Отдельный раздел в конце – цели и задачи работы

Цель сделать из названия.

Задачи: пункт 2 нужна модель для симуляции. Нужна сисимета, где можно все смоделировать, нужно написать симулятор. Это задача номер 2.

Патентный обзор отдельно задачей.

Проанализирвать существующий рещения по развертыванию ПО на северных фермах. Потом патенты. Потом сущ. Алгорритмы и **то, что собыраюсь использваоть** – это пока не писать, реализация алгоритма, иссселдование работы о=алгоритма. Технико-экономическое обоснование. Нужно схема L2 и L3, докер, соврем вычисления, есть цоды, есть микросервисы, содержит сотни севрисов одно приложение. Как из разворацичвают, как этим управлять, пирамида автоматизации, т.е. это все применияет на L3. L2 не использует оркерстаторы, т.к. ограниченное кол-во серверов, но на l2 тоже используют микросевисы.на l2 также нужен алгоритм для решения задачи распредедения поленой нагрузки.

**Введение**

В современных экономических условиях предприятия заинтересованы в сокращении издержек. С целью увеличения прибыли компании доходы инвестируются в модернизацию технологий и внедрение информационных систем. Для совершенствования управления отраслями и отдельными предприятиями на основе применения математических методов, современных средств вычислительной техники и средств связи для наилучшего использования производственных фондов, увеличения выпуска продукции, снижения ее себестоимости, повышения производительности труда, рентабельности производства и роста прибылей, то есть с целью оптимизации производства, на металлургических предприятиях внедряются системы управления производственными процессами (MES), системы планирования ресурсов предприятия (ERP), управления складами (WMS).

С целью более эффективно обеспечивать безопасность данных и сохранность коммерческой тайны компании под данные информационные системы разрабатывают собственные центры обработки данных, предназначенные только для внутреннего корпоративного пользования.

С целью сокращения издержек на всех уровнях автоматизации производства необходимо следить за рациональным использованием вычислительных ресурсов центра обработки данных. Если на уровне автоматизированных систем управления предприятием для решения данного вопроса используется такие подходы, как контейнеризация и оркестрация, т.е. автоматическое распределение, масштабирование программных компонентов и балансировка нагрузки на них, то на уровне автоматизированных систем управления техническим процессом применяются устоявшиеся технические решения, и программное обеспечение размещается на серверах без применения контейнеризации. Однако, такие ресурсы также нуждаются в управлении и вручную такие задачи, особенно в условиях непрерывно функционирующего предприятия с большим уровнем автоматизации становится решать крайне трудно и даже опасно.

В этой связи заслуживают изучения способы оптимизации развертывания микросервисных приложений на северах ЦОД для минимизации затрат на обслуживание серверов, что и определяет актуальность данной работы.

Объектом исследования является инфраструктура развертывания программного обеспечения на серверах центра обработки данных АО «Северсталь-Инфоком».

Предметом исследования являются методы оптимизации автоматического развертывания микросервисных приложений на серверах.

Цель работы заключается в повышении экономической эффективности обслуживания центра обработки данных.

Для достижения цели выпускной квалификационной работы необходимо выполнить следующие задачи:

1. Проанализировать существующие решения, методы и алгоритмы оптимизации развертывания ПО на серверах, провести патентный обзор.
2. Выбрать подходящий алгоритм, разработать программное обеспечение для системы оптимизации автоматического развертывания микросервисных приложений.
3. Провести экспериментальные исследования системы.
4. Выполнить технико-экономическое обоснование проекта.

Научная новизна работы заключается в применении генетического алгоритма при решении задачи оптимального распределения программного обеспечения на серверах и разработке функций этого алгоритма:

· первоначального заполнения;

· скрещивания;

· мутации;

· функции приспособленности.

ГЛАВА 1. ОБЗОР СОСТОЯНИЯ РАССМАТРИВАЕМОГО ВОПРОСА.

1.1 Описание объекта исследования

Главной особенностью современного этапа развития техники, в частности средств производства, является широкое использование вычислительной техники для автоматизации процессов умственного и физического труда. Коренным образом изменяется характер средств производства, по существу, создается новая материально-техническая база общества. Современная автоматизация производства объединяет множество самых разных задач - от технических до управленческих. Попытка систематизировать эти задачи привела к появлению так называемой "пирамиды автоматизации" (рисунок 1). Это модель, объединяющая все сферы деятельности современного предприятия в единую информационную среду. В основе пирамиды технологические объекты – станки, конвейеры и т.п.



Рисунок 1. Пирамида комплексной автоматизации предприятия.

В структуре пирамиды компьютерной автоматизации различают 5 уровней, связанных между собой как по горизонтали, так и по вертикали информационными каналами.

* уровень 1 представляет из себя набор датчиков, исполнительных устройств, которые предназначены для сбора первичной информации и реализации управляющих воздействий;
* уровень 2 содержит программируемые контроллеры, осуществляющие локальное управление технологических объектом;
* уровень 3 включает системы диспетчеризации, сбора данных и оперативного управления технологическим процессом;
* уровень 4 занимает система управления производством, позволяющая управлять производственными и людскими ресурсами в ходе технологического процесса, управлять качеством продукции, следить за обслуживанием оборудования.
* уровень 5 представляет из себя системы, оснащённые компьютерным оборудованием с программным обеспечением, позволяющим иметь полную информацию о всем производстве и осуществлять планирование ресурсов.
* уровень 6 содержит системы бизнес-аналитики. На этом уровне информация собирается в виде, удобном для восприятия и обработки.

Первые три уровня образуют АСУ ТП - автоматизированную систему управления технологическим процессом, а четвертый, пятый и шестой образуют АСУП - автоматизированную систему управления предприятием.

Работа данных информационных систем невозможна без использования центров обработки данных, которые поддерживают жизненно важные процессы. Центр обработки данных — это сложный комплекс, включающий в себя вычислительные мощности, элементы ИТ — инфраструктуры, строительных и инженерных систем, основными функциями которого являются – хранение, обработка и передача информации. В ЦОД на относительно небольшой площади сосредоточены мощные вычислительные ресурсы: сервера и системы хранения данных (СХД), осуществляющие хранение и обработку информации; сетевое оборудование, отвечающее за обмен данными внутри ЦОД, а также за связь с внешними потребителями; инженерные системы, системы безопасности, системы диспетчеризации и мониторинга, обеспечивающие эффективную работу и защиту сосредоточенного в ЦОД вычислительного центра.

При увеличивающихся темпах цифровизации возникает задача оптимального использования ресурсов серверов, т.е. уменьшения количества задействованных виртуальных машин, а, соответственно, и уменьшения потребления электроэнергии, арендной платы при учитывании ограничений, таких как необходимое количество оперативной памяти, место на жестком диске, тип процессора и других.

Когда количество программных компонентов увеличивается, а процесс развертывания программного обеспечения рассматривается, как повторяющийся цикл разработки, тестирования и размещения программных продуктов на серверах, вручную решать задачу оптимального размещения программных компонентов на серверах становится не только очень сложно, но и крайне неэффективно. Появляется необходимость управлять ресурсами северов.

На сегодня широкое распространение получили технологии контейнеризации. Контейнеризация – это легковесная виртуализация и изоляция ресурсов на уровне операционной системы, которая позволяет запускать приложение и необходимый ему минимум системных библиотек в полностью стандартизованном контейнере, соединяющемся с хостом или чем-либо внешним по отношению к нему при помощи определенных интерфейсов. Контейнер не зависит от ресурсов или архитектуры хоста, на котором он работает.

Все компоненты, необходимые для запуска приложения, упаковываются как один образ и могут быть использованы повторно. Приложение в контейнере работает в изолированной среде и не использует память, процессор или диск операционной системы, в которой запущен контейнер. Это гарантирует изолированность процессов внутри контейнера. В результате контейнеризованное приложение может быть запущено на различных типах инфраструктуры. Благодаря такой высокой эффективности, контейнеризация обычно используется для упаковки множества отдельных микросервисов, из которых состоят современные приложения. На сегодняшний день самая распространенная система, позволяющая упаковать приложение со всем его окружением и зависимостями в контейнер – Docker. В связке с docker используются такие системы кластеризации, как Docker Swarm и Kubernetes.

Приведем пример. На СХКР (склад холоднокатанных рулонов) развернута система слежения за продукцией на складе (WMS система). Программные компоненты данной системы представляют собой микросервисы. Если коротко, то архитектурный стиль микросервисов — это подход, при котором единое приложение строится как набор небольших сервисов, каждый из которых работает в собственном процессе и коммуницирует с остальными используя сетевые протоколы передачи данных. При развертывании микросервисов данной WMS системы существует необходимость в размещении нескольких программных компонентов на различных серверах с учетом дополнительных параметров, таких как операционная система, уровень автоматизации, т.е. уровень АСУТП или уровень АСУП и необходимое количество оперативной памяти, свободное место на жесткой диске.

Количество сервисов в рассматриваемой системе - 20, серверов - 2. Возникает задача размещения данных сервисов на серверах так, чтобы занять наименьшее количество дискового пространства, оперативной памяти и других ресурсов машин. На данный момент данная задача решается следующим образом – на каждый сервер вручную распределяются программные компоненты (службы Windows), обновление также производится вручную.

В данном примере рассматривается система слежения для одного цеха. Еще несколько деталей не уточнил у Саши, уточню и допишу.

Задача оптимального размещения программных компонентов на серверах сводится в общем случае к задаче об упаковке объектов предопределённой формы в конечное число контейнеров предопределённой формы и является [NP-трудной](https://ru.wikipedia.org/wiki/NP-%D1%82%D1%80%D1%83%D0%B4%D0%BD%D0%BE%D1%81%D1%82%D1%8C). Использование точного переборного алгоритма возможно только при небольших размерностях. Обычно для решения задачи используют [эвристические](https://ru.wikipedia.org/wiki/%D0%AD%D0%B2%D1%80%D0%B8%D1%81%D1%82%D0%B8%D0%BA%D0%B0) приближённые полиномиальные алгоритмы и другие методы решения задач комбинаторной оптимизации, как имитация отжига [7] и генетические алгоритмы[8]. Перед тем, как рассмотреть данные алгоритмы подробнее, проведем поиск существующих решений в данной области.

1.2 Анализ существующих решений, методов и алгоритмов оптимизации развертывания ПО на серверах.

Рассмотрим существующие решения в области автоматизации развертывания, масштабирования и управления программными компонентами. Также, будут проанализированы методы балансировки нагрузки, применяемые в облачных решениях, также будут рассмотрены методы распределения ресурсов и балансировки сетевой нагрузки при управлении контейнеризованными приложениями и балансировка нагрузки в операционной системе Windows Server 2019.

Рассмотрим способы управления динамическими рабочими нагрузками в облачных вычислениях. Облачные вычисления — модель обеспечения сетевого доступа по требованию к определенному разделяемому фонду вычислительных ресурсов, например, серверам, устройствам хранения данных, приложениям и сервиса, которые могут быть оперативно предоставлены и освобождены с минимальными эксплуатационными затратами или обращениями к провайдеру этих услуг.

Потребители облачных вычислений могут значительно уменьшить расходы на инфраструктуру информационных технологий (в краткосрочном и среднесрочном планах) и гибко реагировать на изменения вычислительных потребностей, используя свойства вычислительной эластичности облачных услуг. Одной из основных возможностей для динамического перераспределения ресурсов является живая миграция виртуальных машин [4]. Она позволяет облачным провайдерам перемещать виртуальные машины с перегруженных хостов, поддерживая их производительность при заданном SLA (соглашение об уровне услуг: средняя доступность, выраженная как среднее число сбоев на период предоставления сервиса, минимальная доступность для каждого пользователя, среднее время отклика сервиса и т.д.) и динамически консолидировать виртуальные машины на наименьшем числе хостов, чтобы экономить электроэнергию при низкой загрузке. Используя «живую» миграцию и применяя онлайн-алгоритмы, которые позволяют принимать решения о миграции в реальном времени, можно эффективно управлять облачными ресурсами, адаптируя распределение ресурсов к нагрузкам VM, поддерживая уровни производительности VM в соответствии с SLA и снижая энергопотребление инфраструктуры. Важной проблемой в контексте живой миграции является обнаружение состояния перегрузки или недогрузки хоста. Перспективным является подход принятия решений о живой миграции на основе прогнозов использования ресурсов на несколько шагов вперед. Это не только повышает стабильность, так как миграционные действия начинаются только когда нагрузка сохраняется в течение нескольких временных интервалов, но также позволяет облачным провайдерам прогнозировать состояние перегрузки до того, как это произойдёт.  Для обнаружения перегрузки используется долгосрочное прогнозирование временных рядов [5]. В контексте этой работы это означает предсказание значений для 7 временных интервалов в будущем. Хост объявляется перегруженным, если фактическое и прогнозируемое общее использование CPU для 7 временных интервалов в будущем превышают порог перегрузки. Таким же образом определяется и недостаточная загрузка сервера. Глобальный агент принимает решения о распределении ресурсов провайдера с помощью живых миграций виртуальных машин с перегруженных или недогруженных хостов на другие узлы для снижения нарушений SLA и потребления энергии. Он получает уведомления от хост-агента, если узел будет перегружен или недогружен в будущем, и выполнит перенос VM, если оно того стоит.

Одна из наиболее активно развивающихся сегментов IT-рынка, ориентированный на корпоративное использование облачных услуг [1] - контейнеризация. Технология контейнеризации стала популярной относительно недавно. В основном благодаря технологии Docker, позволяющей запускать приложения в контейнере. Это похоже на обычную виртуальную машину, но гораздо лучше, поскольку достигается практически полная независимость от инфраструктуры и пониженное потребление ресурсами. Именно поэтому сейчас можно заметить смещение от традиционных виртуальных машин в сторону контейнеров. Крупные предприятия используют их при [создании облачной ИТ-инфраструктуры](https://www.xelent.ru/services/vdc/).

Когда говорят о контейнерах, нельзя не упомянуть об оркестрации. Оркестрация — это координация взаимодействия нескольких контейнеров. Конечно, можно работать и без оркестрации — никто не запрещает создать контейнер, в котором будут запущены все необходимые процессы. Однако в этом случае такой подход лишен гибкости, масштабируемости, а также возникают вопросы безопасности, поскольку запущенные в одном контейнере процессы не будут изолированы и смогут влиять друг на друга.

Оркестрация позволяет создавать информационные системы из множества контейнеров, каждый из которых отвечает только за одну определенную задачу, а общение осуществляется через сетевые порты и общие каталоги. При необходимости каждый такой контейнер можно заменить другим, что позволяет, например, быстро перейти на другую версию базы данных при необходимости.

Существуют различные платформы для оркестрации контейнеров. Они позволяют реализовать удобные и эффективные средства развертывания контейнерных систем, построения единой централизованной консоли для применения политик управления. Наиболее известны следующие системы: Kubernetes, Docker Swarm и Apache Mesos. Это не единственные системы — есть еще Nomad, Fleet, Aurora, Amazon EC2 Container Service, Microsoft Azure Container Service, однако они менее популярны.

Рассмотрим алгоритмы, применяющиеся при оркетсровке контейнеризованных приложений на примере одного из наиболее популярных продуктов с открытым исходным кодом – Kubernetes. Kubernetes — OpenSource-система для управления контейнерными кластерами. Появилась в результате наработок Google при использовании механизма для изоляции процессов в виртуальной среде (Borg). В 2014 г. Google открыла код Kubernetes и стала распространять систему под лицензией Apache 2.0. Kubernetes распределяет контейнеры по узлам кластера в зависимости от текущей нагрузки и имеющихся потребностей в работе сервисов и обеспечивает распределение сетевой нагрузки.

Распределение ресурсов [3] в процессе работы Kubernetes состоит из того, что ресурсы узлов динамически распределяются между выполняемыми на них контейнерами, и не нужно заботиться о том, как распределить контейнеры в кластере.

Рассмотрим механизм распределения ресурсов на примере. Для обеспечения повторяемости и изоляции на хосте есть Docker-контейнеры. В этом примере (рис. 2) службы запускаются на сервере nginx, php-fpm и mysql. У этих служб в действительности будет ещё больше работающих процессов, каждый из которых требует вычислительных ресурсов. Чтобы с этим можно было удобно работать, логично объединить процессы по группам (например, все процессы nginx в одну группу «nginx»). Простой и очевидный способ сделать это — поместить каждую группу в контейнер (рисунок 2):

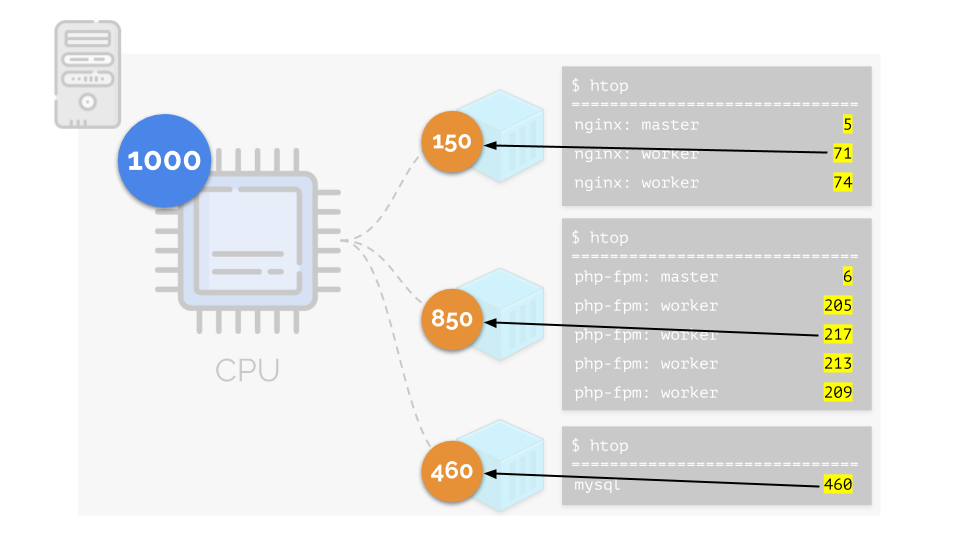


Рисунок 2. Группы процессов

Цифры на рисунке 2, выделенные желтым – потребность каждого процесса в ресурсах CPU, выраженная в условных единицах. При этом у самого CPU есть некий конечный ресурс (в примере это 1000), которого всем может не хватать (сумма потребностей всех групп — 150+850+460=1460). В этом случае ядро начинает раздавать ресурсы и делает это, выдавая одинаковое количество ресурсов каждой группе. Но в первом случае их больше нужного (333>150), поэтому излишек (333-150=183) остаётся в резерве, который тоже равно распределяется между двумя другими контейнерами. Так работает планировщик задач CFS в Linux. Kubernetes следит за тем, чтобы не размещать на сервере больше групп, чем есть ресурсов CPU для суммы весов всех групп. Иначе, Kubernetes автоматически подключит дополнительные виртуальные машины в кластер.

Балансировка сетевой нагрузки обеспечивается следующим образом. Каждое приложение разворачивается как в своем экземпляре контейнера и ему назначается IP-адрес, контейнеры скрыты за сервисом балансировки, также имеющим сетевой адрес [2]. IP-адрес сервиса используется только как точка входа и не обслуживается каким-либо процессом, слушающим этот ip-адрес и порт.   
В Kubernetes реализована псевдобалансировка.  
Если у вас есть три пода (под - это абстрактный объект Kubernetes, представляющий собой группу из одного или нескольких контейнеров приложения (например, Docker или rkt) и совместно используемых ресурсов для этих контейнеров), балансировщик напишет следующие правила:

1. Выбрать первый под с вероятностью 33%, иначе перейти к следующему правилу.
2. Выбрать второй под с вероятностью 50%, иначе перейти к следующему правилу.
3. Выбрать третий под.

Такая система приводит к тому, что каждый под выбирается с вероятностью 33% (Рис. 3).

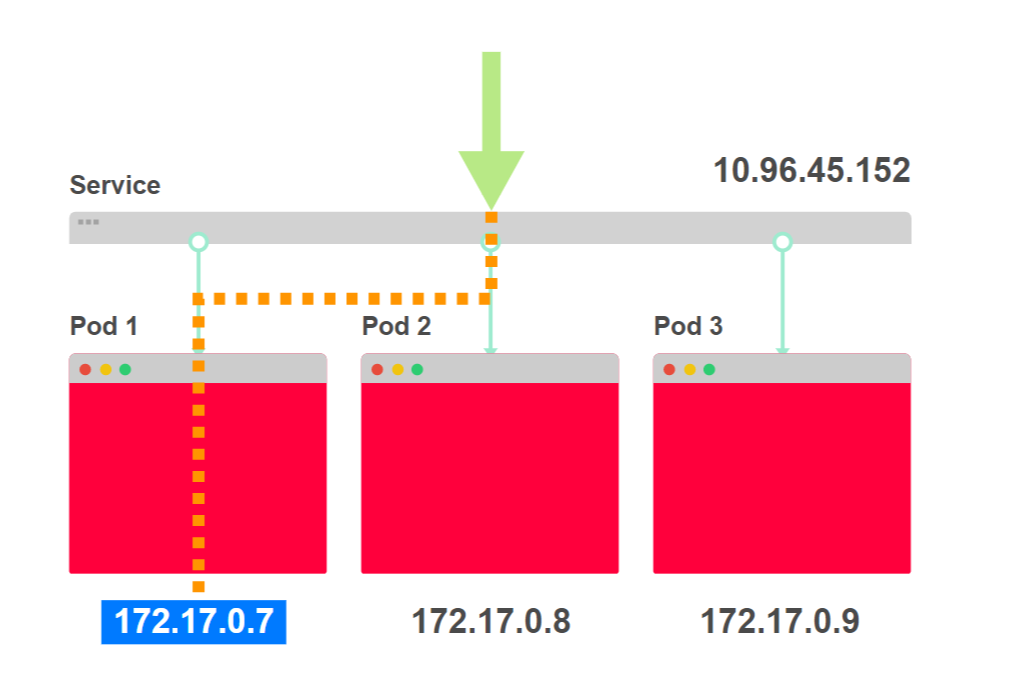


Рисунок 3. Балансировка нагрузки в Kubernetes

Использование метрик, как, например, потребность CPU в ресурсах, описанных в механизме распределения ресурсов в Kubernetes выше, может быть применено при разработке алгоритма оптимизации развертывания микросервисных приложений.

Ключевым фактором для развертываний частных облаков является затраты предприятия на приобретение новых серверов, а также на их модернизацию. Очень часто приходится добавлять избыточность в развертывания частного облака во избежание снижения емкости во время пикового трафика в рабочей среде, но это увеличит денежные затраты. Необходимость избыточности определяется несбалансированными частными облаками, в которых на некоторых узлах размещается больше виртуальных машин, а другие используются недостаточно (например, недавно перезагруженный сервер).

Балансировка нагрузки виртуальных машин — это встроенная функция в Windows Server 2019 и Windows Server 2016, которая позволяет оптимизировать использование узлов в отказоустойчивом кластере. Он определяет перегруженные узлы и перераспределяет виртуальные машины от этих узлов на наименее загруженные при помощи живой миграции. Живая миграция — перенос виртуальной машины с одного физического сервера на другой без прекращения работы виртуальной машины и остановки сервисов. Ниже приведены некоторые ключевые аспекты этой функции:

* Это решение с нулевым временем простоя: Виртуальные машины переносятся в активное состояние на бездействующие узлы.
* Применяется эвристика для балансировки: нехватка памяти и использование центрального процессора узлом.

Балансировка нагрузки виртуальных машин оценивает загрузку узла на основе следующих эвристик:

1. Текущая загрузка памяти: память является наиболее распространенным ограничением ресурсов на узле.
2. Использование процессора усредненного узла в течение 5-минутного периода.

Существует автоматическая и ручная балансировка нагрузки. Данный метод балансировки использует интересующие нас характеристики, как использование ЦП и ОЗУ. Однако, балансировка здесь не применяется для оптимизации распределения программных компонентов по виртуальным машинам. Данные характеристики будут использованы при определении метрик, необходимых для оценки использования серверов микросервисами.

Рассмотрим, также, метод оптимизации распределения задач в операционной системе. Распределение компонентов по задачам и планирование задач являются сложными проблемами, и используются разные подходы. Имитация отжига и генетические алгоритмы являются примерами алгоритмов, которые часто используются для оптимизации.

В этой статье описывается общая схема распределения компонентов по задачам, при сохранении дополнительных функциональных требований. Проверяются временные ограничения и распределения для низкого потребления оперативной памяти и загрузки ЦП. Покажем взаимодействие компонентов и задач на операционной системе реального времени Autocomp (рисунок 4).

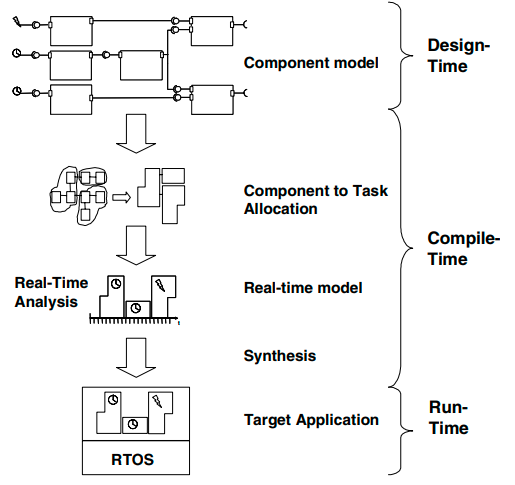


Рисунок 4 - Схема системы Autocomp

На рисунке 4 описано обозначение сборки компонентов с шестью компонентами и четырьмя транзакциями. Опишем компонент Ci набором <Si, Qi, Xi, Mi>, где Si - сигнал от другого компонента, внешнее событие или синхронизированное событие. Qi представляет собой минимальное время прибытия сигнала в случае внешнего события. Он представляет период в случае синхронизированного триггера и не используется если сигнал идет от другого компонента. Параметр Xi - это время выполнения наихудшего случая для компонента, а Mi - это размер стека, необходимое для компонента. Изолированное множество I определяет отношения между компонентами, которые не следует размещать в одной задаче, например, когда компонент имеет неоднозначное время выполнения наихудшего случая.

Компонентная транзакция Сtri - это упорядоченное отношение между компонентами

Ni = c1, c2, ..., cn и сквозным дедлайном dci. Дедлайн имеет отношение к событию, которое инициировало транзакцию компонента, а первый компонент в транзакции определяет триггер транзакции. Компонент транзакция может распространяться на один или несколько компонентов, а компонент может участвовать в нескольких составных транзакциях. Компонент Сa должен выполняться до того, как компонент Сb, а компонент Сb должен выполняться перед Сc, чтобы дать ожидаемые результаты. Правильное поведение исполнения для набора N = c1, c2, ..., cn формализуется регулярным выражением, обозначенным в (1)

c1Σ∗c2Σ∗...cn (1)

Где Σ ∗ обозначает все разрешенные элементы, определенные N

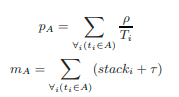
Для набора задач A, который был сопоставлен из компонентов взаимно однозначным образом, легко вычислить потребление системной памяти и накладные расходы ЦП.

Когда несколько компонентов распределяются под одну задачу, нам нужно рассчитать целесообразность распределения и свойств задач. Для набора компонентов c1, ..., cn,

выделенный на набор задач A, рассматриваются следующие свойства.

- Потребление CPU Pa

- Потребление памяти Ma



Где Pa представляет собой сумму накладных расходов на переключение задач, деленную на период для всех задач в системе, а Ma представляет собой общий объем используемой памяти для стеков и блоков управления задачами для всех задач в системе. Распределение может быть выполнено несколькими способами. В маленькой системе все возможные распределения можно оценить и выбрать лучшее. Для более крупной системы однако это невозможно из-за огромного количества комбинаций. Предложенная в статье структура распределения может использоваться для расчета планируемости, нагрузки на ЦП и общей нагрузки на память. Имитация отжига [7], генетические алгоритмы[8] и упаковка контейнеров[6] - хорошо известные алгоритмы, часто используемые для задач оптимизации.

В данном исследовании решено выполнить набор распределений и сравнить результаты с соответствующим однозначным распределением, где каждый компонент выделяется на задачу.

Реализация основана на генетических алгоритмах (рис. 5), и показывает, что каждый ген представляет компонент и содержит ссылку на задачу, которой он назначается. Каждая хромосома представляет собой всю систему со всеми компонентами, назначенными задачам. Каждое распределение, произведенное GA, оценивается структурой, и ему присваивается значение пригодности, зависящее от действительности распределения, потребление памяти и накладные расходы ЦП.

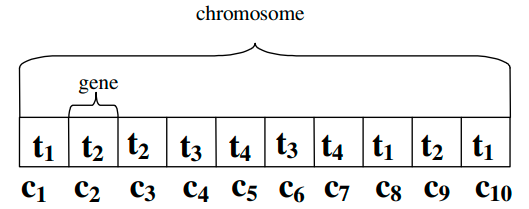


Рисунок 5 – хромосома для генетического алгоритма. Система

с десятью компонентами, распределенными по четырем задачам

Фитнес-функция основана на осуществимости распределения с учетом потребления памяти и CPU. Функция осуществимости состоит из: I, который представляет изоляцию компонентов, IT, представляющего пересекающиеся транзакции, Tr, представляющего типы триггеров и периоды времени, Sc, представляющего планирование задач. Каждому из этих значений присвоено логическое 1 или 0. N – количество компонентов.

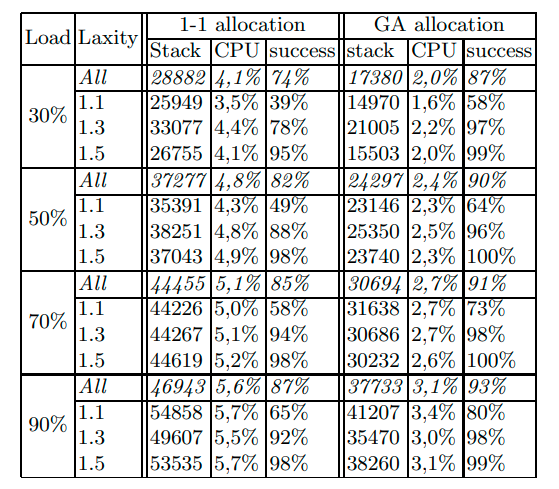


Рисунок 5 – результаты исследования

На рисунке 5 представлены результаты исследования. Моделирование выполняется для четырех различных уровней загрузки процессора: 30%, 50%, 70% и 90%. Сравниваются результаты распределения компонентов и задач один к одному и много компонентов на одну задачу.

Столбцы «Стэк» и «CPU» указывают на процент исползования памяти и процессора для указанных распределений. Среднее улучшение распределения GA по сравнению с распределением 1-1 для коэффициента успеха составляет 10%. Использование ОЗУ уменьшается на 32%, а накладные расходы на переключение задач сокращаются на 48%.

Данное исследование показывает, что применение генетического алгоритма при решении задачи распределения заданий в операционной системе дает положительные результаты. Способы задания хромосом и функции приспособленности могут быть использованы при разработке системы оптимизации развертывания микросервисных приложений.

Вырезать, источники распинать по возможности

Рассмотрим алгоритмы комбинаторной оптмизации, о которых упоминалось ранее, более подробно. Простейшими полиномиальными [алгоритмами](https://ru.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC) упаковки являются:

* 1. Алгоритм «Следующий подходящий» (NF). Упаковка предметов происходит в произвольном порядке по следующему правилу: первый предмет помещается в первый контейнер. На k-м шаге помещается k-й предмет в текущий контейнер. Если предмет помещается, то переходим к следующему шагу, иначе помещаем предмет в новый контейнер.
  2. Алгоритм «Первый подходящий» (FF). Первый предмет помещается в первый контейнер. На k-м шаге ищется контейнер с наименьшим номером, куда помещается k-й предмет, и помещаем его туда. Если такого контейнера нет, то берем новый пустой контейнер и помещаем предмет в него.
  3. Алгоритм «Первый подходящий с упорядочиванием» (FFD).

Предметы сортируются по невозрастанию весов w1 ≥ w2 ≥ … ≥ wn и применяется алгоритм FF.

* 1. Алгоритм «Наилучший подходящий по убыванию» (BFD). Предметы сортируются по невозрастанию весов w1 ≥ w2 ≥ … ≥ wn и применяется алгоритм NF.

Идеи локального поиска получили свое дальнейшее развитие в так называемых метаэвристиках, то есть в общих схемах построения алгоритмов, которые могут быть применены практически к любой задаче дискретной оптимизации. Идея этих методов основана на предположении, что целевая функция имеет много локальных экстремумов, а просмотр всех допустимых решений невозможен, несмотря на конечность их числа. В такой ситуации нужно сосредоточить поиск в наиболее перспективных частях допустимой области. Таким образом, задача сводится к выявлению таких областей и быстрому их просмотру. Каждая из метаэвристик решает эту проблему по‐своему [15]. Рассмотрим генетический алгоритм. Это эвристический алгоритм поиска, используемый для решения задач оптимизации и моделирования путём случайного подбора, комбинирования и вариации искомых параметров с использованием механизмов, аналогичных естественному отбору в природе. Цель генетического алгоритма при решении задачи оптимизации состоит в том, чтобы найти приближённое решение, близкое, но не гарантированно оптимальное решение. Впервые эти нестандартные идеи были применены к решению оптимизационных задач в середине 70-х годов [9]. Примерно через десять лет появились первые теоретические обоснования этого подхода [10, 11]. На сегодняшний день генетические алгоритмы доказали свою конкурентоспособность при решении многих NP-трудных задач [12, 13, 14]. Применение генетических методов для решения NP-трудных комбинаторных задач оптимизации полезно тогда, когда необходимый объем вычислительных затрат может оказаться большим, но скорость, с которой этот объем увеличивается при экспоненциальном росте «размерности» задачи дискретной оптимизации, часто может расти лишь линейно [15].

Рассмотрим еще один эвристический алгоритм - алгоритм имитации отжига. Экзотическое название данного алгоритма связано с методами имитационного моделирования в статистической физике, основанными на технике Монте‐Карло. Исследование кристаллической решетки и поведения атомов при медленном остывании тела привело к появлению на свет стохастических алгоритмов, которые оказались чрезвычайно эффективными в комбинаторной оптимизации. Впервые это было замечено в 1983 году [16]. В настоящее время этот подход является популярным

как среди практиков, благодаря своей простоте, гибкости и эффективности, так и среди теоретиков, поскольку для него удается доказать асимптотическую сходимость к глобальному оптимуму.

Алгоритм основывается на имитации физического процесса, который происходит при кристаллизации вещества, в том числе при отжиге металлов. Предполагается, что атомы уже выстроились в кристаллическую решётку, но ещё допустимы переходы отдельных атомов из одной ячейки в другую. Предполагается, что процесс протекает при постепенно понижающейся температуре. Переход атома из одной ячейки в другую происходит с некоторой вероятностью, причём вероятность уменьшается с понижением температуры. Алгоритм останавливается по достижении точки, которая оказывается при температуре ноль. Алгоритм имитации отжига относится к классу пороговых алгоритмов локального поиска. На каждом шаге в окрестности текущего решения выбирается новое решение. Если разность по целевой функции между ними не превосходит заданного значения, то новое решение заменяет текущее. В противном случае, выбирается другое соседнее решение.

1.3 Патентный обзор

Проанализируем патенты, полученные в ходе патентного поиска.

1. Управление ресурсами сервера, анализ и предотвращение вторжения к ресурсам сервера [RU 2 316 045 C2].

Сущность системы заключается в выполнении мониторинга выбранных ресурсов на одном или более компьютерных серверов и снижении коэффициента загрузки ресурсов в случае перегрузки указанных ресурсов сервера . Система предотвращения вторжения к ресурсам сервера (программный комплекс) выполняет мониторинг выбранных ресурсов на одном или более компьютерных серверах. Программный комплекс может включать один или более программный хост-компонент и программный консольный компонент. Хост-компонент располагается на компьютере сервера и отслеживает использование ресурсов сервера. В случае, если уровень загрузки конкретного ресурса превышает текущий пороговый уровень, хост-компонент может принять корректирующие меры. Указанные меры могут включать в себя, например, снижение уровня использования ресурсов или уведомление пользователя через консольный компонент. Недостатком такого подхода к управлению ресурсами состоит в жестком ограничении доступа к серверу в случае превышения порога загрузки ресурсов сервера.

1. Программно-определяемая автоматизированная система и архитектура[RU 2 729 885 C2].

Данная система предусматривает конфигурирование сети и развертывание функций/приложений автоматизации на лету на системном уровне с помощью технологий виртуализации. В исследуемом патенте описан алгоритм распределения программных компонентов. Алгоритм представляет из себя простую условную конструкцию. Компонент выбора вычислительных узлов, может использовать одно или более правил, регулирующих требования к ресурсам данного хоста, ассоциированного с гостем, чтобы выбирать вычислительный узел для развертывания. Примеры правил, которые может применять компонент выбора вычислительных узлов:

1. Если технология виртуализации хостов представляет собой виртуальную машину, то выбор вычислительного узла с высокопроизводительным процессором (например, многоядерным Xeon-процессором).
2. Если технология виртуализации хостов представляет собой контейнер, то выбор вычислительного узла со среднепроизводительным процессором (например, многоядерным Atom-процессором).
3. Если гость имеет небольшой размер (например, менее 32 MB, между 16 MB и 64 MB), то выбор вычислительного узла без программного обеспечения.

Недостатком данного алгоритма является отсутствие отслеживания загрузки каждого вычислительного узла и наиболее полного использования ресурсов каждого вычислительного узла.

1. Способ и система интеллектуального управления распределением ресурсов в вычислительных средах [RU 2 609 076 C2].

Способ управления распределением информационных ресурсов заключается в том, что посредством компьютера формируют модель использования и перераспределения ресурсов в облачных вычислительных средах. Отличительной особенностью способа является то, что модель использования и перераспределения ресурсов в облачных вычислительных средах в вычислительном облаке формируют с использованием концепции интеллектуальных алгоритмов, последовательно выполняя совокупность операций, включающую три основных этапа: на первом этапе в ВО выделяют ресурсы запускаемому экземпляру, на втором этапе проводят прогноз динамических параметров функционирования хостов ( серверов ) ВО, на третьем этапе осуществляют динамическое перераспределение ресурсов между экземплярами ВО. При этом модель формируют для максимализации показателя ЕВО эффективности функционирования ВО в соответствии с выражением (1) и учетом ограничения (2):





где ЕВО - комплексный показатель эффективности ВО, отражающий совокупную синергетическую эффективность выполнения экземпляров в рамках каждого типа ресурсов ВО;

ЕЦП, ЕОЗУ, ЕД, ЕС - совокупность частных показателей, равных числу экземпляров с удовлетворенной потребностью в ресурсах , соответственно центрального процессора (ЦП), оперативной памяти (ОЗУ), в дисковых (Д) ресурсах , в сетевых (С) ресурсах ;

Т - время функционирования ВО (процессорное время);

RИСП - общий объем использующихся экземплярами ресурсов ВО;

RИМ - общий объем имеющихся ресурсов ВО.

Недостатки: используется все дисковое пространство, нет стремления к минимизации использования ресурсов серверов. Такой подход может применяться при управлении всеми ресурсами вычислительного центра, но не при хостированию отдельных микросервисных приложений. Также, распределение ресурсов выполняется только на основе сформированного вычислительного облака, что приводит к увеличению времени развертывания новых экземпляров приложений.

1. Система и способ оптимизации использования ресурсов компьютера [RU 2 475 819 C1].

Согласно изобретению, дополнен следующими операциями:

1. На первом этапе совокупности операций по формированию модели производится оценка поступления ресурсов от источников, каждый источник характеризуется индексом поступления ресурсов , который задается как отношение поступивших ресурсов к запрошенным у источника.

2. На втором этапе дополнительно учитываются необходимые ресурсы для функционирующих экземпляров приложений, а также составляется перечень ресурсов , которые необходимы для экземпляров приложений, в которых имеется потребность.

3. Четвёртым этапом, на котором запрашиваются необходимые ресурсы у источников с более высоким индексом поступления ресурсов с целью получения недостающих ресурсов и последующего динамического распределения между экземплярами приложений.

Перечисленная новая совокупность существенных признаков позволяет за счет дополнительного учета характеристик источников при запросе дополнительных ресурсов не только распределять ресурсы , имеющиеся в вычислительном облаке, но и запрашивать недостающие ресурсы для вычислительного облака и выделять их экземплярам, которым не хватило ресурсов , тем самым повысить устойчивости функционирования информационно-вычислительной системы путем запуска всех необходимых экземпляров приложений в вычислительном облаке.

Недостатки: в данном изобретении не учитывается возмжность нерационального исползования ресурсов, то есть возможность с помощью данного алгоритма получения схемы распределения ресурсов, далекого от оптимального.

1. Система управления и диспетчеризации контейнеров [RU 2 666 475 C1].

Система управления размещением программных контейнеров предполагает, что программные контейнеры могут быть выполнены с возможностью выполнения в качестве задач в соответствии с определением задач, а определение задач может сохраняться в форме файла определения задач. Файл определения задач может описывать один или большее количество программных контейнеров, назначенных для запуска в качестве группы. Образы программного обеспечения программных контейнеров, которые могут представлять собой полную копию конкретного состояния программного контейнера на момент создания образа программного обеспечения, выполненные с возможностью выполнения в экземплярах программных контейнеров, могут предоставляться поставщику служб вычислительных ресурсов или в его местоположениях, указанных в определении задач. Определение задач также может определять потребности в ресурсах, отношения между контейнерами, используемые сетевые порты и совместно используемые ресурсы. При получении запроса на запуск задач из определения задач планировщик может определить, в соответствии со схемой размещения, какие экземпляры программных контейнеров в кластере будут выполнять задачи. В некоторых случаях поставщик служб вычислительных ресурсов может предоставлять многопользовательский планировщик для определения того, где выполнять программные контейнеры, а в некоторых случаях поставщик служб вычислительных ресурсов может позволить клиентам обеспечивать и настраивать собственные планировщики для настройки работы планировщика. В некоторых случаях планировщик может быть выполнен с возможностью использования случайной схемы выбора для случайного (согласно некоторой схеме стохастического распределения ) или циклического выбора экземпляра контейнера для размещения указанного программного контейнера, учитывая потребность в ресурсах , указанную в определении задач. Данная система использует жестко заданный алгоритм распределения ресурсов, что является ее недостатком.

1.4 Цель и задачи работы

Обзор существующих технических средств показал, что существует достаточно решений управления развертыванием программного обеспечения на уровне автоматизации управления производством, однако, готовых систем, позволяющих оптимально хостировать микросервисные приложения нет, что подтверждает необходимость разработки собственного решения.

Цель работы: оптимизация автоматического развертывания микросервисных приложений.

Задачи работы:

1. Проанализировать существующие решения по развертыванию ПО на северных фермах. Провести патентный обзор.
2. Рассмотреть существующие алгоритмы оптимизации, разработать математическую модель системы, выбрать наиболее подходящий алгоритм оптимизации и реализовать симулятор системы.
3. Реализовать программное обеспечение оптимизации развертывания ПО, провести экспериментальные исследования работы системы.
4. Выполнить технико-экономическое обоснование проекта.

1.5 Выводы по главе 1

В данной главе были рассмотрены факторы, обуславливающие необходимость оптимизации развертывания микросервисных приложений. Были проанализированы существующие на данный момент программные продукты, позволяющие балансировать использование ресурсов и сетевую нагрузку на сервера ЦОД, однако такие средства используются на уровне автоматизированных систем управления предприятием. В ходе анализа выявлено, что готовых систем, позволяющих оптимально хостировать микросервисные приложения нет, что подтверждает необходимость разработки собственного решения.

Список литературы:

1. Что такое оркестрация контейнеров <https://www.xelent.ru/blog/chto-takoe-orkestratsiya-konteynerov/>
2. Балансировка нагрузки и масштабирование соединений в Kubernetes. <https://habr.com/ru/company/mailru/blog/493820/>
3. Управление ресурсами в Kubernetes. <https://habr.com/ru/company/flant/blog/459326/>
4. Балансировка нагрузки в облаках. https://habr.com/ru/company/cloud4y/blog/329416/
5. Дориан Минаролли, Артан Мазрекай и Бернд Фрайслебен в их исследовании «Tackling uncertainty in long-term predictions for host overload and underload detection in cloud computing» <https://link.springer.com/article/10.1186/s13677-017-0074-3>
6. Jansen, K., R, O.S.: Approximation algorithms for time constrained scheduling. In: proceeding of Workshop on Parallel Algorithms and Irregularly Structured Problems, IEEE (1995) 143–157
7. Tindell, K., Burns, A., Wellings, A.: Allocating hard real-time tasks (an np-hard problem made easy). Real-Time Systems 4 (1992)
8. Monnier, Y., Beauvis, J.P., Deplanche, J.M.: A genetic algorithm for scheduling tasks in a real-time distributed system. In: Proceeding of 24th Euromicro Conference, IEEE (1998) 708–714
9. Растригин Л. А. Случайный поиск — специфика, этапы истории и предрассудки. Вопросы кибернетики. Вып. 33 (1978), с. 3–16.
10. Holland J. H. Adaptation in natural and artificial systems. Ann Arbor: University of Michigan Press. 1975.
11. Rechenberg I. Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der Biologischen Information, Freiburg: Fromman, 1973.
12. Еремеев А.В. Разработка и анализ генетических и гибридных алгоритмов для решения задач дискретной оптимизации. Дисс. канд.физ.-мат.наук. Омск, 2000.
13. Goldberg D. E. Genetic algorithms in search, optimization, and machine learning. Reading, MA: Addison-Wesley. 1989.
14. ГЕНЕТИЧЕСКИЙ АЛГОРИТМ ДЛЯ КОНКУРЕНТНОЙ ЗАДАЧИ О P-МЕДИАНЕ. Е.В. Алексеева, А.В. Орлов. Институт математики им. С.Л. Соболева СО РАН, Новосибирский Государственный Университет, Новосибирск
15. Батищев Д.И., Неймарк Е.А., Старостин Н.В. Применение генетических алгоритмов к решению задач дискретной оптимизации. Учебно-методический материал по программе повышения квалификации «Информационные технологии и компьютерное моделирование в прикладной математике». Нижний Новгород, 2007, 85 с.
16. KirkpatrickS., Gelatt C.D., Vecchi M.P. Optimization by simulated annealing // Science. 1983. Vol. 220, P.671–680.