САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
КАФЕДРА КОМПЬЮТЕРНОГО МОДЕЛИРОВАНИЯ И МНОГОПРОЦЕССОРНЫХ СИСТЕМ

МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ

Тема: «Исследование модели виртуального вычислительного кластера на основе контейнеров приложений»

Направление: 02.04.02 – ФИиИТ

Магистерская программа: ВМ.5502.2014 – Вычислительные технологии

Выполнил студент гр. 633 Кобышев Сергей Сергеевич

Научный руководитель,
PhD В. В. Корхов

Санкт-Петербург – 2016

Содержание

Введение

Высокопроизводительные вычисления занимают важную роль как в сфере информационных технологий как предмет обучения, так и во многих смежных сферах, как инструмент вычислений, моделирования, анализа и т.д. В частности благодаря им есть возможность создавать тренажеры, которые имитируют реальные условия с целью подготовки квалифицированных кадров. В случае натурного эксперимента, есть вероятность утраты дорогостоящего оборудования или, что еще хуже, потери человеческих жизней. В случае виртуальной эмуляции всегда есть возможность прекратить эксперимент и предотвратить трагедию. Данные системы используются для прогнозирования погоды, собирая и обрабатывая огромное количество данных с метеостанций, расположенных по всему миру. Даже для того, чтобы найти необходимую информацию в интернете используется огромное количество вычислительных средств. Одним из способов организации высокопроизводительных вычислений является создание кластера. Кластер — это объединение множества отдельных вычислительных ресурсов в единую систему. Что позволяет получить огромный вычислительный потенциал. Однако, к сожалению при построении кластера не учитываются особенности исполняемых программ. В большинстве своем кластеры не подстраиваются под конкретную задачу, предоставляя всевозможные ресурсы без учета особенностей запускаемых задач. Часто не учитывается то, что на предоставляемых ресурсах она может работать неэффективно. Кроме того, само приложение может быть рассчитано на использование других по мощности ресурсов. И в случае, когда предоставляемые ресурсы будут отличаться от прогнозируемых, производительность приложения может пострадать. Также возможна ситуация, когда программа умышленно или случайно потребляет максимально возможное количество ресурсов или мешает другим приложения исполняться. То есть в целом носит па-

разитический характер. Вне зависимости от того является ли это злым умыслом, или ошибкой в программе, необходимо предусмотреть механизм предотвращения подобного поведения.

Предложенная в данной работе модель решает часть проблем описанных выше. Используя разработанную систему, у пользователей появляется возможность сконфигурировать ресурсы окружения исходя из своих необходимостей и нужд. При этом есть возможность выбрать оптимальную конфигурацию ресурсов и, в случае наличия в резерве достаточного количества ресурсов, создать виртуальный кластер, на котором непосредственно и произойдет выполнение программы. При этом, с одной стороны, ресурсы кластера подстраиваются под задачу, с другой стороны, для каждой задачи создается ограниченное по ресурсам окружение. Любые действия программы изолированы и не влияют на исполнение других задач. По окончанию работы, использованные ресурсы освобождаются, а созданный кластер уничтожается со всеми составляющими.

Постановка задачи

Целью работы является создание виртуальной распределенной вычислительной среды, которая позволит сконфигурировать предоставляемые вычислительные ресурсы в соответствиям с требованиями приложения.

Для достижения цели необходимо было решить следующие задачи:

- провести обзор предметной области;
- выбрать основу для виртуальной вычислительной архитектуры;
- разработать модель для выявления оптимальных параметров;
- создать прототип виртуальной вычислительной системы;
- провести тестирование полученной модели.

Глава 1. Технологии виртуализации

Понятие виртуализация имеет множество определений, в данном случае понимаются инструменты и подходы, которые позволяют абстрагироваться от аппаратной основы вычислительной системы. Работа с системой на более высоком уровне абстракции позволяет более гибко работать с ресурсами. Для примера рассмотрим ситуацию работы с некоторым сервером. При отсутствии виртуализации, в случае выхода сервера из строя, для продолжения работы, необходимо поставить новый сервер, сконфигурировать его и произвести замену старого сервера на новый. Так как новая аппаратура полностью независима от предыдущей, то нет гарантий, что все настроено правильно и будет работать также как и раньше. В случае, если сервер является некоторой виртуальной сущностью, то после выхода из строя его можно подменить копией, созданной на другой аппаратной основе, если это необходимо. При этом виртуализацию можно также использовать для уменьшения нагрузки на сервер. В случае с виртуализацией есть возможность оперативного динамического распределения нагрузки на копии исходного сервера. В случае с отсутствием виртуализации это труднореализуемо. Это только некоторые из примеров. В реальности существует множество ситуаций, когда необходимо некоторое настроенное окружение. При наличии виртуализации развертывание и настройка нового окружения занимает гораздо меньше времени. При этом большая часть нового элемента будет полностью идентична старому.

Виртуализация делится на следующие классы:

- Виртуализация платформ.

  Опираясь на работу [1] можно выделить следующие технологии виртуализации платформ:

  1 Полная виртуализация (Full Virtualization).

Помимо данного термина в некоторых работах используется термин hardware emulation. В рамках этой технологии используется система управления виртуальными машинами (hypervisor), которая управляет виртуальными машинами. При этом происходит полная изоляция виртуальной машины от машины, на которой производится ее запуск. Все элементы виртуальной машины эмулируются и система управления отвечает за выполнение команд, эмуляцию устройств и взаимодействие между машинами. Некоторые инструменты позволяют ускорить производительность, например за счет кэширования одинаковых инструкций.

2 Частичная виртуализация (Paravirtualization).

Как и в предыдущем случае данная технология подразумевает систему управления. Однако, помимо полной виртуализации она вносит изменения в операционную систему, на которой запускается. Это позволяет оптимизировать процесс выполнения для более высокой производительности по сравнению с полной виртуализацией. Преимущество данного подхода является то, что все устройства остаются полностью виртуализованными. Недостатком подхода является то, что необходимо вносить изменения в операционную систему. В случае, когда исходный код операционной системы закрыт это может привести к проблемам.

3 Виртуализация на уровне операционной системы.

В отличии от предыдущих подходов, при использовании данной технологии основным инструментам для работы с виртуальными единицами является операционная система. Которая в свою очередь создает изолированные экземпляры некоторой операционной системы. Одно ядро управляет ресурсами всех

экземпляров. Гостевая операционная система часто связаны с Virtual private servers (VPS). Преимуществом данного подхода является еще больший прирост производительности по сравнению с предыдущими подходами. Отсутствием системы управления и отсутствием необходимости модификации операционной системы. При этом полученные показатели производительности в целом близки к показателям полученным на исходной машине. Помимо этого использование одного ядра для всех экземпляров уменьшает количество ресурсов, необходимых для поддержки нескольких экземпляров. Главным недостатком является то, что при выходе из строя узла, отвечающего за управление всеми экземплярами, они также выходят из строя.

4 Нативная виртуализация (Native virtualization).

Данный подход подразумевает наличие поддержки инструментов виртуализации у аппаратных устройств. Цель заключается в том, чтобы сократить расходы при виртуализации. Он позволяет запускать множество задач, которые работают совместно и выполняются напрямую на процессоре без эмуляции. Выйгрыш в производительности особенно заметен, если мы сравниваем со случаем, когда происходит эмуляция другой версии процессора. Текущие версии процессоров поддерживают эту технологию.

- Виртуализация ресурсов.

К данному классу относится объединение ресурсов. В том числе и кластеризацию вычислительных ресурсов, для объединения их в единую вычислительную систему, с целью использования, например, для распределенных вычислений. Помимо этого данный подход включает в себя консолидацию серверной части, тем самым позволяя выполнять задачи пользователя параллельно, и увеличивая производи-

тельность системы. Виртуализация особенна успешна, если достигнут достаточно высокий уровень инкапсуляции и пользователь работает с системой, как с единым целым, не опускаясь на более низкие уровни взаимодействия с системой.

Существует множество аспектов виртуализации в целом. Начнем с инструментов, которые используются для виртуализации сети. Далее произведен обзор существующих решений для моделирования сети, в частности:

1 Mininet;

2 Cisco Packet Tracer;

3 GNS3;

4 NetBoxIT;

Использование данных программных продуктов позволяет эмулировать работу сети. Проводить тестирование и настройку используя различные топологии сети и различные сетевые параметры. Несмотря на общую основу они имеют разные цели. При этом все решения обладают общим недостатком: они направлены на моделирование сети на локальном компьютере. При использовании их в составе кластера возникают проблемы интеграции. Ниже вкратце рассмотрен каждый из них.

1 Mininet.

Позволяет работать с software-defined networking (SDN) сетями. Отличительной особенностью данных сетей является то, что отдельно существует уровень управления. Согласно данному подходу вся топология сети, построения маршрутов передачи данных и в целом вся логика обработки данных в сети происходит на данном уровне. Устройства, которые непосредственно отвечают за передачу данных

занимаются только исполнением команд, приходящих с уровня управления и, по факту, занимаются только передачей данных [2]. Продукт Mininet дает набор средств для тестирования и отладки работы SDN сетей. Используя его есть возможность разработать и отладить контроллер управления сетью и произвести моделирование работы сети с использованием данного контроллера. Что позволяет оценить применимость использования данной технологии и произвести оценку необходимости внедрения. В данный момент Mininet используется в Стэнфорде, Принстоне и других ведущих вузах для решения практических заданий в обучающих курсах по компьютерным сетям. В частности для выполнения заданий курса Stanford CS144 «Введение в компьютерные сети» [3]. Недостатком является то, что Mininet эмулирует только то, что касается непосредственно сети. Все остальные ресурсы — это ресурсы локальной машины на которой происходят вычисления.

2 Cisco Packet Tracer.

Позволяет эмулировать большую часть элементов сети. При этом данный продукт хорошо использовать для начинающих, например [4, 5]. При дальнейшем усложнении архитектуры сети данный программный продукт перестает удовлетворять некоторым критериям. Трудности возникают уже на этапе проектирования VPN, а при детальном описании становиться полностью не применим. Для моделирования более сложных систем используют следующий продукт.

3 Graphical Network Simulator 3 (GNS3) [6];

Он не концентрируется на какой-то конкретной технологии и позволяет эмулировать локальную сеть в целом. Он содержит в себе функциональность предыдущего программного продукта и расширяет ее охватывая большую часть технологий. Позволяет не только моделировать

сети, но также и управлять внешними аппаратными средствами. Ко всему прочему является бесплатно распространяемым программным обеспечением с открытым исходным кодом. К сожалению у данного продукта есть ряд недостатков. Во-первых его возможности порождают сложности в применении. Используя предыдущий инструмент есть возможность абстрагироваться от некоторых элементов в сети. В данном же случае сеть рассматривается детально, что требует наличия определенного опыта в данной сфере. Во-вторых свободно распространяемое ПО, часто, обладает рядом проблем связанных со стабильностью работы. К сожалению данный программный продукт не является исключением.

4 NetBoxIT;

Позволяет эмулировать разнородные сети, в частности его можно использовать для моделирования сети аварийных служб, включающей в себя все устройства от портативных устройств у сотрудников до взаимодействия со спутниками [7]. Однако несмотря на подробное описание принципов работы и результаты проводимых экспериментов, данный продукт не находится в открытом доступе, в связи с этим нет возможности произвести его полноценный обзор из-за недостатка материала.

Подводя итог, существует множество способов виртуализации сетевых ресурсов. В данной работе рассмотрены некоторые из них. Большая их часть эмулирует работу сети на локальном компьютере, не задействуя внешние ресурсы. Это позволяет спроектировать взаимодействие сетевых элементов, однако использование внешних аппаратных устройств затруднительно. Инструменты, которые обладают необходимой функциональностью содержат ряд других недостатков, которые усложняют процесс их использования. Более подробное развитие темы виртуализации сетей мож-

но найти, например в работе [8].

В данный момент в качестве основы построения распределенной системы можно использовать следующие элементы:

1 Аппаратный узел.

Данный подход подразумевает, полное отсутствие виртуализации и использование только аппаратных средств. Применяется в случаях, когда количество узлов не велико и однородно. В случаях, когда количество аппаратных средств велико их становиться тяжело настраивать и конфигурировать. Помимо этого необходимо контролировать работу пользователей и поддерживать изолированность. Когда поддерживать аппаратное решение становится проблематично, используют средства виртуализации и оперируют с технологиями представленными ниже.

2 Виртуальная машина.

Используя данную технологию происходит программно-аппаратная эмуляция для работы некоторой платформы [9]. Все взаимодействие с аппаратными ресурсами происходит через гипервизор. Данный подход использует полную виртуальную эмуляцию аппаратной системы, что позволяет исполнять на ней различные операционные системы. При этом существуют как решения для кластеров, так и для отдельных устройств. Существует ряд инструментов, которые дают возможность работать с ними, в частности:

- Virtualbox;
- VMware;
- KVM;
- Xen;
- и другие

Одним из популярных решений является VMWare vSphere, на его основе можно построить два вида кластеров:

- HA (High-availability) кластер;
- DRS (Distributed Resource Scheduler) кластер.

Различие данных видов заключается в том, что в первом случае упор производится на доступность узлов кластера. И при выходи необходимого узла из строя создается новый, являющийся его копией узел. После чего все запросы, адресованные на упавший узел перенаправляются на копию. Для контролирования процесса создания копий можно использовать VMware Fault Tolerance. Однако данное решение обладает рядом требований, которые могут усложнить процесс его использования или заставить от него отказаться вообще.

Во втором случае все ресурсы кластера объединяются в единую вычислительную единицу и осуществляется множество процессов, необходимых для поддержание эффективной и постоянной работы. В качестве примера можно привести автоматическое распределение нагрузки на аппаратную составляющую кластера. Для более эффективной работы используются дополнительные средства, например технология vMotion. Преимущество данной технологии заключается в том, что она позволяет произвести миграцию виртуальной машины с одного сервера на другой. При этом нет необходимости останавливать виртуальную машину или прекращать выполнение процессов в ней. Все происходит в режиме реального времени и пользователь этого не замечает.

Надо понимать, что виртуальная машина эмулирует большую часть составляющих реальной вычислительной машины. Это удобно, когда необходимо поделить вычислительные мощности между группой

пользователей, тем самым у каждого из пользователей будет создаваться ощущение, что он работает на реальной физической машине, которая располагается удаленно. Наша цель не ориентирована на пользователей. Мы направлены на анализ приложений на кластере. С этой точки зрения использование виртуальных машин является излишней растратой ресурсов, которые мы хотим минимизировать.

3 Контейнеры.

Контейнеры — система виртуализации. Использует уровень операционной системы позволяет запускать несколько операционных систем. В своей основе использует системы взаимодействия с ядром системы.

Данный метод похож на предыдущий, с разницей в том, что в основе находятся не виртуальные машины, а контейнеры приложений.

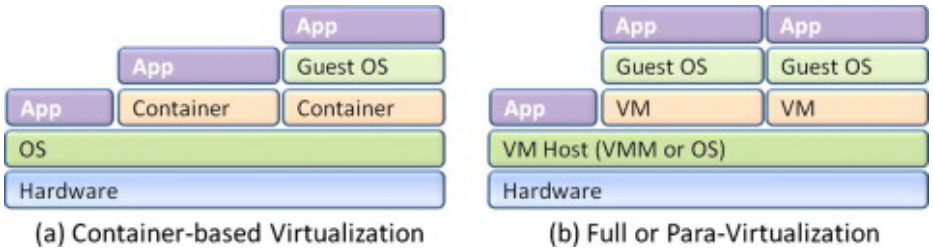

Рис. 1. Различие контейнеров и виртуальных машин [7].

Используя данную технологию взаимодействие с виртуальной единицей происходит через операционную систему, при этом гипервизор отсутствует рис. 1. Существует несколько контейнеров, основополагающим является LXC (LinuX Container), а также LXD, CGManager, LXCFS [10]. Для более простой и удобной работы используют программные продукты, которые в свою очередь базируются на контейнерах и представляют удобный интерфейс взаимодействия с ними. В качестве примера можно привести одну из популярных платформ для работы с контейнерами Docker[11]. В целом использование данного

подхода позволяет создавать виртуальные единицы более эффективно и менее ресурсозатратно, по сравнению с созданием виртуальной машины. При этом функциональные возможности, которые предоставляют контейнеры приложений схожи с предыдущим подходом. К сожалению имеется ряд недостатков, главным из них является то, что механизмы для работы с контейнерами опираются на возможности ядра операционной системы. Тем самым, если ядро операционных систем различно, то нет возможности ее эмуляции в контейнере. При работе с новыми версиями операционных систем Linux возможны ситуации, когда возникают проблемы совместимости и часть функционала работает некорректно [12]. Говоря же про другие операционные системы, например Windows, возникают более критические проблемы. Из-за того, что данные операционные системы имеют различные ядра, с первого взгляда кажется, что нет возможности использовать функциональность, предоставляемую контейнерам в ОС Windows. Однако существуют решения, когда на базе ОС Windows создается минималистичная виртуальная среда, которая эмулирует ядро Linux и на его базе есть возможность работать с контейнерами, именно такой подход используется в решении [13]. Ситуация эмуляции Windows на базе контейнеров Linux проблематична ввиду того, что Windows закрытая операционная система. Однако при использовании программу для эмуляции Windows API — Wine есть возможность эмулировать работу Windows. В любом случае в рамках данной работы в использовании данной операционной системы нет необходимости.

В поисках аналогичных решений был найден продукт от Google [14], который базировался на тех же вещах. Отличия заключаются в том, что:

- описываемая в работе модель исходит из того, что у нее нет собствен-

ных ресурсов;

- как следствие необходимо учитывать распределенную сеть узлов и упростить процесс настройки нового узла;

Глава 2. Работа параллельных приложений в распределенной среде

Перед созданием вычислительной системы необходимо провести обзор видов параллельных приложений. В данный момент существую деление всех параллельных приложений на два типа:

- Параллельные приложения с общей памятью.

  Данный подход используется в случаях, когда приложение работает на одном устройстве. В таком случае в каждый момент времени работы приложения, к каждого потока есть доступ к памяти, используемой всем приложением. Используя данный подход мы решаем проблему с общим доступом к данным, но при этом упираемся в производительность одного устройства. Данный подход использует библиотека OpenMP, которая позволяет с помощью директив препроцессора переработать существующий код в параллельный вариант. Небольшое количество специфичных функций для параллельного запуска позволяет без большого труда переработать подготовленный код и разработать его параллельную реализацию. При этом, стоит отметить, что алгоритмы параллельных вычислений не всегда эффективно применимы к существующему решению. Если приложение изначально спроектировано таким образом, что нет возможности выделить независимые параллельные блоки, то добавление соответствующих директив глобально не повлияет на производительность приложения. Однако, тем не менее данный подход показывает высокую эффективность в

случае, когда приложение изначально разрабатывается с возможностью разбиения на независимые участки выполнения в случае, если ресурсов устройства достаточно для достижения оптимальной производительности.

- Параллельные приложения с распределенной памятью.

  В данном случае, чаще всего, приложение выполняется на нескольких устройствах. Соответственно их оперативная память разделена физически и нет возможности организовать общий доступ к данным без использования дополнительных механизмов. Примером одной из таких библиотек является MPI. При использовании данной библиотеки разработчику предоставляется ряд функций, которые дают возможность обмениваться информацией между потокам и распределять задачи. Стоит отметить, что данный подход сложнее в реализации. Во-первых в данном случае нужна большая переработка исходного кода, чем в предыдущем случае. Во-вторых скорость обмена данными снижается. В первом случае мы ограничены скоростью работы оперативной памяти, во втором случае, помимо работы с оперативной памятью, также необходима работа с локальной сетью. Тем самым скорость обработки информации уменьшается. Помимо этого, необходимо произвести настройку всех машин и организовать их связность. В-третьих требуется дополнительное программное обеспечение. В частности для компилирования задач с использованием MPI используются отдельные компиляторы. Для запуска данных приложений, помимо библиотеки, необходима система, которая будет производить запуск задач, следить за ходом ее выполнения и распределять текущую нагрузку. Тем не менее, несмотря на возникающие трудности, для задач, которым требуется огромное количество данных и/или ресурсов данный подход является единственным. В данном случае есть возможность

объединять огромное количество вычислительных ресурсов для решения поставленной задачи.

В целом, подводя итог можно сказать:

- Если во время проектирования приложения, полученная оценка необходимых ресурсов находится в рамках одного устройства и нет необходимости привлечения большого числа ресурсов, то в этом случае можно разрабатывать однопоточное приложение. При этом стоит произвести грамотное разбиение общей задачи на более мелкие подзадачи на уровне архитектуры приложения. Для тех задач, которые являются независимыми, будет возможности применить подходы параллельного программирования с использованием соответствующей библиотеки. При этом глобальных проблем с внедрением подходов параллельного программирования проблем возникнуть не должно.

- Если решение поставленной задачи требует большого количества ресурсов и полученная оценка превосходит вычислительные ресурсы одного устройства, то в данном случае необходимо использовать подход с разделяемой памятью. И при этом необходимо изначально разрабатывать приложение с учетом особенности используемой системы. Помимо этого, стоит учитывать, что наличие кластера влечет за собой ряд задач, которые необходимо решать, например: безопасность, отказоустойчивость, надежность и т.д.

Помимо библиотеки MPI, которая была отмечена ранее, также стоит отметить такую систему распределенных вычислений как Hadoop[15]. Разработана на языке Java. Изначально данная система является проектом Apache Software Foundation [16]. Однако в ходе его развития существует ряд проектов, созданных сторонними разработчиками, для упрощения работы с Hadoop, например Cloudera Distribution including Apache Hadoop, Hortonworks Data Platform, MapR.

Основной идеей является применение подхода MapReduce основанного на следующих фазах:

- Map - первая фаза, в ходе которой происходит обработка входных данных. На основе них и выполняется распределение задачи обработки входных данных по узлам кластера. На выходе генерируется множество пар ключ-значение, которые передаются на следующие этапы.

- Sort - следующий шаг, в ходе которого все пары сортируются по ключу.

- Reduce - заключительный этап, в ходе которого происходит свертка полученных пар и вычисление результата. Распределение задачи выполняется исходя из ключей, переданных ранее. На каждом узле располагаются все пары с одним ключом.

Стоит отметить, что этап сортировки данных часто опускается в теоретических работах. Данная технология распределения задач имеет название MapReduce. Помимо этапов, описанных выше существуют и вспомогательные, например фаза combine - необязательная фаза в которой происходит обработка данных после этапа map. Похожа на заключительный этап свертки, но особенность ее заключается в том, что она выполняется на каждом узле перед сортировкой. Это позволяет перед отправкой произвести дополнительную обработку данных с целью оптимизировать процесс.

Система Hadoop позволяет реализовать данный подход, однако в качестве альтернативы можно использовать Spark, разработанный на Scala. Из особенностей можно выделить то, что все промежуточные операции производятся в памяти, в отличии от Hadoop, где они сохраняются на диск, данный подход выигрывает в производительности. Помимо этого в основе Spark находятся "ленивые вычисления то есть непосредственно вычисления производятся в момент, когда запрашивается результат, до этого создает-

ся последовательность действий необходимая, для получения результата. Тем самым Spark позволяет описать серию операций, которая может заключать в себе несколько вызовов map, reduce и других фаз, которые необходимы для получение результата, в частности можно производить некоторую фильтрацию данных.

Помимо этого существует такое решение как Tez. Он представляет задачу в виде направленного ациклического графа компонентов-обработчиков. Планировщик запускает вычисление графа и при необходимости динамически переконфигурирует его, оптимизируя под данные. Это естественная модель для выполнения сложных запросов к данным. Набольшее применение нашел в Hive, системе для написания SQL подобных запросов к данным [17].

Для реализации модели вычислений MapReduce на базе Hadoop используется целый набор программных продуктов решающих задачи разного вида.

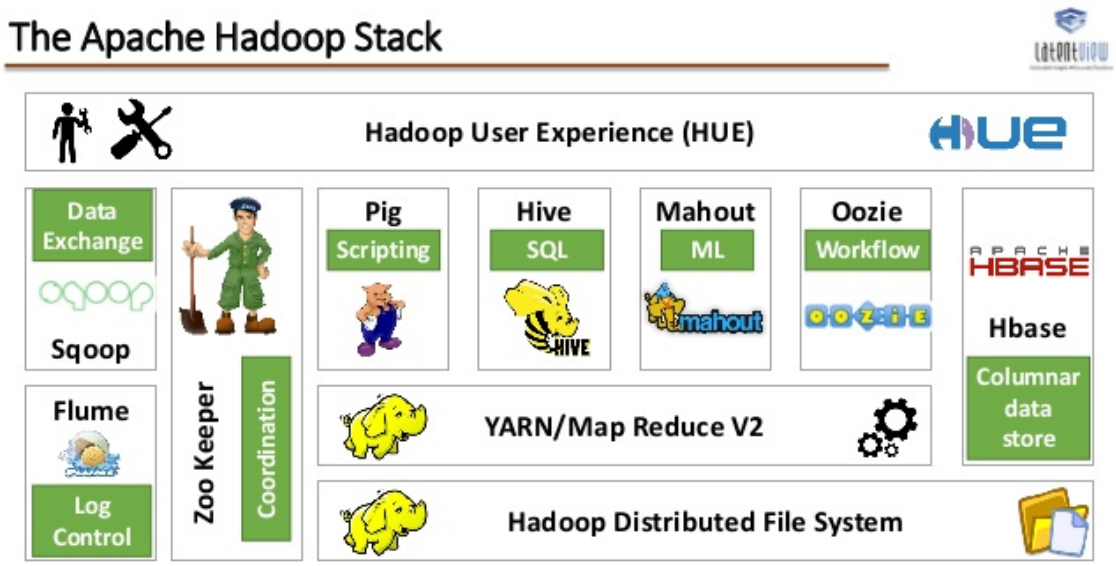

Рис. 2. Набор технологий, используемых совместно с Hadoop[18].

Полный обзор инструментов выходит за рамки данной работы. Рис. 2

наглядно показывает, насколько сложно разработать и спроектировать полноценную систему для организации распределенных вычислений. В ходе эксплуатации возникает множество дополнительных требований, решение которых может привести к разработке нового программного обеспечения. Каждый аспект работы с распределенной системой может вырасти в полноценную задачу. Нужно заранее учитывать, такие аспекты как работа с пользователя, работа с данными, хранение данных, анализ данных, оперативное вмешательство в процесс исполнения задачи. И проектировать систему так, чтобы либо разрабатываемая система могла полноценно решать данные задачи, либо имела возможность разработки расширений, которые возьмут на себя решение каждого из недоработанных аспектов.

Глава 3. Разработка распределенной вычислительной инфраструктуры для параллельных приложений

Для разработки распределенной системы нужно учесть несколько моментов:

- Необходимо создать модель управления ресурсами.

  Ввиду проектирования распределенной системы необходимо создать полную модель взаимодействия ресурсов в рамках системы. Необходимо учитывать такие моменты как: присоединение новых ресурсов к системе, взаимодействие между ресурсами и элементами системы, а также ситуации, когда ресурс выходит из строя или перестает быть ресурсом. Следует учитывать способы выполнения задач в распределенной системе (резервация ресурсов, вызов по требованию, вызов по расписанию). Помимо этого следует иметь ввиду неоднородность ресурсов их надежность и, наконец их технические характеристики и

роль в системе. Мощный ресурс может помочь в решении нескольких задач, однако его преимущество переходит в недостаток, в случае его выхода из строя, так как запущенные на нем задачи также прекратят свою работу. В некоторых системах предусматривается планировщик задач, который контролирует доступность ресурсов. Помимо этого он контролирует процесс запуска задач и доступным средствами пытается продолжить процесс вычислений после выхода ресурса из строя. Примеры хорошей архитектуры основанной на планировщике заданий можно увидеть в работах [19, 20, 21].

- Необходимо создать модель оптимального распределения загрузки ресурсов.

  Моделируемая система должна учитывать разнородность ресурсов. Необходимо некоторым образом разложить все ресурсы по некоторому базису, чтобы иметь возможность сравнить их и выбрать оптимальный набор ресурсов. Для решения этой задачи пользуются различными методами, например в данной работе [22] используется множественный критерий для подбора оптимальных ресурсов для запуска задач, основанный на модели принятии решений Promethee.

- Необходимо учитывать многопользовательскую модель.

  Моделируемая система должна учитывать, что она будет использоваться множеством пользователей и при моделировании системы нужно найти баланс между необходимыми ресурсами и конечными пользователями. Ситуация, если пользователями будет ощущаться недостаток ресурсов и они не смогут выполнять свои задачи повлечет за собой отток пользователей. С другой стороны большое количество неиспользуемых ресурсов влечет за собой расходы на содержание, что уменьшает экономическую эффективность разрабатываемой модели. Система должны иметь возможность своевременно решать задачи,

запущенные пользователями.

В данной работе решена часть задач связанных с проектированием системы.

- Во-первых решена задача управления ресурсами.

  Несмотря на множество способов управления ресурсами создание максимальной отказоустойчивой системы с планировщиком задач не являлось основной частью данной работы. Акцент на запуске приложения с конфигурруемыми параметрами. При этом основным большим приоритетом обладает запускаемая задача а не пользователь. Ситуация, когда программа не отработает или отработает с ошибкой является нормальным поведение в контексте данной задачи. Это дает дополнительную информацию о поведении программы при данном наборе параметров. Исходя из этого было принято использовать модель запуска распределенных задач по требованию. Т.е. в момент запуска задачи система находит ресурсы, которые удовлетворяют запросам пользователя и использует их для запуска.

- Во-вторых решена задача неоднородности предоставляемых ресурсов.

  Как говорилось ранее любая система распределенных вычислений должна учитывать, что используемые ресурсы могут быть разнородны. Необходимо решить вопрос о присоединении таких ресурсов к системе. Решение этой задачи достигается путем использования системы управления контейнерами Docker. Имея на каждом узле контейнеры, созданные из одного образа системы достигается однородность используемых. При это однородность достигается как на уровне операционной системы контейнеров, так и на используемых ресурсов. При этом есть возможность доработки системы с целью моделирования ситуации с разнородными ресурсами. Это дает большие возмож-

ности для исследования задач, однако в данной работе рассматривается вариант, когда все контейнеры, используемые в качестве узлов кластера являются однородными.

- В-третьих частично решена задача оптимального распределения загрузки ресурсов.

  При запуске распределенных задач ресурсы выбираются таким образом, чтобы удовлетворять критерием пользователя. Тем самым при запуске множества задач происходит распределение загруженности по ресурсам без участия дополнительных инструментов или дополнительной логики. Недостатком данного подхода, а тем самым и частичное решение задачи происходит из-за того, что возможны ситуации при которых пользователи подберут параметры таким образом, что нагрузка системы будет распределена неравномерно. И в случае выхода узла из строя возможна ситуация некорректного завершения задач. Данный момент планируется развивать в дальнейшем, при этом он не является приоритетным. В случае выхода узла из строя предполагается перезапуск задачи и подбор новых ресурсов для вычислений.

- В-четвертых в модели учитывается наличие множества пользователей.

  В рамках данной модели решается задача с избытком ресурсов. Так как не происходит дополнительное обслуживание ресурсов и они не принадлежат владельцу системы. Однако ситуация недостатка ресурсов остается. Данную проблему можно нивелировать покупкой дополнительных ресурсов, принадлежащих непосредственно владельцу системы. Данный вариант не соответствует идеологии модели и его рекомендуется применять только в крайнем случае, при условии острого недостатка в ресурсах. Другим способом является распространение идей данной системы и реклама в некотром ее роде. С целью привле-

чения людей, которые будут готовы предоставить свои ресурсы. Также возможна ситуация, когда клиенты будут предоставлять свои ресурсы. Для упрощения процесса добавления ресурса предполагается использование предварительно настроенных образов системы. Если предлагаемое решение получит популярность, то необходимый образ можно будет получить из общего репозитория [23]. До этого есть возможность использовать удобный интерфейс, предоставляемый докером [24], для создания необходимого образа.

Ниже представлена непосредственно схема разрабатываемой модели:

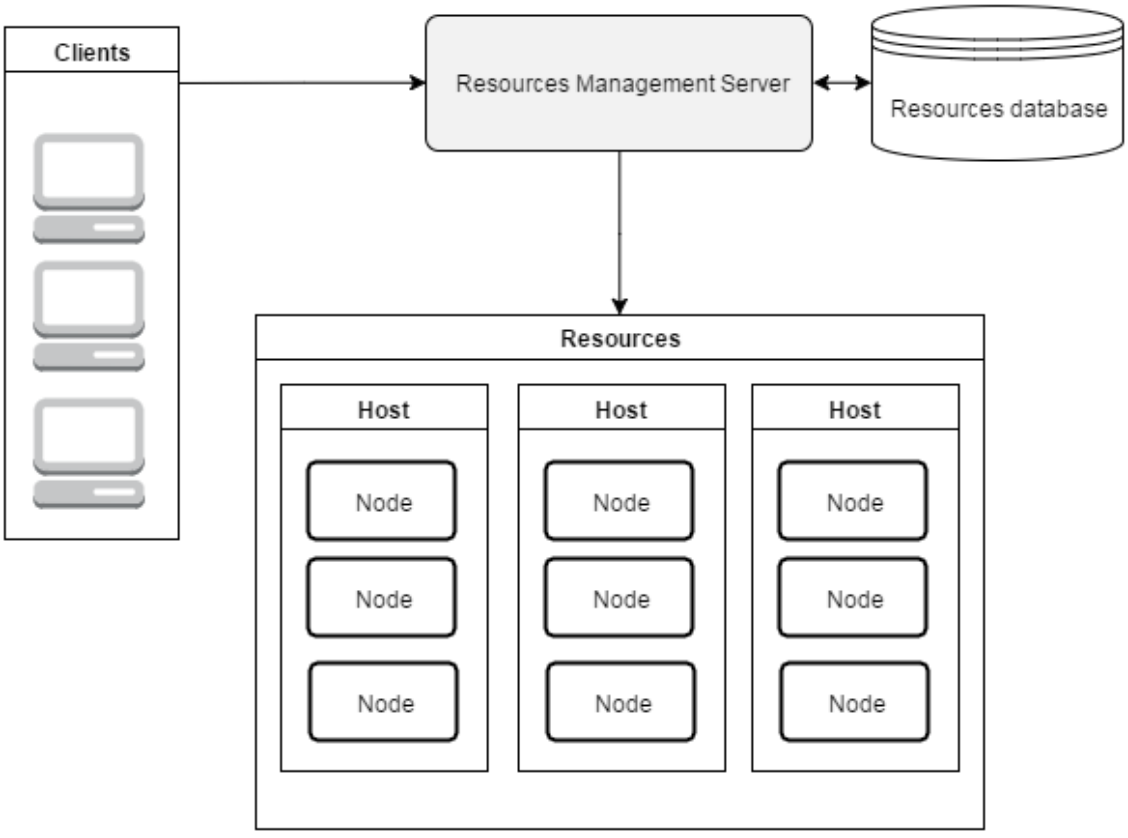

Рис. 3. Схема модели.

На рис. 3 представлена общая структура модели. В ней присутствуют:

1 Ресурсы.

Подразумевается, что пользователи будут сами предлагать свои ресурсы. В случае, если некто захочет предоставить свои ресурсы, он

создает контейнер с настроенным образом системы и предоставит системе управления ресурсами (СУР) доступ. После этого, пока ресурс доступен СУР, он будет использоваться в вычислениях. В случае необходимости на одном ресурсе может быть запущено несколько вычислительных узлов работающие независимо друг от друга. Считается, что все ресурсы доступны друг для друга. Подбор ресурсов для конкретной задачи осуществляется путем ввода пользователем необходимых параметров для задачи и фильтрация согласно этим данным. использование контейнеров позволяет из разнородных ресурсов создать однородные вычислительные узлы. При это для создания контейнеров более пригодны операционные системы семейства Unix.

2 Пользователи.

В данном случае мы разделяем систему от ресурсов. В качестве пользователей подразумеваются люди, которым необходимо произвести расчет. Для этого они предоставляют системе все необходимые данные для решения своей задачи. Подразумевается,что пользователи будут взаимодействовать с СУР через web-интерфейс и не будут иметь напрямую доступ к ресурсам. Тем самым персонал, которому необходимо произвести параллельные расчеты, сможет меньше времени уделять техническим аспектам и больше времени потратить на оптимизацию и устранение ошибок работы программы. Кроме того, данный подход позволяет усложнить архитектуру кластера с целью увеличения отказоустойчивости, безопасности, быстродействию и т.д. Так как вся техническая реализация скрыта от пользователя и с его точки зрения ничего не изменится.

3 Система управления ресурсами.

Представляет собой программный комплекс, который получает от пользователя все данные, которые необходимы для запуска задачи и

параметры системы, при которых необходимо выполнить программу. Получив всю необходимую информацию, данная система подбирает ресурсы, которые удовлетворяют заданным критериям, запускает на них вычислительные узлы. По завершению программы вся информация выдается пользователю и все вычислительные узлы очищаются. После этого, освобожденные ресурсы можно повторно использовать повторно для решения других задач с другими характеристиками. Для хранения и обработки данных о ресурсах предполагается использовать систему управления базой данных.

Ниже представлен ряд сценариев, который описывает возможны взаимодействия в системе.

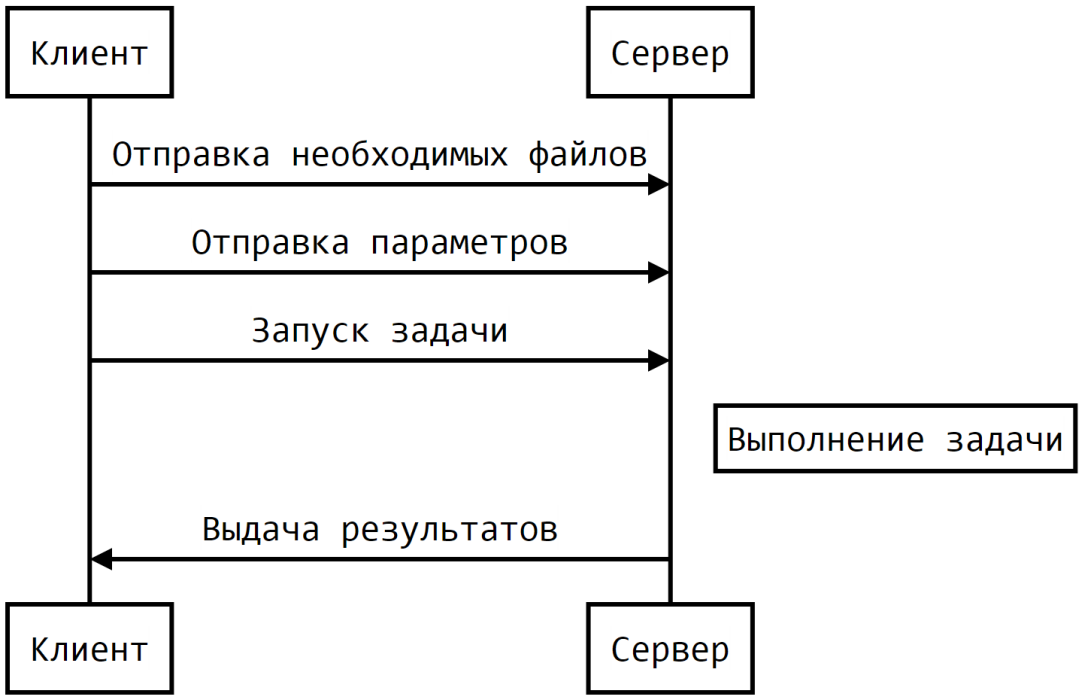

Рис. 4. Взаимодействие клиента и сервера.

Подразумевается, что программная реализация модели предоставит пользователю web интерфейс в котором пользователь введет параметры системы, которые ему необходимы, а также приложит исполняемый файл

и файлы с данными. После этого он инициирует запуск задачи и по факту завершения, получит результаты расчетов. Для вывода результатов предполагается выводить стандартные потоки вывода stdout и stderr. При этом клиент взаимодействует только с системой управления ресурсами.

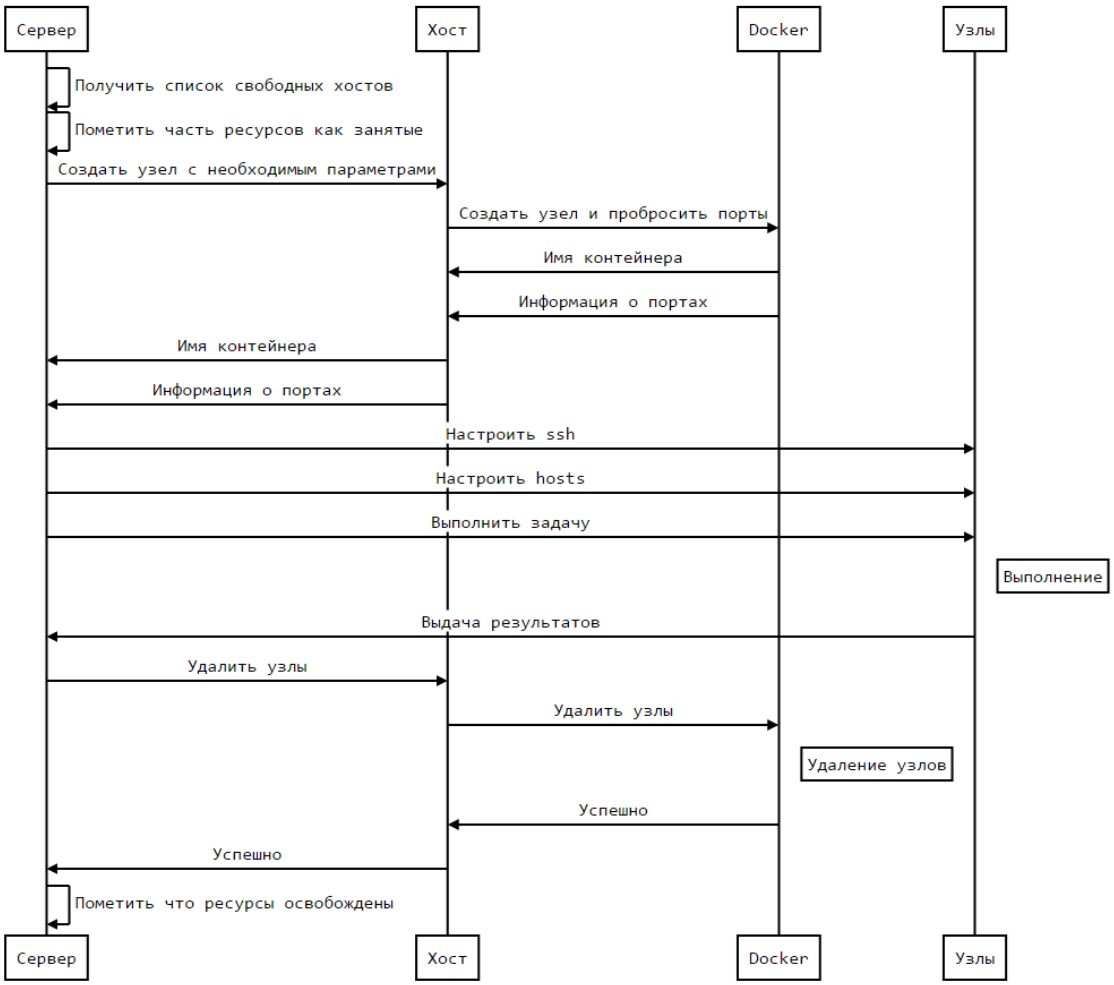

Рис. 5. Взаимодействие сервера и ресурсов.

Вся логика, которая происходит при выполнении задачи должна быть скрыта от пользователей, чтобы не вызывать неудобств. Все используемые хосты настроены и содержат в себе:

1 docker;

2 образ системы;

3 скрипты для запуска.

В рамках данного сценария выбираются ресурсы, которые удовлетворяют заданным критериям. После этого создаются контейнеры, которые будут использоваться в качестве вычислительных узлов. Далее производится настройка узлов с целью объединения их в единый кластер. После этого на кластере выполняются необходимые вычисления, результаты которых в конечном итоге выдаются пользователю. И по окончанию вычислений освобождаются занятые ресурсы.

Далее рассматриваются взаимодействия с базой данных.

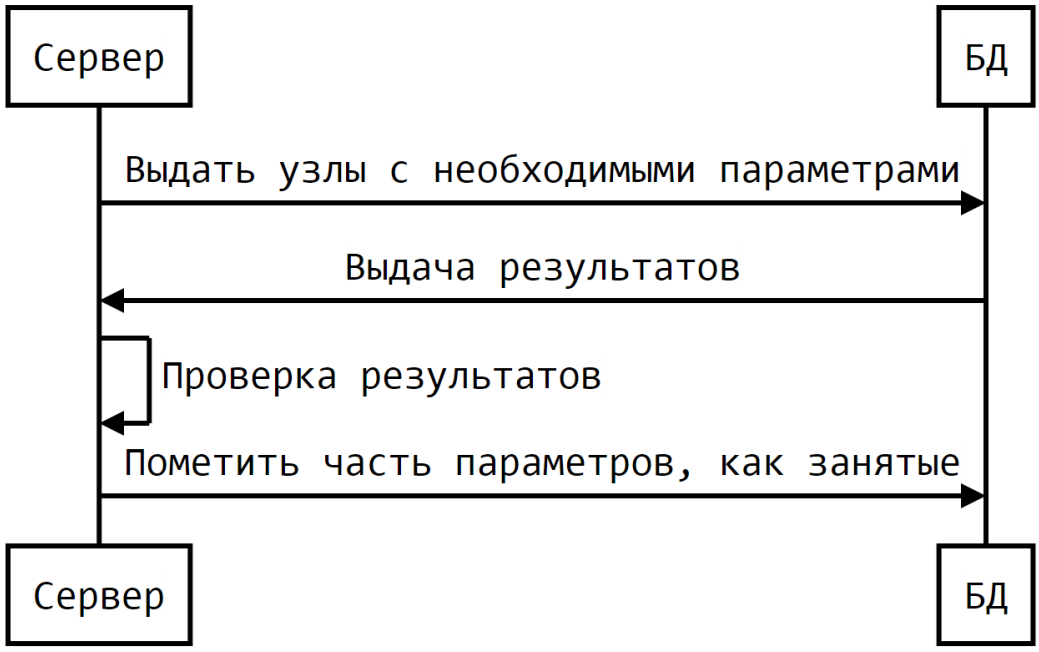

Рис. 6. Взаимодействие сервера и БД.

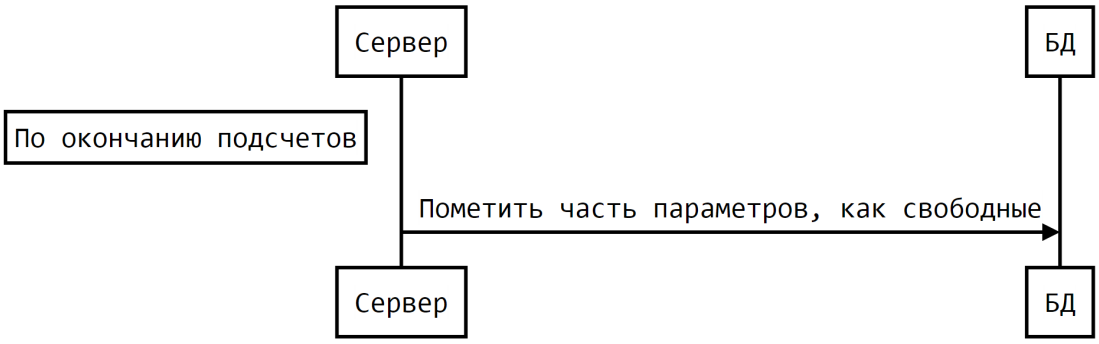

Рис. 7. Взаимодействие сервера и БД.

Взаимодействия с базой данных происходят при: добавлении и удалении ресурса. В этом случае информация заносится в базу данных или, соответственно, удаляется. После запуска задачи пользователем сервер ищет узлы, которые подходят по запрашиваемым параметрам. Если подходящих узлов нет или их меньше, то выполнение сценария невозможно. В случае, когда ресурсов достаточно, берется необходимое количество хостов и часть их ресурсов помечается как занятое. И по завершению задачи и удалению вычислительных узлов, занятая часть ресурсов для данной задачи помечаются как свободные.

Еще одна особенность данной модели заключается в ресурсах. В отличии от классической ситуации, когда все ресурсы находятся под управлением владельца сервера. Предполагается, что ресурсы будут предоставляться на добровольной основе.

Ввиду того, что серверу требуется использовать системные процедуры среди всех языков программирования, предпочтение отдается скриптовым языкам и языкам которые ориентированы на работу с системой на низком уровне.

В рамках данной модели все ресурсы системы рассматриваются как множество системных параметров. Тем самым мы имеем некоторый базис, который позволяет сравнивать ресурсы между собой и подбирать необхо-

димую конфигурацию. Рассмотрим эти параметры:

- Пропускная способность сети.

  Для ее ограничения используется стандартный инструмент Linux tc, он позволяет управлять передаваемыми пакетами и в частности ограничивать пропускную способность исходящего потока пакетов.

- Память.

  Для ограничения памяти используется стандартные инструменты предоставляемые контейнерами Linux.

- CPU.

  Для ограничения CPU предполагается использовать стандартные инструменты предоставляемые контейнерами Linux.

Стоит отметить, что такие параметры как память и CPU не зависят от других элементов системы. Помимо этого, так как ресурсы предоставляются на добровольной основе, то эти значения задаются клиентом в момент передачи ресурсов системе. Тем самым для получения этих значений достаточно единожды их сохранить и использовать в дальнейшем с учетом потраченного ресурса. В случае с сетью ситуация иная. Пропускная способность сети помимо того, что является относительным критерием, имеет также свойство изменяться во времени. Для получения актуальной информации о данном ресурсе, в рамках данной системы, планируется производить проверку каналов связи между ресурсами и выполнять подбор ресурсов с учетом возможных изменений параметров канала связи до 10%. При этом подразумевается, что решение задачи допустимости не стоит. Система исходит из того, что все ресурсы доступны в глобальной сети Internet и доступны друг другу. Ситуация, когда часть ресурсов недоступна не окажет критического воздействия на систему, однако эта информация позволит произвести оптимизацию процесса опроса узлов, тем самым ускорить про-

цесс подбора необходимых ресурсов. В данном случае этого происходить не будет и ситуация, когда часть узлов недоступна другими приведет к увеличению времени подбора оптимального набора ресурсов. В случае, если модель доступности ресурсов далека от полного графа рекомендуется произвести доработку системы опроса и построения сети ресурсов с учетом частных особенностей конкретной сети ресурсов.

Глава 4. Применение модели

В рамках данной работы был реализован прототип моделируемой системы.

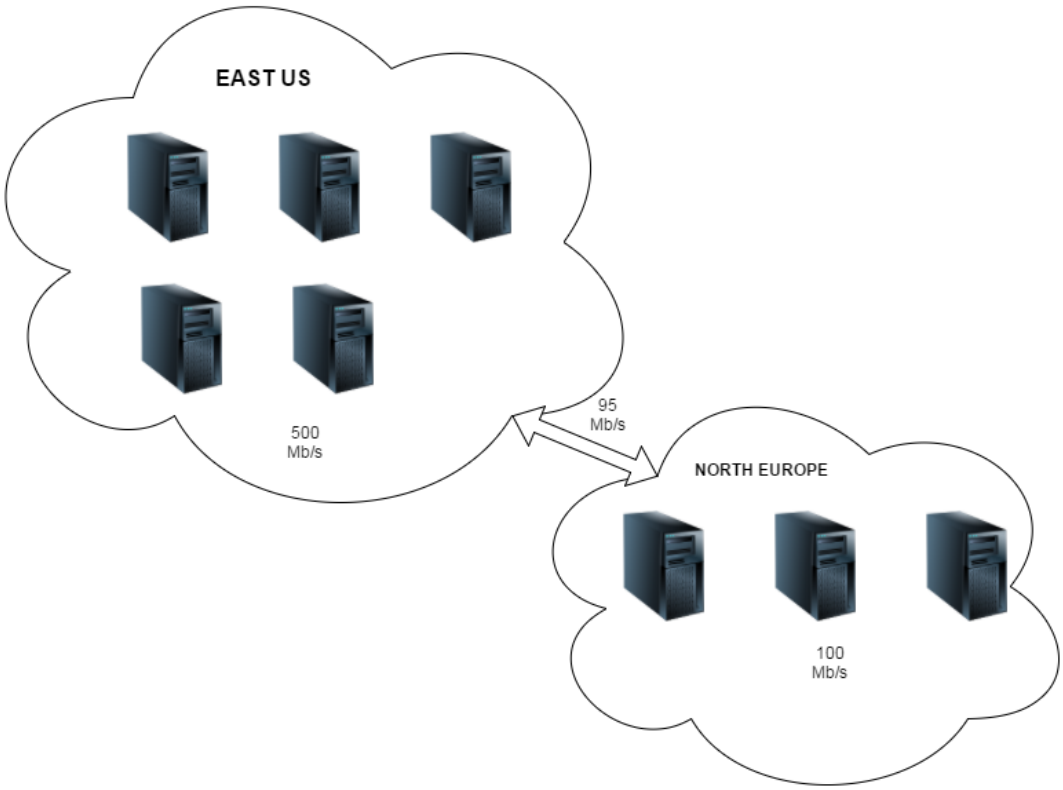

Рис. 8. Ресурсы прототипа.

В качестве ресурсов использовались виртуальные машины, развернутые в облаке Windows Azure. Это облачная платформа предоставляемая Windows. В рамках этой платформы реализовано две облачные модели:

- Platform as a Service (PaaS);
- Infrastructure as a Service (IaaS).

Данная платформа обладает огромными возможностями. В данной работе нас интересует создание виртуальных машин с системой docker. Для этого использовались виртуальные машины на базе Ubuntu 14.04 LTS. На которую уже происходила установка docker и последующая работа. В рамках данной работы использовались различные мощности ресурсов:

Тарифный план	A1	D1
Количество ядер	1	1
Количество дисков данных	1	2
Оперативная память (GB)	1.75	3.5
Макс. операций ввода-вывода в сек.	2×500	2×500
Балансировка нагрузки	✓	✓
Автомасштабирование	✓	✓

В серии экспериментов было задействовано восемь машин, при этом пять из них располагались в Восточной Америке, а оставшиеся три в Северной Европе.

Для тестирования прототипа использовался Numerical Aerodinamic Simulation(NAS) Parallel Benchmarks, который базируется на задачах научно-исследовательского центра NASA [28]. Данный набор программ разрабатывался для вычисления производительности кластеров. Методики, которые используются при подсчете производительности являются теоретически обоснованными. В основе используется методы, применяемые при решении задач аэродинамики а также при моделирование механических систем. Существуют и другие инструменты для подсчета производительности кластера в частности LINPACK, в основе которого лежит параллель-

ное решение систем линейных уравнений большой размерности. Однако инструмент использовавшийся в данной работе более приближен к научной реальности, так как основывается на решениях задач, которые необходимы в данный момент. И тем самым являются менее синтетическими, по сравнению с решение произвольно системы уравнений большой размерности. В качестве программ, которые отражают изменения системных параметров был выбраны некторое, которые входят в состав пакета NAS Parallel Benchmarks, а именно:

- FT — Использует быстрое дискретное преобразование Фурье и требует коммуникации между всеми узлами.
- CG — Реализует расчет сопряженных градиентов, использует случайный доступ к памяти и коммуникацию между узлами.
- MG — Реализует многосеточный метод на последовательности поверхностей, использует различные виды коммуникаций и интенсивно использует память.

Для реализации программной части, в качестве основного языка программирования выбран Python. Во-первых это скриптовый язык, который позволяет использовать систему на низком уровне. Во-вторых простота языка позволяет ускорить процесс разработки. В-третьих есть возможности расширить прототип и использовать его в основе серверной части, используя web framework, например, django. Помимо этого данный язык содержит огромное количество библиотек, которые в текущий момент позволили решить все возникшие проблемы. Помимо этого данный факт упростит развитие системы в дальнейшем. В качестве версии Python использовалась версия 2.7.

После разработки прототипа и настройки ресурсов, необходимо было произвести проверку системы. Для этой цели был проведен ряд вычис-

лительных экспериментов с различными системными параметрами. Далее производился запуск контрольных задач по завершению работы которых было получено время выполнения каждой программы. Результаты серии экспериментов приведены ниже.

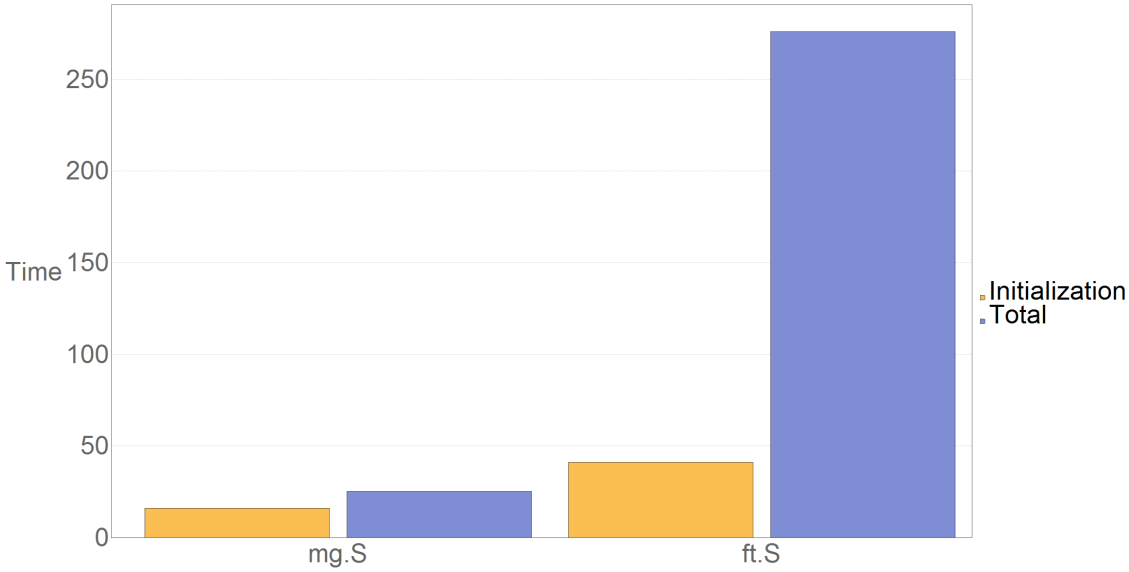

Рис. 9. Память 256 MB, сеть 100 kbit/s.

Используя данные ограничения отработали только две задачи. Другие же не смогли запуститься при данных параметрах.

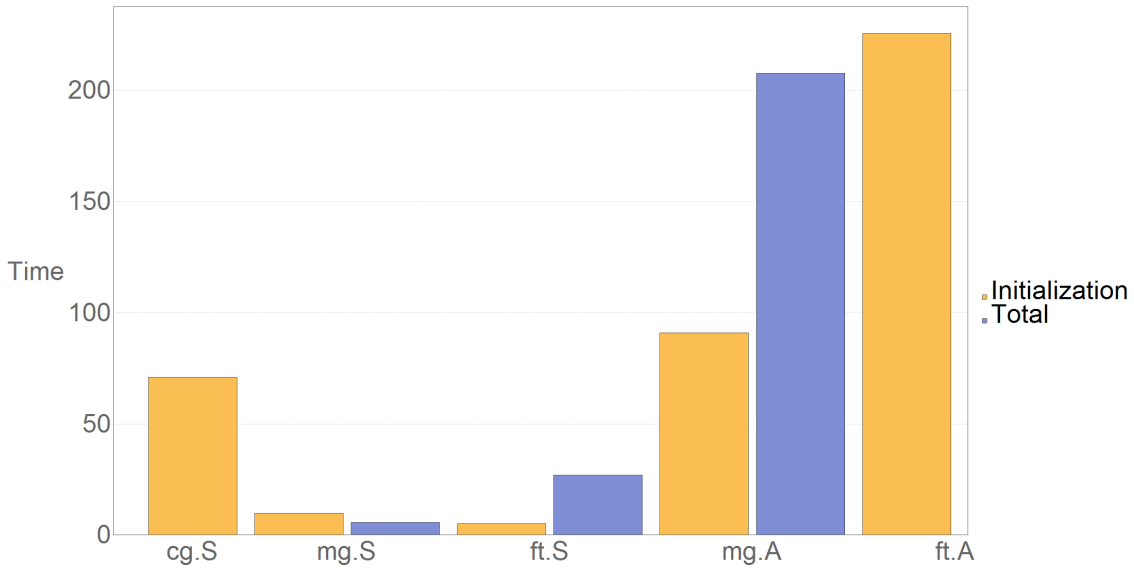

Рис. 10. Память 512 MB, сеть 1024 kbit/s.

При ограничениях использованных на рис. 10 запустились и отработали все задачи класса S. Однако при запуске задач класса A успешно закончила выполнение только задача mg. При этом задача ft данного класса выполнила инициализацию. Вычислила все итерации и на этапе финального подсчета данных перестала отвечать.

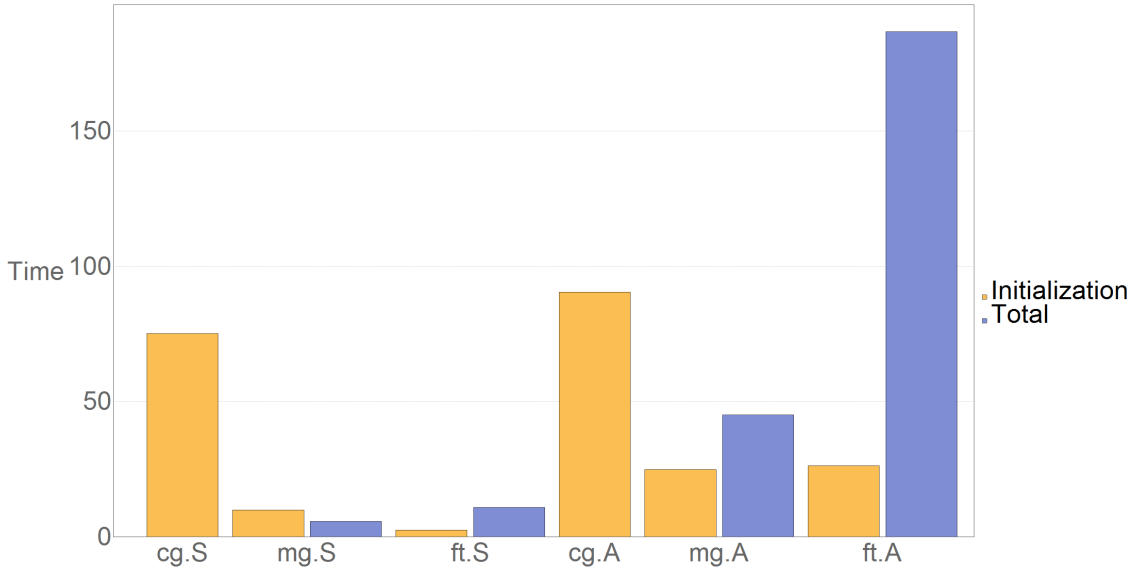

Рис. 11. Память 1024 MB, сеть 10240 kbit/s.

При данном наборе параметров все задачи успешно отработали и теперь есть возможность подвести предварительные итоги по всем трем конфигурациям.

Как видно из гистограмм, вводимые ограничения влияют на время выполнения программ, в частности это можно заметить на примере задачи ft размера S. При первом ограничении скорость ее выполнения составила 276.06 секунд, при второй конфигурации 26.98, при третьей 10.91. Стоит отметить, что не все задачи запустились при заданных конфигурациях например задача ft размера A на первой конфигурации не запустилась, а на второй конфигурации запустилась, выполнила этап инициализации и перестала отвечать. Теперь есть возможность провести более детальный анализ и подобрать оптимальные параметры для каждой из задач. Однако перед этим был произведен эксперимент параллельного запуска двух задач.

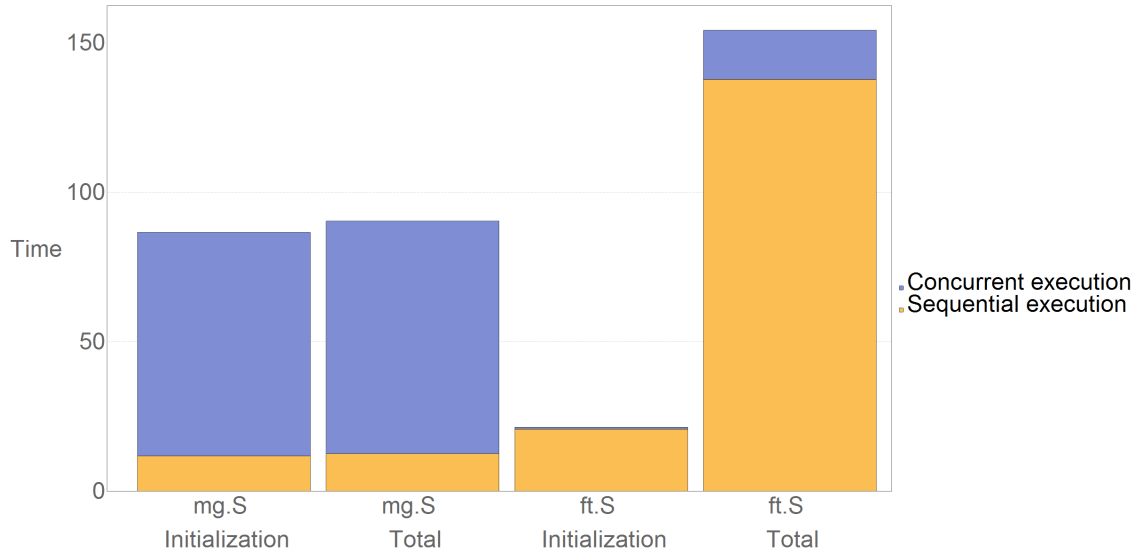

Рис. 12. Память 512 MB, сеть 200 kbit/s, одновременный запуск на одинаковых контейнерах.

Ограничение по памяти 512 MB, ограничение по сети 200 kbit/s. На гистограмме представлены два случая запуска, последовательного и параллельного, на одних и тех же контейнерах. Как можно видеть из гисто-

граммы, время выполнения программ заметно отличается, по сравнению с последовательным запуском. Ниже представлен модуль разности последовательного выполнения и параллельного в случае, когда задачи выполнялись на разных контейнерах.

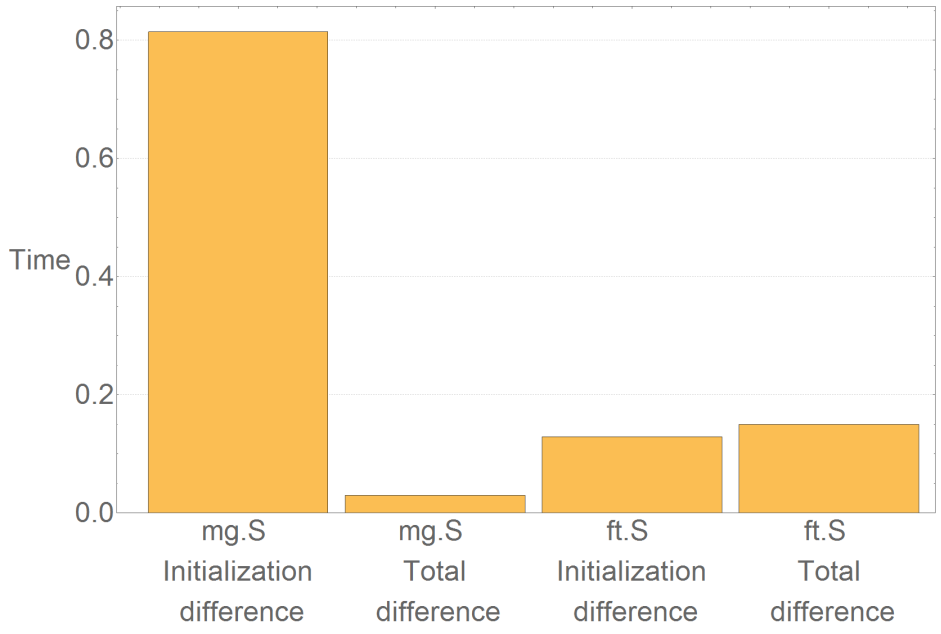

Рис. 13. Память 512 MB, сеть 200 kbit/s, одновременный запуск на разных контейнерах.

Как видно из рис. 13, помимо решения основной задачи было достигнуто свойство изолированности. К сожалению, из-за того, что эксперименты проводились на базе виртуальных машин, поэтому имеют место накладные расходы в вычислениях.

Далее была взята задача ft размера S и проведен ряд экспериментов, чтобы вычислить ее оптимальные параметры запуска. Сперва была зафиксирована память в 100 MB и в качестве изменяемого параметра обозначена пропускная способность сети.

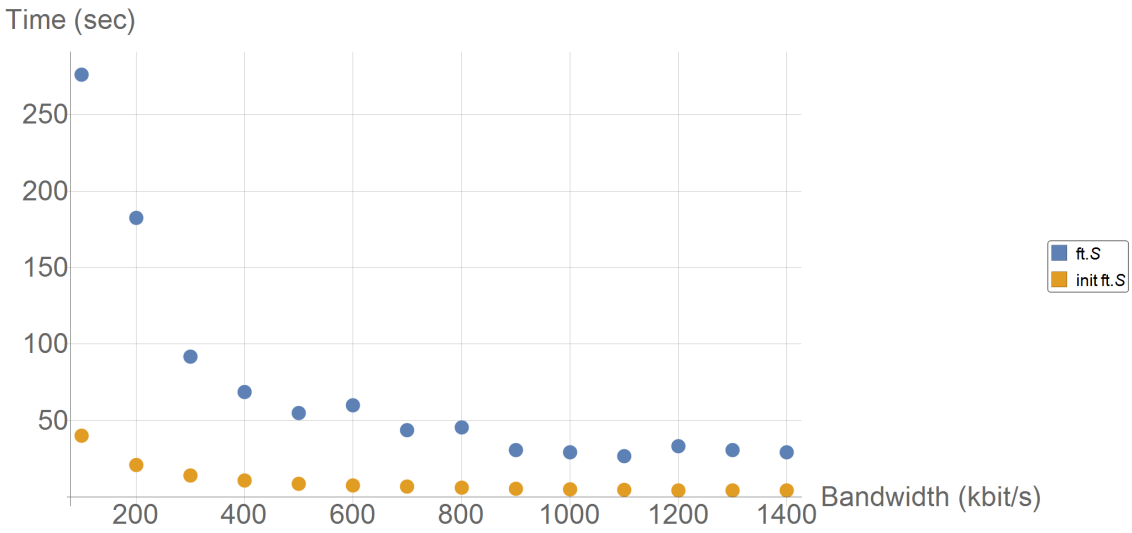

Рис. 14. Задача ft, размер S, зависимость от параметров сети.

Как видно из рис. 14, при ширине полосы пропускания больше 1 MB/s скорость работы программы не уменьшается. Используя эти данные построим зависимость изменения времени от операционной памяти для данной задачи.

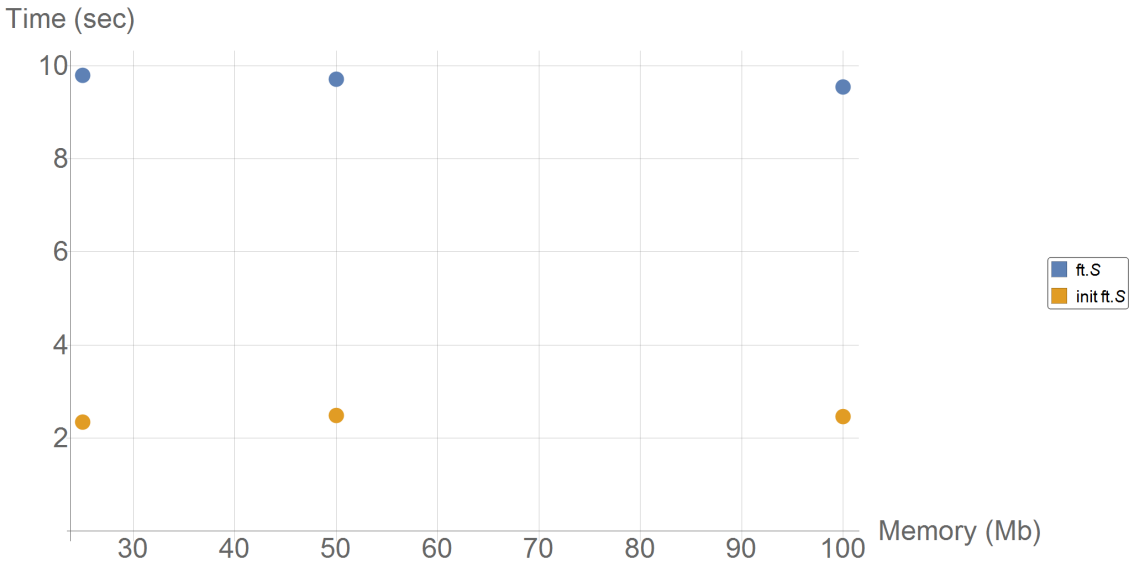

Рис. 15. Задача ft, размер S, зависимость от памяти.

Изменения по памяти не ускорили программу, выяснилось, что задача отказывается выполняться, когда объем оперативной памяти меньше 16

MB.

Далее рассмотрим задачу ft размера A, используя такой же подход вычислим сперва оптимальные параметры сети.

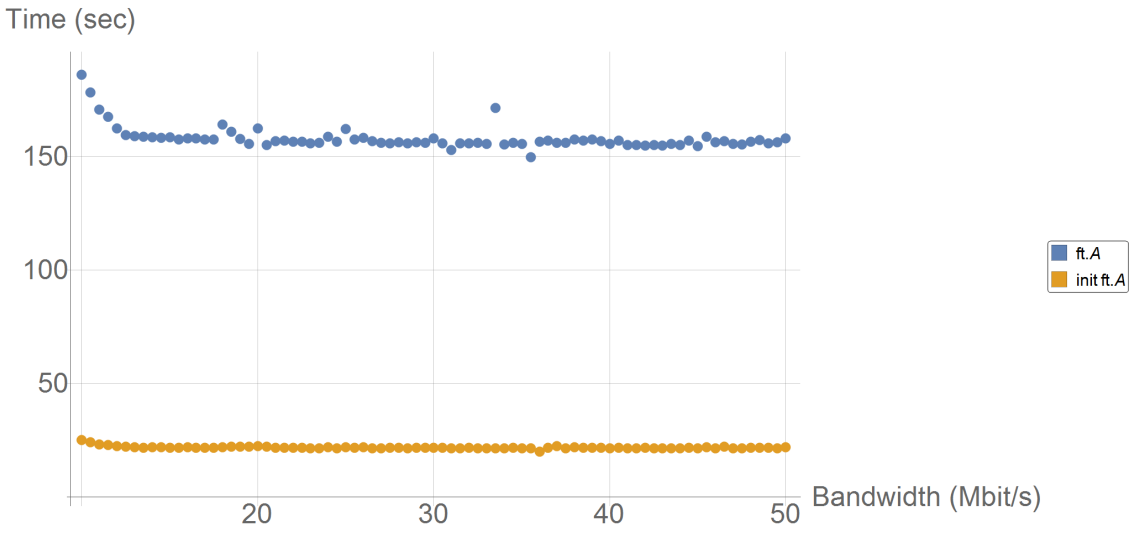

Рис. 16. Задача ft, размер A, зависимость от параметров сети.

Как видно из графика начиная с ширина полосы пропускания равной 12,5 Mb/s скорость выполнения программы остается примерно на том же уровне. Ниже представлена зависимость скорости выполнения от ограничений, накладываемых на память.

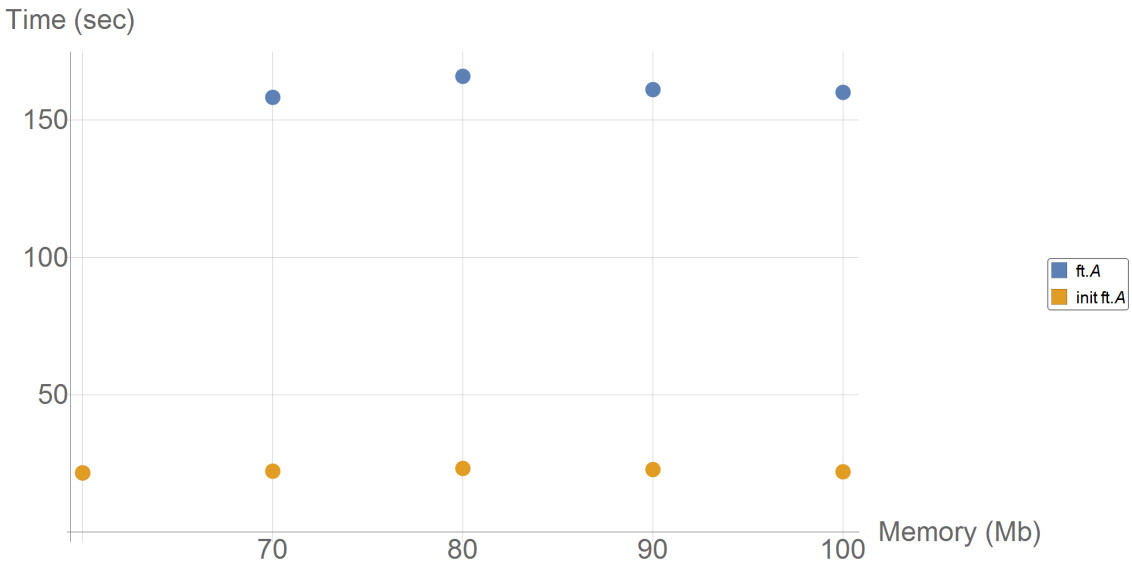

Рис. 17. Задача ft, размер A, зависимость от памяти.

Как видно из графиков, увеличение памяти не влияет на скорость выполнения, если задача запустилась. Рекомендуемое ограничение по памяти примерно 70 MB. Далее рассматривается задача mg размера S.

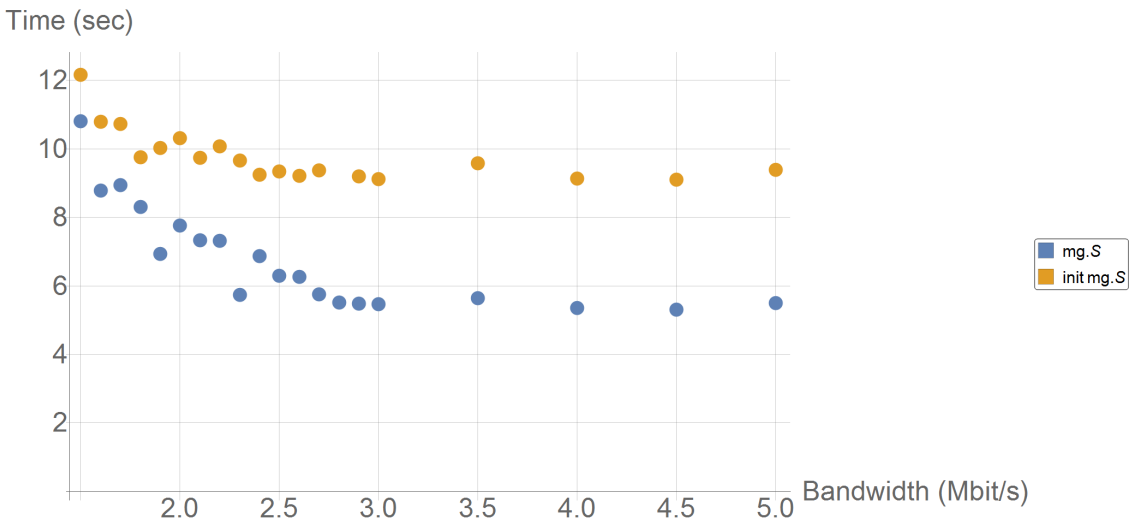

Рис. 18. Задача mg, размер S, зависимость от параметров сети.

Ограничение по памяти 512 MB, как видно из графика скорость выполнения практически не меняется при ограничении полосы пропускания от 3 MB/s. Зафиксируем оптимально положение сети и посмотрим зависимость от памяти.

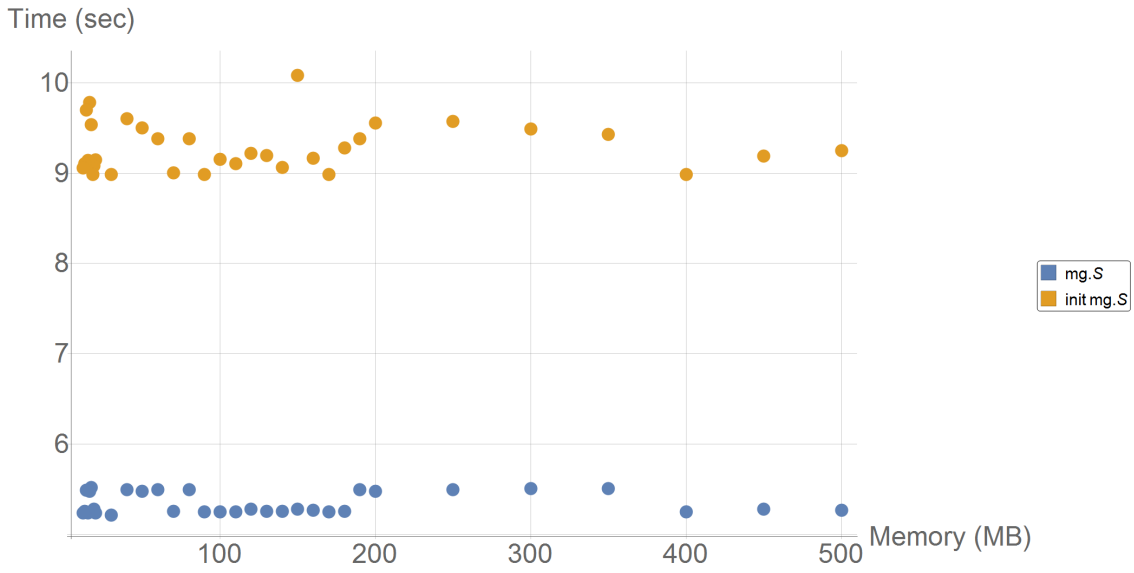

Рис. 19. Задача mg, размер S, зависимость от памяти.

Как видно из графика, если задаче хватает памяти для успешного выполнения, то изменения памяти мало влияют на время работы задачи. Тем самым можно взять оптимальную память для выполнения равную 12 MB.

Теперь рассмотрим mg размера A, зафиксировав память на отметке 512 MB получим следующее:

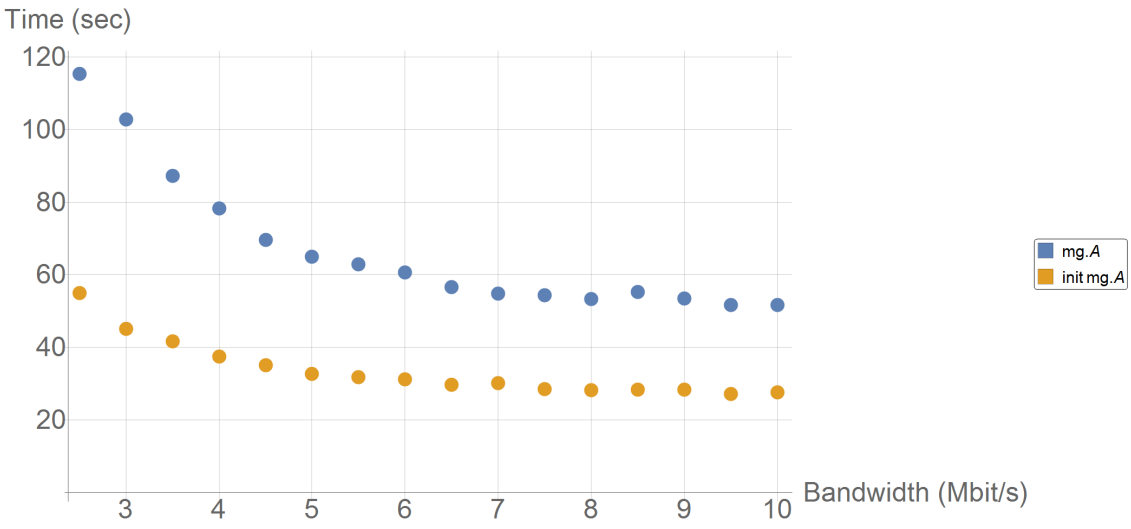

Рис. 20. Задача mg, размер A, зависимость от параметров сети.

Как видно из рис. скорость выполнения остается на том же уровне начиная с 7000 kbit/s. Используем полученные данные, для анализа оптимального параметра по памяти.

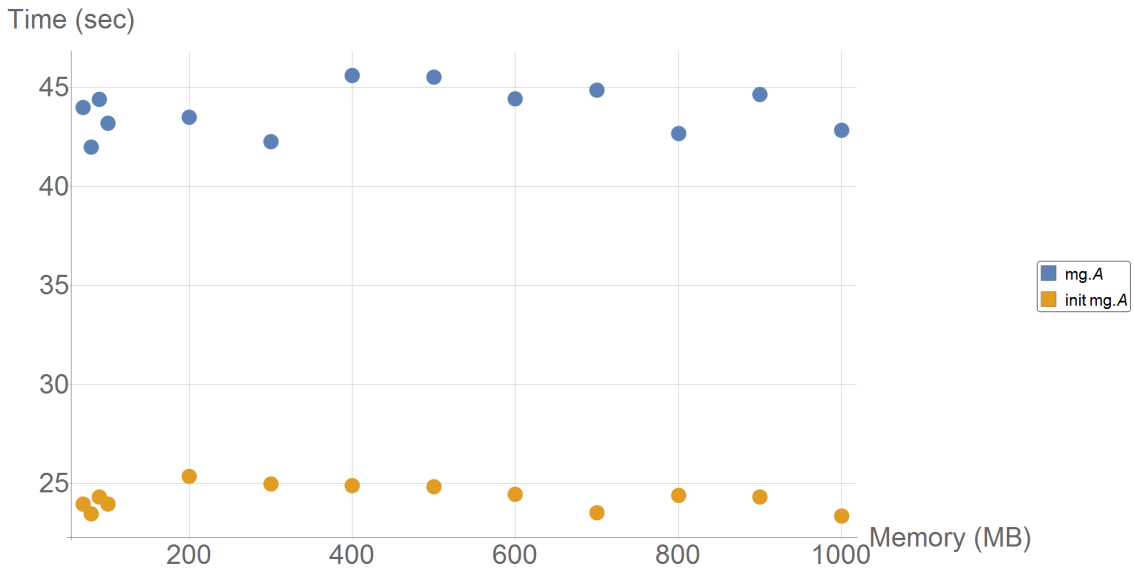

Рис. 21. Задача mg, размер A, зависимость от памяти.

Как видно из рис. 21, ограничения по памяти также не оказывают видимого воздействия, при условии, что программе хватает памяти для завершения работы. Тем самым в качестве оптимального значения можно выбрать минимальный объем памяти, при котором программа завершается корректно. Данный уровень примерно равен 70 MB.

Дополнительно был произведен ряд экспериментов, для демонстрации того, что вычисления производятся изолировано друг от друга.

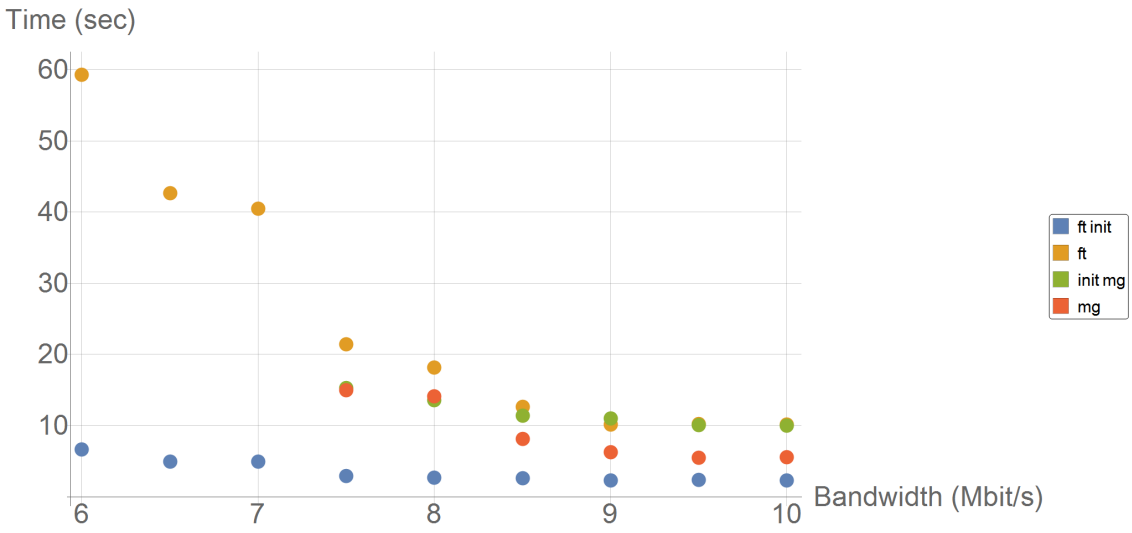

Рис. 22. Борьба за ресурсы.

На рис. 22 смоделирована ситуация, когда две задачи запускаются одновременно используют общие ресурсы, находясь в одних и тех же контейнерах. На кластерах, для более эффективного распределения ресурсов используется дополнительное программное обеспечение. В нашем же случае две задачи вступили в борьбу за ресурсы, что сказалось на их производительности.

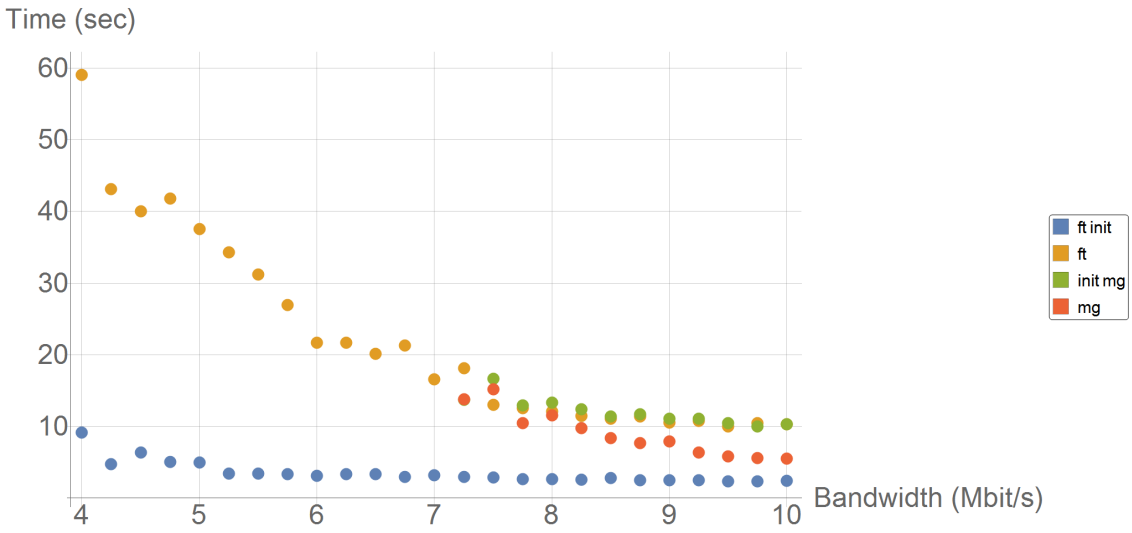

Рис. 23. Независимое выполнение.

На рис. 23 смоделирована ситуация, когда две задачи запускаются од-

новременно и используют различные ресурсы, находясь в разных контейнерах. Это позволяет более эффективно распределить ресурсы и исключить сценарий, когда происходит борьба за ресурсы.

Как видно из представленных выше графиков, поведение задач схоже. Однако при эмуляции сценария, когда приложения делят между собой ресурсы график сдвинут относительно оси абсцисс. Тем самым для более эффективного запуска задачи требуется больше ресурсов.

Помимо этого в тестовом режиме была добавлена интеграция с базой данных. В качестве системы управления базам данных были выбраны следующие:

- Microsoft SQL;
- MySQL;
- Elasticsearch.

Интеграция прошла успешно. При тестировании выяснилось, что No-SQL система управления базами данных Elasticsearch несмотря на отсутствие структуры данных работает на порядок быстрее конкурентов и при этом ее использование не требует наличие в системе дополнительно драйвера для взаимодействия. В частности в системах Linux все запросы можно формировать используя системную утилиту curl. Данный факт упрощает внедрение и среди предложенных вариантов Elasticsearch рекомендуется к использованию.

Глава 5. Результаты

Анализируя полученные результаты, можно определить оптимальные параметры для следующих задач:

	Память (MB)	Полоса пропускания (MB/s)
mg.S	12	3
ft.S	16	1
mg.A	70	7
ft.A	70	13

Изменения параметров влияют на время выполнения задач. Помимо этого есть возможность запускать несколько задач используя одни и те же ресурсы. В отличии от классической модели кластера в данном случае задачи исполняются изолировано и практически не влияют друг на друга. Стоит отметить, что эксперименты производились в облаке на виртуальных машинах, т.е. результаты менее точны, по сравнению с результатам, полученными на аппаратной основе. Однако наличие распределенной системы узлов позволило приблизить процесс проектирования и прототипирования к реальности. В частности была выявлена ситуация, когда политика безопасности сети, в которой находится ресурс, запрещает доступ. Анализируя использованные задачи можно придти к выводу, что у каждой из задач присутствует минимальный порог памяти. Если задачи предоставить памяти меньше, то у нее не хватит ресурсов для корректного завершения работы. При этом, если памяти достаточно, то дальнейшее увеличение памяти не влияет на производительность. При этом влияние, оказываемое изменением параметров пропускной способности сети более заметно. В рамках данной работы рассматривалась небольшие по ресусозатратам задачи. Это позволило произвести большее количество запусков, тем самым более подробно

изучить поведение приложения.

Поставленная цель была достигнута частично, несмотря на то, что поставленные задачи были выполнены и есть рабочий прототип не на все рассматриваемые параметры были введены ограничения. В данный момент не решена задача ограничения CPU, ввиду особенностей ресурсов.

Заключение

В рамках данной работы необходимо было решить поставленные задачи:

- Провести обзор предметной области.
  Обзор предметной области произведен. Необходимые аспекты освещены.

- Выбрать основу для виртуальной вычислительной архитектуры;
  Произведен обзор возможных вариантов, в качестве рабочего было решено использовать контейнеры приложений.

- Разработать модель для выявления оптимальных параметров.
  Описываемая модель позволяет найти оптимальные параметры методом подбора. Что продемонстрировано в ходе работы.

- Создать прототип виртуальной вычислительной системы.
  Прототип системы был создан. Предварительно была продумана архитектура системы с учетом теоретической части работы.

- Провести тестирование полученной модели.
  Произведено тестирование полученной модели. Подробности и результаты также описаны в работе.

Тем самым цель работы: создание виртуальной распределенной вычислительной среды, которая позволит сконфигурировать предоставляе-

мые вычислительные ресурсы в соответствиям с требованиями приложения была достигнута.

В рамках данной работы представлена новая модель информационной системы для реализации распределенных вычислений. На основе теоретической модели создан прототип. Проведен ряд вычислительных экспериментов, который показал применимость прототипа при решении поставленных задач. Данный подход помимо анализа приложения и оптимального подбора ресурсов, может послужить и для других целей. Например изолированность позволяет реализовать ситуацию, когда на основе аппаратных средств имеются несколько персональных кластеров созданных, к примеру, с целью обучения. При этом программные ошибки и сбои одного кластера не будут влиять на другой кластер, работающий на тех же ресурсов.

В качестве дальнейшей работы можно выделить следущее:

- решить вопрос с ограничением мощности процессора при запуске задач;
- проанализировать данные, которые необходимы для работы приложения с целью проектирования структуры СУБД;
- реализовать полноценную серверную часть;
- реализовать клиентскую часть;
- провести компанию по сбору ресурсов;
- вывести продукт в стадию, когда им можно будет пользоваться.

В ходе развития данной промежуточные результаты были освещены на международной конференции ICCSA 2015 и опубликована работа [8]. В текущем году было выступление на XLVII Международной научной конференции аспирантов и студентов «Процессы управления и устойчивость»

Control Processes and Stability (CPS'16) с последующей публикацей. А также принята статья на ICCSA 2016.

В целом есть надежда, что развитие данной работы может послужить началом эры краудфандинга в информационной среде.

Список литературы

1. Walters J., Chaudhary V. A fault-tolerant strategy for virtualized HPC clusters // The Journal of Supercomputing. 2009. Vol. 50, Issue 3, P. 209-239.
2. Lantz B., Heller B.,McKeown N. A network in a laptop: rapid prototyping for software-defined networks // Proc. of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks. 2010. Article No. 19 .
3. CS144: Introduction to Computer Networking [Электронный ресурс]: URL:http://www.scs.stanford.edu/11au-cs144/(дата обращения: 17.03.16).
4. Сети для самых маленьких. Часть нулевая. Планирование [Электронный ресурс]: URL:https://habrahabr.ru/post/134892/(дата обращения: 17.03.16).
5. Курс молодого бойца. Практический курс по Cisco Packet Tracer [Электронный ресурс]: URL:http://www.scs.stanford.edu/11au-cs144/(дата обращения: 17.03.16).
6. Graphical Network Simulator [Электронный ресурс]: URL:https://www.gns3.com/(дата обращения: 17.03.16).
7. Calarco G., Casoni M. On the effectiveness of Linux containers for network virtualization. 2013. Vol. 31. P. 169–185.
8. Korkhov V., Kobyshev S., Krosheninnikov A. Flexible Configuration of Application-Centric Virtualized Computing Infrastructure // Computational Science and Its Applications – ICCSA. 2015. Vol. 9158. P. 342-353.
9. Wei H., Jiuxing L., Bulent A., Dhabaleswar K. A case for high performance computing with virtual machines // Proc. of the 20th annual Int. conf. on Supercomputing (ICS). 2016. P. 125–134.
10. Linux Containers [Электронный ресурс]: URL:https://

linuxcontainers.org/(дата обращения: 17.03.16).

11. Docker Offical site [Электронный ресурс]: URL:https://docs.docker.com/ (дата обращения: 17.03.16).
12. LXC (Linux Containers) [Электронный ресурс]: URL:http://habrahabr.ru/post/208746/(дата обращения: 17.03.16).
13. Docker toolox [Электронный ресурс]: URL:https://www.docker.com/products/docker-toolbox(дата обращения: 17.03.16).
14. Google Cloud Platform. Container Engine [Электронный ресурс]: URL:https://cloud.google.com/container-engine/docs/clusters/ (дата обращения: 17.03.16).
15. Apache Hadoop [Электронный ресурс]: URL:http://hadoop.apache.org/(дата обращения: 17.03.16).
16. The Apache software Foundation [Электронный ресурс]: URL:http://www.apache.org/(дата обращения: 17.03.16).
17. Hadoop: что, где и зачем [Электронный ресурс]: URL:https://habrahabr.ru/post/240405/(дата обращения: 17.03.16).
18. Scaling up with hadoop and banyan at ITRIX-2015 [Электронный ресурс]: URL:http://www.slideshare.net/rohitkulky/scaling-up-with-hadoop-and-banyan-itrix-feb-2015-public-copy(дата обращения: 17.03.16).
19. Urbah E., Kacsuk P., Farkas Z.,Fedak G., Kecskemeti G., Lodygensky O., Marosi A.,Balaton Z. ,Caillat G., Gombas G., Kornafeld A.,Kovacs J., He H., Lovas R. EDGeS: Bridging EGEE to BOINC and XtremWeb, J. Grid Comput. 7 (3) (2009) 335–354.
20. Kacsuk P., Kovacs J., Farkas Z., Marosi A.Cs., Balaton Z. Towards a powerful european dci based on desktop grids, J. Grid Comput. 9 (2) (2011) 219–239.
21. Silberstein M. ,Sharov A. , Geiger D. ,Schuster A. Gridbot: execution of bags of tasks in multiple grids, in: Proc. of the Conf. on High Performance

Computing Networking, Storage and Analysis, SC '09, ACM Press, 2009, pp. 11:1–11:12.

22. Figueira J., Greco S., Ehrgott M. Multiple Criteria Decision Analysis: State of the Art Surveys, Springer, 2005.
23. Docker Hub [Электронный ресурс]: URL:https://hub.docker.com (дата обращения: 17.03.16).
24. Docker Image Builder [Электронный ресурс]: URL:https://docs.docker.com/engine/reference/builder/ (дата обращения: 17.03.16).
25. QoS on OpenFlow 1.0 with OVS 1.4.3 and POX inside Mininet [Электронный ресурс]: http://users.ecs.soton.ac.uk/drn/ofertie/openflow_qos_mininet.pdf
26. Kocoloski B., Lange J. Better than native: using virtualization to improve compute node performance // Proc. of the 2nd International Workshop on Runtime and Operating Systems for Supercomputers. 2012. Article No. 8
27. Lange J., Pedretti K., Dinda P., Bridges P., Bae C., Soltero P. and Merritt A. Minimal-overhead virtualization of a large scale supercomputer // Proc. of the 7th ACM SIGPLAN/SIGOPS international conference on Virtual execution environments. 2011. P. 169-180.
28. Bailey D., Barszcz E., Barton J., Browning D., Carter R., Dagum L., Fatoohi R., Fineberg S., Frederickson P., Lasinski T., Schreiber R., Simon H., Venkatakrishnan V., Weeratunga S. The NAS parallel benchmarks, RNR Technical Report, RNR-94-007, 1994.