 1 уровня. Особенности работы почтовых ящиков таковы, что созданный ящик не уничтожается, поэтому все операции отправки к одному серверу заканчиваются в момент завершения процесса пользователя, и поле “End” не может использоваться для оценки времени завершения операции. Операция “send” завершается только тогда, когда сообщение принято. Поэтому в момент начала следующей операции “receive” сообщение (запрос) точно получено сервером. Но и начало, и завершение операции “receive” не могут использоваться для определения времени ответа от сервера, операция “receive” завершится только при завершении работы пользователя. А момент начала следующей операции – “sleep” как раз означает, что ответ на запрос получен. Поэтому время ответа будем определять как разность между временем начала sleep и временем начала send. Результаты представлены в таблице 2.

В первых двух строках для 1 и 10 пользователей все значения удовлетворительные, это отладочные запуски, и они интересны только для отслеживания правильности прохождения запроса. При увеличении количества пользователей до 100 видно, что сильно увеличилось время ответа на первый запрос, то есть увеличилось время на принятие запроса, поиск подходящего сервера и создание экземпляра обслуживающего сервера. Время обслуживания запроса, когда соединение с обслуживающим сервером уже установлено, практически не меняется.

Таблица 2 – Среднее, минимальное и максимальное время ответа на запрос пользователя

№	Количество машин							Время обработки первого запроса			Время обработки запроса		
	cb	lb	db	app	web	pm	users	Ср.	Min	Max	Ср.	Min	Max
1	1	1	1	1	1	2	2	.29685	.296852	.296852	.28578	.285781	.285782
2	1	1	1	1	1	2	10	.35681	.296849	.560660	.28579	.285781	.285890
3	1	1	1	1	1	2	100	1.94974	.296849	4.200625	.28580	.285781	.286888
4	10	1	1	1	5	2	100	1.80569	.296849	3.940081	.28578	.285781	.285782
5	10	10	1	1	5	10	100	1.74818	.296849	3.940081	.28578	.285781	.285852
6	10	10	1	1	5	100	100	2.02705	.296849	4.365600	.28578	.285780	.285890
7	10	10	1	1	5	100	1000	61.4945	.296849	122.6663	.28582	.285770	.286928

Увеличение количества доступных физических машин ситуацию меняет незначительно, увеличение количества базовых серверов только ухудшает ситуацию. «Узкое место» – это сам контроллер облака – он является точкой входа в систему, принимает решение об отклонении или принятии запроса, поэтому разумно увеличить количество точек входа, то есть применить горизонтальное масштабирование. Для более тонкой балансировки необходимо вести мониторинг состояния физических и виртуальных серверов, что поможет выявить другие узкие места, а также применить более совершенные алгоритмы балансировки нагрузки.

Заключение

Представленный подход к моделированию сервера значительно облегчает создание моделей ЦОД и позволяет добавлять новые типы серверов, при этом для каждого типа сервера нужно написать только блок обработки. Количество серверов каждого типа также легко изменяется, так как не надо отдельно описывать новые связи между серверами, правила именования почтовых ящиков, используемых SimGrid для имитации работы линий связи, позволяет унифицировать код сервера. Унифицированный код сервера позволяет моделировать инфраструктуру приложений с любой длиной последовательных обращений к серверам, требуется только написать функции обработки запроса для серверов всех типов. Представленная модель ЦОД не является полноценной моделью, скорее это прототип модели. Требуется работа по калибровке модели для уточнения параметров программного обеспечения, серверов, линий передачи данных, задержки с использованием параметров, получаемых от реальных приложений, журналов реальных веб-серверов. Верификация модели для простого случая с фиксированным набором серверов может быть проведена на примере реального приложения, но в основном это сравнение с результатами, получаемыми в экспериментах с другими моделями, например, ИРВ-моделью[5], моделями CloudAnalyst [12]. CloudAnalyst – графическое приложение на основе платформы моделирования CloudSim[13], использующееся множеством исследователей алгоритмов балансировки нагрузки в облачных системах. Для исследования динамических алгоритмов балансировки нагрузки в модель ЦОД необходимо включить сервер мониторинга, который будет собирать данные о текущем состоянии физических и виртуальных машин. Более подробное описание разных групп пользователей, подключение датчиков позволит создать полноценную модель ОИИС. Для исследования алгоритмов балансировки нагрузки между ЦОД создается несколько зон ЦОД со своими кластерами физических машин, описываются линии связи и маршруты между ЦОД, а в код контроллера добавляется проверка критериев принятия или перенаправления поступающего запроса в другой ЦОД. Таким образом, на настоящий момент получен прототип инструмента для исследования алгоритмов балансировки как внутри одного, так и между нескольких ЦОД, который планируется развивать далее. Модель ОИИС также получила развитие, так как теперь можно представить ЦОД не как жестко заданную конфигурацию серверов, а как облачную инфраструктуру, меняющуюся в зависимости от нагрузки.

Литература

1. Гайдамако В.В. Инфраструктура Sensor-Cloud – облачные информационно-измерительные системы. // Проблемы автоматики и управления – 2018. – №2 (35). –С. 109 – 118.

2. SimGrid: Versatile Simulation of Distributed Systems/ URL:https://simgrid.org/ (дата обращения 25.09.2023)

3. Гайдамако В.В. Моделирование облачной информационно-измерительной системы с помощью библиотеки Simgrid // Проблемы автоматики и управления. –Бишкек:Илим, №1 (36). – 2019. – С.90–99.

4. SimGrid Tutorials. URL: https://simgrid.org/doc/latest/intro_concepts.html (дата обращения 25.09.2023).

5. Гайдамако В.В. Моделирование производительности многоуровневого веб-приложения на основе исчисления реального времени // Проблемы автоматики и управления. – Бишкек: Илим, №2 (39), 2020.

6. Dalia Abdulkareem Shafiq, N.Z. Jhanjhi, Azween Abdullah, Load balancing techniques in cloud computing environment: A review // Journal of King Saud University - Computer and Information Sciences, v. 34, issue 7,2022, pp 3910-3933, https://doi.org/10.1016/j.jksuci.2021.02.007/ URL: https://www.sciencedirect.com/science/article/pii/S131915782100046X (датаобращения 25.09.2023)

7. Уолтон, Шон. Создание сетевых приложений в среде Linux. / Пер. с англ.— М.: Изд-ий дом. "Вильяме", 2001. — 464 с.: ил. — Парал: ти. англ. ISBN 5 8459.

8. L. Mello Schnorr, B. de Oliveira Stein, J. Chassin de Kergommeaux. Paje trace file Format, version 1.2.5. Technical Report, February, 2013. PAJÉ – trace file format/ URL: http://paje.sourceforge.net/download/publication/lang-paje.pdf/ (дата обращения 25.09.2023).

9. Visual Trace Explorer / URL: http://vite.gforge.inria.fr/index.php/ (дата обращения 25.09.2023).

10. The R Project for Statistical Computing / URL: https://www.r-project.org// (дата обращения 25.09.2023).

11. Webpage Size – Why is it important? And how do you optimize it? / URL: https://www.seoptimer.com/blog/webpage-size/ (дата обращения 25.09.2023).

12. CLOUDSIM & CLOUD SETUP / URL: https://cloudsim-setup.blogspot.com/2013/01/running-and-using-cloud-analyst.html(дата обращения 25.11.2023).

13. CloudSim: A Framework For Modeling And Simulation Of Cloud Computing Infrastructures And Services / URL: https://github.com/Cloudslab/cloudsim(датаобращения 25.11.2023).