Санкт-Петербургский политехнический университет
Петра Великого
Инженерно-строительный институт

Ефимов Вадим Вячеславович

Интеграция управления и обмена данными в глобально распределенной корпоративной сети

Направление подготовки: 09.06.01 Информатика и вычислительная техника
Направленность: 09.06.01_01 Инженерная геометрия и компьютерная графика

Санкт Петербург – 2021 г.

Выпускная квалификационная работа выполнена в Санкт-Петербургском Политехническом Университете Петра Великого

Научный руководитель: Мещеряков Сергей Владимирович, д.т.н., доцент, профессор высшей школы дизайна и архитектуры СПбПУ Петра Великого

Официальные рецензенты:

Балакшин Павел Валерьевич, к.т.н., доцент факультета программной инженерии и компьютерной техники ФГАОУ ВО “Национальный исследовательский институт ИТМО”;

Янчус Виктор Эдмундасович, к.т.н., доцент высшей школы дизайна и архитектуры СПбПУ Петра Великого.

Оглавление

Введение

Актуальность темы работы обусловлена следующими факторами, сложившимися на данный момент времени в индустрии “облачных” вычислений:

- в процессе обслуживания глобально распределенного вычислительного комплекса, как правило, генерируется большой объем данных, требующих хранения и передачи (500 Мбит/с; 1,5 ТБ в день; и др.)

- разрозненность регионов глобально распределенной информационной системы (ГРИС) в мире, например, западная и восточная части США (USW и USE), Амстердам и Цюрих (AMS и ZRH). И, как следствие, необходимость интеграции данных в единую информационную систему;

- разнородность управляющей информации в задачах технического обслуживания ГРИС: информация о сбоях из системы управления инцидентами (Incident Management Portal, IMP), информация об изменениях из системы управления изменениями (Change Management Portal, CMP), информация о проектах из системы управления проектами (Project Management, PM), мониторинг, capacity planning и др. показатели практи эксплуатации информационных систем (Information Technology Infrastructure Library);

- предоставление облачных сервисов с уровнем доступности 0,99999;

- обслуживание всех регионов ГРИС в режиме реального времени 24/7;

- планирование вычислительных ресурсов и своевременное обеспечение профилактики в условиях быстро меняющейся рабочей нагрузки и возрастающей активности пользователей.

Объект исследования – ГРИС на примере международной компании RingCentral как ведущего представителя современной IT индустрии в сфере облачных технологий.

Предмет исследования – методы организации информационно-управляющей системы (ИУС) и средства мониторинга ГРИС

Цель выпускной работы:

Повышение эффективности технического обслуживания и уровня доступности облачных сервисов на основе разработки новых и совершенствования существующих методов интеграции ИУС на платформах ГРИС.

Задачи:

1. Исследование архитектуры и информационных потоков ГРИС ведущих международных компаний и научное обоснование выбора объекта диссертационного исследования.

2. Разработка модели интеграции разнородных ИУС, используемых в процессе эксплуатации ГРИС, без изменения структур баз данных.

3. Разработка автоматизированных процедур взаимодействия разрозненных ИУС на уровне веб приложений в составе обслуживаемого ГРИС.

4. Совершенствование существующих и разработка новых методов и алгоритмов оценки показателей эффективности интеграции разрозненных ИУС на основе объективных данных их мониторинга.

Научная новизна:

1. Принципы интеграции данных о состоянии ГРИС на уровне веб приложения без модификации структуры данных разнородных ИУС.

2. Прикладная автоматизированная информационная система поддержки принятия решений об изменении состояния облачных сервисов.

3. Методика оценки эффективности управления ГРИС с использованием новых количественных индикаторов мониторинга на основе объективных статистических данных.

Практическая значимость и внедрение:

1. Практическая реализация интеграционной модели обеспечила интеграцию данных в реальном времени и непрерывность обработки глобально распределенной информации в условиях разнородности и разрозненности ИУС международной компании RingCentral.

2. Программная реализация и внедрение интеграционных веб приложений в международной компании RingCentral позволили повысить степень автоматизации и снизить совокупные глобальные затраты на техническое обслуживание ГРИС.

3. Практическая значимость реализованных методов объективного мониторинга Zabbix и полученных результатов оценки эффективности обслуживания ГРИС RingCentral заключается в возможности повысить интенсивность часов плановой профилактики практически в 2 раза при сохранении требуемого уровня доступности облачных сервисов 0,99999.

Основные положения, выносимые на защиту:

1. Организационно-технические принципы интеграции баз данных состояния ГРИС

2. Информационная модель предметной области эксплуатации глобально-распределенных информационных систем

3. Методика прагматического анализа информации состояния «облачного» сервиса в интересах организации интерфейса службы эксплуатации глобально распределенной информационной системы

Апробация:

Результаты диссертационной работы докладывались на ежегодных научных семинарах СПбПУ (Санкт-Петербург, 2014-2021), а также на международных конференциях:

- ICUMT 2014 - the 6th International Congress on Ultra Modern Telecommunications and Control Systems (St. Petersburg, Russia, October 6-8, 2014)
- XLIV Неделя науки СПбПУ: науч. конф. с междунар. участием, СПб, СПбПУ, 2015
- Distributed computer and communication networks (DCCN-2015): control, computation, communications (Institute of control Science RAS, Россия, 19-22 октября 2015)

- Distributed computer and communication networks (DCCN-2016): control, computation, communications (Москва, РУДН с 21 по 25 ноября 2016 г.)
- XLV Неделя науки СПбПУ: науч. конф. с междунар. участием, СПб, СПбПУ, 2016
- International Conference on Analytical and Computational Methods in Probability Theory (Moscow, 2017)
- Distributed computer and communication networks (DCCN-2017): control, computation, communications (Москва, РУДН с 25 по 29 сентября 2017 г.)
- Компьютерное моделирование: науч.-техн. конф. (КОМОД-2017), СПб, СПбПУ, 2017 (с 3 по 4 июля 2017 г.)
- Системный анализ в проектировании и управлении (SAEC-2017): междунар. науч.-практич. конф. (Санкт-Петербург, СПбПУ, 29 – 30 июня 2017 г.)
- XLVI Неделя науки СПбПУ: науч. конф. с междунар. участием, СПб, СПбПУ, 2017 г.
- XLVII Неделя науки СПбПУ: науч.-практич. конф. с междунар. участием, СПб, СПбПУ, 2018 г.

Публикации:

Всего опубликовано 19 научных трудов, из них 4 статьи в журналах из перечня ВАК, 3 работы проиндексированы в Scopus и 11 в РИНЦ (из них 2 работы без соавторов), общим объемом 8 п.л. Индекс цитирования Хирша составляет 3 по состоянию на 2021 г.

Глава 1. Аналитический обзор современных облачных технологий

1.1. Анализ ведущих мировых представителей в сфере облачных Интернет услуг

Ранние концепции использования вычислительных ресурсов по принципу системы коммунального хозяйства относят к 1960-м годам (к Джону Маккартиили и Джозефу Ликлайдеру) [1].

Первым реальным облачным сервисом считается система управления взаимоотношениями с клиентами (client relationships management system, CRM-system) Salesforce.com, которая появилась в 1999. Следующей важной вехой является книжный магазин Amazon.com, который начал в 2002 году предоставлять доступ к своим вычислительным ресурсам через интернет [2]. Компания Amazon благодаря этим услугам фактически превратилась в технологическую компанию, что привело к идее вычислительной эластичности и запуску в 2006 году проекта Elastic Cloud Computing (Amazon EC2). Практически одновременно с этим, Эрик Шмидт, глава корпорации Facebook, использовал термины cloud и cloud computing в одном из своих выступлений. Дальше эти термины получили широкое распространение благодаря средствам массовой информации и публикациям специалистов по информационным технологиям. Облако стало использоваться как символ интернета на различных диаграммах, а также как образ сложной инфраструктуры, за который скрываются все технические детали.

Следующим крупным шагом в развитии концепции облачных вычислений считается запуск в 2009 году приложений Google Apps [2]. В последующие годы (2009 – 2011) были сформулированы обобщенные представления об облачных вычислениях. Была выдвинута модель частных облачных вычислений (private cloud), применимая внутри организации, были выделены модели обслуживания такие как инфраструктура как сервис (infrastructure as a service, IaaS), платформа как сервис (platform as a service, PaaS), программный продукт как сервис (software as a service, SaaS). В итоге, в 2011 году было сформулировано единое понятие облачных вычислений, что позволило Национальному институту стандартов и

технологий США покончить с расхождениями в определении облачных вычислений.

Национальным институтом стандартов и технологий США зафиксированы следующие обязательные характеристики облачных вычислений [1]:

Самообслуживание по требованию (self service on demand) — потребитель самостоятельно определяет и изменяет вычислительные потребности, такие как серверное время, скорости доступа и обработки данных, объём хранимых данных без взаимодействия с представителем поставщика услуг;

Универсальный доступ по сети — услуги доступны потребителям по сети передачи данных вне зависимости от используемого терминального устройства;

Объединение ресурсов (resource pooling) — поставщик услуг объединяет ресурсы для обслуживания большого числа потребителей в единый комплекс для динамического перераспределения мощностей между потребителями в условиях постоянного изменения спроса на мощности; при этом потребители контролируют только основные параметры услуги (например, объём данных, скорость доступа), но фактическое распределение ресурсов, предоставляемых потребителю, осуществляет поставщик (в некоторых случаях потребители всё-таки могут управлять некоторыми физическими параметрами перераспределения, например, указывать желаемый центр обработки данных из соображений географической близости);

Эластичность — услуги могут быть предоставлены, расширены, сужены в любой момент времени, без дополнительных издержек на взаимодействие с поставщиком, как правило, в автоматическом режиме;

Учёт потребления — поставщик услуг автоматически исчисляет потребленные ресурсы на определённом уровне абстракции (например, объём хранимых данных, пропускная способность, количество пользователей, количество транзакций), и на основе этих данных оценивает объём предоставленных потребителям услуг.

Поставщик облачных услуг имеет возможность экономить вычислительные мощности благодаря тому, что потребители используют их не регулярно. Одни и те же вычислительные мощности могут обслуживать разных пользователей в

разные промежутки времени. Второй источник экономии – автоматизация выделения вычислительных ресурсов. Все это вместе существенно снижает затраты на обслуживание абонентов.

Потребитель в результате получает услугу с высоким уровнем доступа (high availability), низкими рисками неработоспособности. Пользователь может быстро масштабировать свою вычислительную систему, запрашивая вычислительные ресурсы по потребности.

Удобство и универсальность доступа обеспечивается широкой доступностью услуг и поддержкой различного класса терминальных устройств (персональных компьютеров, мобильных телефонов, интернет-планшетов).

Согласно исследованию Джеймса Антони из аналитического агентства FinancesOnline [2], рынок облачных услуг характеризуется следующей статистикой:

1. В 2018 году поставщики облачных услуг потратили 63,1 миллиарда долларов на научно-исследовательские и опытно-конструкторские работы (НИОКР), что равняется 20% всего НИОКР американских компаний
2. К настоящему дню лучший год по количеству первичных размещений акций компаниями, предоставляющими облачные услуги был 2018 год с 17 размещениями, собравшими суммарно 5,1 миллиард долларов, что вдвое больше предыдущего рекорда
3. Начиная с 2010 года средние затраты компаний на облачные услуги постоянно росли год от года. В 2018 году средние затраты американской компании составили 343 тысячи долларов, что на 78% больше предыдущего года
4. Компания Salesforce является лидером рынка облачных услуг по рыночной капитализации, которая на 2019 год составила 117,8 миллиардов долларов США
5. Среднее количество различных облачных приложений, используемых американскими компаниями выросло на 30% в 2019 году по отношению к 2018 году и составило 137 штук

6. В 2018 году общее количество людей занятых в индустрии облачных сервисов составило 206 миллионов по всему миру и ожидается рост до 380 миллионов к 2021 году
7. На 2020 год в мире насчитывается 15 529 компаний, предоставляющих облачные услуги

Аналитическое агентство Gartner, основанное в 1979 году и являющееся лидером аналитических услуг на рынки информационных технологий, а также входящее в список 500 американский компаний с наибольшей капитализаций (S&P 500) с капитализацией 17 миллиардов долларов, делит индустрию информационных технологий на 153 отдельных рынка, выделяя на каждом рынке лидеров, претендентов на лидерство, визионеров и нишевых игроков.

Оценка игроков рынка производится по двум параметрам - способность исполнять (ability to execute) и полнота стратегии (completeness of vision).

Способность исполнять включает в себя:

1. Набор продуктов и услуг компании. Ключевые продукты и услуги, предлагаемые поставщиков на обозначенный рынок. Это включает в себя набор продуктов и услуг, их качество, функциональность и т.д., предлагаемые либо напрямую, либо через посредников и партнеров.
2. Общую финансовую стабильность компании. Включает в себя оценку финансовой стабильности организации в целом, финансовые показатели подразделения, работающего на обозначенном рынке, вероятность того, что это подразделение продолжит инвестировать в свой продукт и предлагать его на рынке, увеличивая значимость продукта в общем портфолио организации
3. Каналы продаж и ценовую политику. Включает в себя оценку переговорной позиции поставщика услуг на этапе заключения сделки о предоставлении услуги - умение вести сделки, вести переговоры о цене, оказывать поддержку на этапе заключения сделки, общая эффективность каналов продаж

4. Маркетинг компании. Ясность, качество, креативность и эффективность программ компании, направленных на то, чтобы донести послание организации рынку, продвинуть свой бренд, повысить узнаваемость продукта, создать позитивное восприятие продукта, бренда и организации у покупателей. Это может быть достигнуто комбинацией публичности, промо-акций, маркетинговых акций, и пр.
5. Пользовательский опыт. Сам продукт и поддержка пользователя, которые помогают пользователю получить желаемый результат от использования продукта. В первую очередь это включает в себя то, как пользователь может получить техническую поддержку. Также включает в себя вспомогательные инструменты, программы поддержки (их их качество), доступность разграничения прав пользователей, соглашения об уровне оказания услуг и т.д.
6. Операционная деятельность. Способность организации достигать целей и обязательств. Сюда относятся качество организационной структуры, включая навыки, опыт сотрудников, программы, системы и другие способы обеспечения эффективной работы организации на постоянной основе

Полнота стратегии включает в себя:

1. Понимание рынка. Способность поставщика услуг понимать желания и нужды покупателей, конвертировать их в продукты и сервисы. Поставщики, которые наилучшим образом умеют слушать и понимать желания и нужды покупателей, также адаптируют свою стратегию, учитывая их
2. Маркетинговая стратегия. Набор различных, четко сформулированных сигналов, которые компания постоянно коммуницирует как внутри себя, так и вовне через официальный сайт, рекламу, программы привлечения клиентов.
3. Стратегия продаж. Стратегия продажи продуктов, которая использует соответствующую сеть прямых и непрямых продаж, маркетинг, посредников, которые расширяют охват рынка, увеличивают набор навыков, экспертизу, технологии и сервисы компании, а также пользовательскую базу

4. Продуктовая стратегия. Подход поставщика услуг к созданию продукта и поставкам, которые подчеркивают отличительные особенность продукта, его функциональность, методику разработки, учитывая как текущие так и будущие требования рынка
5. Бизнес модель. Оценивается логичность и четкость предпосылок бизнес модели
6. Индустриальная стратегия. Стратегия поставщика направлять необходимые ресурсы, навыки и предложения других компаний на то, чтобы удовлетворить конкретные потребности сегментов рынка
7. Инновация. Планирование ресурсов, экспертиз или капитала для возможности инвестирования, консолидации, защиты от агрессии конкурентов, либо нанесения упреждающего удара
8. Географическая стратегия. Стратегия поставщика направлять ресурсы, навыки и предложения других компаний удовлетворить потребности рынков в регионах вне “домашнего”, как напрямую так и через партнеров и субподрядчиков в зависимости от особенностей региона

Одним из рынков информационных технологий, анализируемых агентством Gartner, является рынок облачных теле-коммуникаций (Unified Communications as a Service, UCaaS). Объем рынка составляет на 2021 год 47,64 миллиарда долларов с прогнозируемым ростом до 210 миллиардов долларов к 2028 году, что означает рост год от года в 23% [4].

Рисунок 1.1: аналитический квадрант агентства Гартнер по рынку облачных телекоммуникаций за 2020 год

Лидерами рынка по мнению аналитического агентства Гартнер (Рис. 1.1) являются 5 компаний: Microsoft, RingCentral, Cisco, Zoom, 8x8. Компанию RingCentral находится в верхней правой части квадрата лидеров, предоставляет услуги обмена мгновенными сообщениями, телефонии, видеоконференций, факсимильных сообщений, СМС сообщений и других. Компания предоставляет услуги по большей части в Европе и Северной Америке. Достижения последних лет включают в себя стратегическое партнерство с AT&T, ALE и Avaya. По

мнению Гартнер, компания RingCentral продолжает совершенствоваться в ключевых областях, таких как разработка продуктовых линеек, глобальное покрытие и поддержка пользователей [5].

Исходя из быстрого роста рынка облачных телекоммуникационных услуг и лидирующего положения на этом рынке компании RingCentral, а также благодаря тому, что эта компания предоставляет свои “облачные” услуги на основе ГРИС, что является типовой архитектурой для многих крупных IT компаний, она была выбрана в качестве объекта исследования в данной работе.

1.2. Исследование типовой архитектуры и информационных потоков в облачных системах

Рассмотрим типовую архитектуру облачной системы на примере архитектуры облачного сервиса компании RingCentral. В основе архитектуры лежит клиент-серверный подход, представляющий из себя связку из базы данных (БД) и серверов приложений, которые отвечают на запросы клиентов (Рис. 1.2)

Рисунок 1.2: архитектура “клиент-сервер”: связка из базы данных и серверов приложений, отвечающих на запросы клиентов

Клиент, например, веб приложение, запущенное в интернет-обозревателе на устройстве пользователя, обращается напрямую к серверам приложений, а те, в свою очередь, отправляют по необходимости запросы в базу данных, в которой хранится информация обо всех пользователях, все тарифы на услуги, а также вся служебная информация.

Такая архитектура имеет существенный недостаток - база данных является узким местом системы. При выходе из строя сервера базы данных, пользователь получает отказ в обслуживании на любой запрос. Чтобы решить эту проблему, а также снять риск выхода из строя всего центра обработки данных (ЦОД), в котором размещаются серверы, архитектура системы расширяется дополнительными компонентами (Рис. 1.3). Появились резервные серверы приложений и база данных, размещенные в другом ЦОД. В случае выхода из строя активной базы данных, либо всего ЦОД, в котором располагается эта база данных, запросы пользователя перенаправляются к резервным серверам. Для того, чтобы перенаправление произошло незаметно для клиента, он общается с балансировщиком запросов. Задачей балансировщика запросов является опрашивание серверов приложений и баз данных с целью выяснения их работоспособности и направление запросов пользователей к активным и работоспособным в данный момент серверам.

Также, для того чтобы переключение происходило быстро и без потери пользовательских данных, между серверами баз данных настроена репликация данных. В каждый момент времени (с поправкой на сетевую задержку) информация в обеих базах данных идентичная.

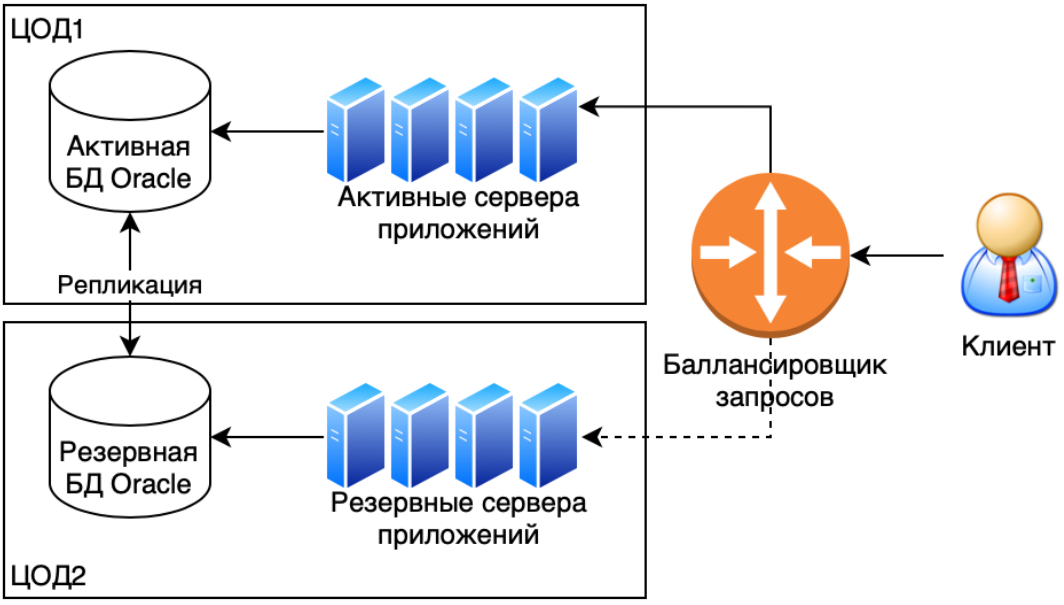

Рисунок 1.3: отказоустойчивая архитектура с резервированием БД и серверов приложений

Следующим этапом усложнения архитектуры “клиент-сервер” стал выход на другие регионы - в Европу, Африку и Азию (рис 1.4) и необходимость дать пользователям возможность совершать звонки по всему миру.

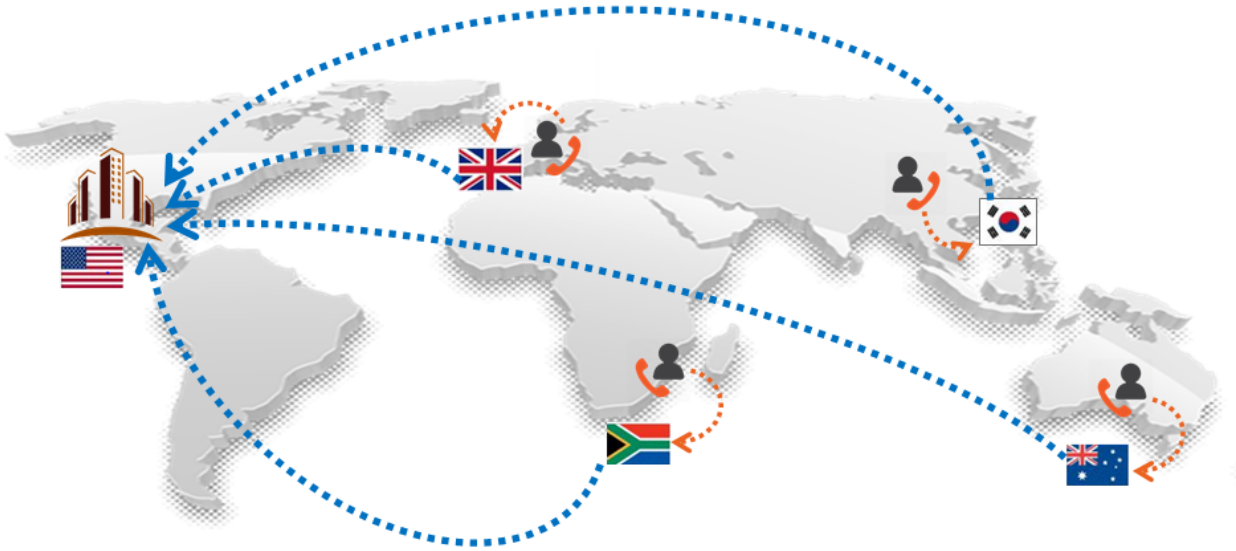

Рисунок 1.4: типовая архитектура ГРИС на примере компании RingCentral

Выход на глобальный рынок потребовал качественного изменения архитектуры, называемого сегментированием. Сегментирование - это выделение групп серверов приложений со своей БД для хранения данных определенных групп пользователей, например по географическому признаку. Ряд стран современного мира накладывает требование на поставщиков информационных услуг, обязывая их хранить данные граждан своей страны на территории этой страны. Таким образом, граждане, например, Германии, при обращении к “облачному” сервису, будут направлены для обслуживания в сегмент сети территориально расположенный в центре обработки данных, находящемся в Германии. Также это архитектурное изменение позволило более эффективно масштабировать сеть. Для региона с большим количеством потребителей сервиса появляется возможность развернуть несколько сегментов, каждый из которых будет обслуживать часть пользователей этого региона.

Однако при наличии нескольких сегментов, каждый со своей базой данных, хранящей информацию только об определенном наборе пользователей, а также

при сохранении внешнего удобства для клиентов - единой точки входа для запросов, потребовалось добавить в систему более умный слой маршрутизации. Запрос пользователя должен быть направлен в тот сегмент, который хранит информацию об этом пользователе, для чего слой маршрутизации должен иметь информацию о том, как идентифицировать пользователя и о том, в какой сегмент направить его запрос. Это потребовало настройки более сложной системы репликации данных - не только между активной и резервной базами данных, но также и между базой данных слоя маршрутизации. Архитектура RingCentral с сегментами (POD, Point of Data) и слоем маршрутизации (Router, Middle Layer) представлена на рисунке 1.5.

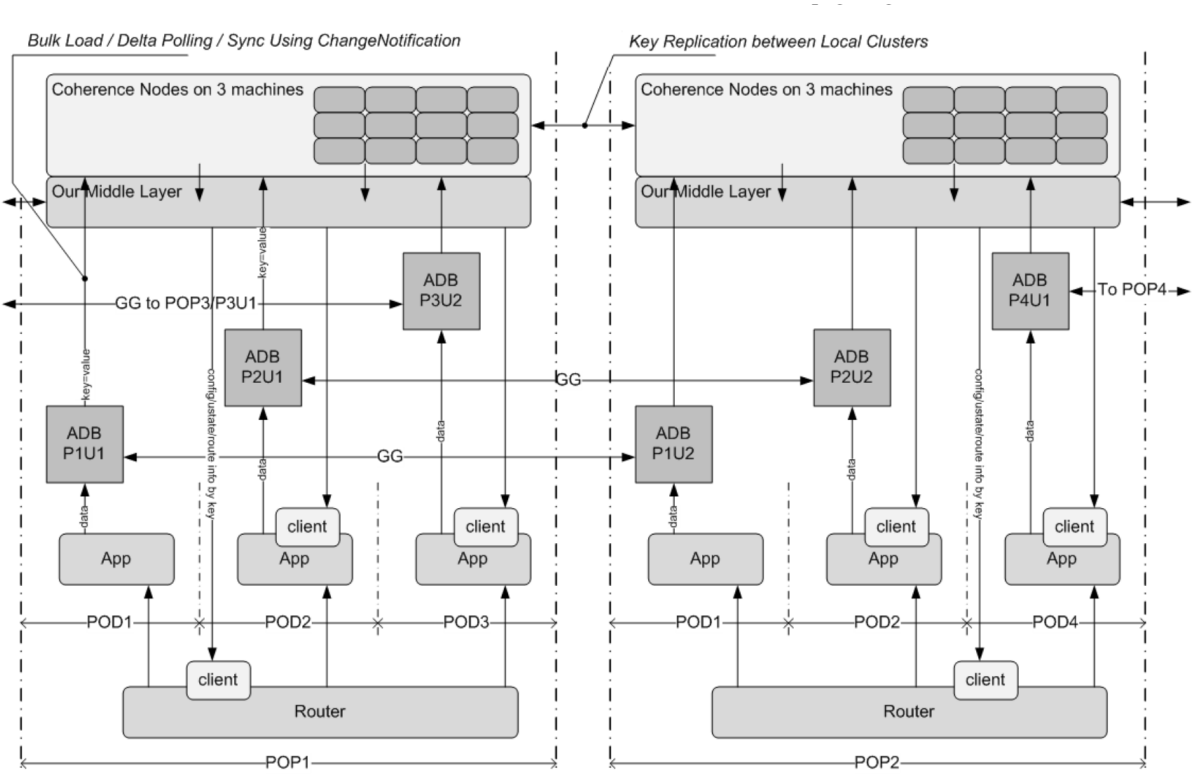

Рисунок 1.5: архитектура RingCentral с сегментами (POD) и слоем маршрутизации (Router, Middle Layer).

Глобальная распределенность вычислительной сети дает еще одно преимущество - устойчивость к авариям большого масштаба, таким как природные катаклизмы, например ураганы. Ураган может обесточить и вывести из строя целый город или регион. С точки зрения сети это означает внезапную потерю базы данных и серверов приложений одного или нескольких сегментов

сети, при этом, в случае если выбор расположения резервных мощностей был сделан таким образом, что они остались доступны, существует возможность быстрого восстановления обслуживания пользователей в полном объеме. Происходит процесс аварийного переключения сегмента на резервные мощности (failover). При этом аварийное переключение, в отличие от штатного переключения (switchover) происходит с разрывом потока данных между рабочей и резервной группой серверов и баз данных, что приводит к частичной потере данных и, соответственно, потере целостности данных. Таким образом, после аварийного переключения, обратное переключение становится невозможным до восстановления целостности данных.

Аварийное переключение в сети компании RingCentral происходит в течении секунд и его скорость обусловлена в первую очередь перестройкой топологии сети Internet по протоколу BGP (Border Gateway Protocol). Помимо времени простоя, еще одним недостатком аварийного переключения является потеря консистентности данных в основной и резервной базах данных из-за нештатного разрыва репликации данных. Это приводит к частичной потере пользовательских данных (например, истории звонков либо настроек маршрутизации звонка до конечного абонента), а также, что немаловажно, к сложности восстановления репликации, т.к. для репликации данных необходима консистентность данных в обеих базах данных.

Для избежания проблем, связанных с аварийным переключением, предусмотрено также плановое переключение сегмента сети на резервные мощности. Это может быть использовано в ситуации, когда метеорологическая служба заранее предупреждает о надвигающемся погодном явлении и есть риск выхода из строя серверов, расположенных в определенном центре обработки данных.

Второй сценарий, подразумевающий необходимость планового переключения связан с внесением изменений в вычислительную сеть. Ряд изменений требует временно вывода из рабочего состояния тех серверов, на которых производятся работы. Например, обновление операционной системы

часто требует перезагрузки сервера. Таким образом, для того чтобы обеспечить непрерывность обслуживания с сохранением качества обслуживания в соответствии с Service Level Agreement (SLA), возникает необходимость производить работы на серверах, которые в данный момент являются резервными. Для этого перед началом работ производится плановое переключение сегмента на резервные мощности, что делает до этого активные серверы резервными.

Стоит также отметить, что предоставление ряда сервисов может не требовать хранения данных пользователей. Примером такого сервиса является простейшая телеконференция с использованием видео связи. Архитектура такого сервиса предполагает динамическое развертывание необходимой инфраструктуры и приложений в начале телеконференции и выключение этой инфраструктуры и приложений по окончании телеконференции для экономии ресурсов. Данные, используемые в процессе предоставления такого сервиса, не требуется сохранять и, поэтому, в данной сети может не использоваться система хранения данных. Однако, на практики такие сервисы бедны функциональностью (например, отсутствует возможность просматривать историю участия в телеконференция, сохранять настройки пользователя и т.п.) и в коммерческих целях не конкурентоспособны, поэтому далее рассматриваться не будут.

В случае выхода из строя инфраструктуры или приложения, обеспечивающих проведение текущей телеконференции, эта конференция будет оборвана и запущен процесс пересоздания, который заново развернет необходимые инфраструктуру и приложения. Такой архитектурный подход позволяет существенно упросить процедуру восстановления сервиса. Также, в случае прогнозируемого природного события, такого как ураган, существует возможность превентивно запретить разворачивание инфраструктуры для новых телеконференций в определенных центрах обработки данных, минимизируя риск отказа в обслуживании пользователей.

1.3. Аналитический обзор программных средств и информационных процессов в “облачных” системах

Для эксплуатации облачных сервисов и их восстановления в случае сбоя в IT компаниях традиционно применяются следующие процессы в соответствии со стандартом IT Infrastructure Library (ITIL) [6]:

1) мониторинг и управление событиями – систематическое наблюдение за сервисами и их компонентами, фиксация и формирование отчета о произошедших изменениях их определенных параметров (мониторинговых событиях)

2) управление сбоями – процесс, нацеленный на максимально быстрое восстановление сервисов после сбоя. Сбоем считается любой незапланированный отказ в обслуживании пользователя либо снижение качества предоставления сервиса ниже установленного соглашением о качестве предоставлении сервиса (SLA), а также выход из строя компонента системы, не приводящего непосредственно к вышеописанным двум состояниям (например, выход одного диска из отказоустойчивого массива дисков - потеря резервирования)

3) контроль за проблемами – процесс выявления и исправления ошибок в системе и предотвращения таким образом повторных сбоев. Для сбоя, который уже произошел, производится анализ развития событий, приведших к данному сбою, формулируются корректирующие действия - будь то исправление программного кода, либо корректировки должностных инструкций сотрудников службы эксплуатации

4) управление изменениями – процесс, направленный на минимизацию рисков, связанных с плановым изменением системы. Изменение может быть связано с добавлением новых вычислительных мощностей, вывод из обслуживания старых в связи с закончившимся гарантийным сроком эксплуатации, а также развертывание обновленных приложений для устранения уязвимостей и ошибок либо добавления новой функциональности.

5) управление версиями обновлений - процесс, целью которого является планирование дат и последовательности внесения различных изменений в

систему, а также информирование заинтересованных сторон об этих планах. Последовательность внесения изменений имеет значение, т.к. ряд изменений зависит от других, например новая версия программного кода на серверах приложений часто будет работать корректно только после того, как будут внесены необходимые изменения в структуру данных на серверах баз данных

В компании RingCentral каждый из этих процессов обеспечивается отдельными подразделениями эксплуатации, которые разрознены географически и используют разнородные инструментальные средства автоматизации (табл. 1.1).

Таблица 1.1: информационные процессы эксплуатации и средства их автоматизации

Процесс	**Подразделение**	**Средство автоматизации**
Мониторинг событий	Команда мониторинга	Zabbix
Управление сбоями	Центр эксплуатации сети	Incident Management Portal
Контроль за проблемами	Команда аналитиков и менеджеров	Production Support Portal, Jira
Управление изменениями	Команда по внесению изменений	Auto-Deployment System Change Management Portal
Управление версиями обновлений	Команда по внесению изменений	Roadmap

Мониторинг событий в компании RingCentral осуществляется с помощью мониторинговой системы Zabbix [7]. Zabbix представляет из себя программный продукт для коммерческого использования, спроектированный для того, чтобы собирать данные от Zabbix-агентов, установленных на множестве серверов. Подробная архитектура системы мониторинга рассмотрена в главе 4.

В случае обнаружения сбоя, он регистрируется в системе управления сбоями IMP. В соответствии с лучшими практиками эксплуатации

информационных систем, описанными в сборнике лучших практик ITIL [6], инциденты классифицируются по степени влияния на потребителей “облачного” сервиса: *level 5* инцидент означает сбой, не оказывающий влияния на потребителей, однако повышающий для системы риск отказа в обслуживании, например, выход из строя резервных мощностей, либо невозможность переключиться на резервные мощности. *Level 4* и выше до *level 1* инциденты отличаются количеством пользователей, испытывающих отказ в обслуживании - от одного для *level 4* до более 25% общего числа для *level 1*.

В рамках процесса по управлению сбоями также фиксируется продолжительность сбоя, список отказавших сервисов и серверов, список сегментов сети и регионов, в которых произошел сбой. Фиксируется каждый этап работ по восстановлению работоспособности системы.

Если для устранения инцидента требуется внести изменение в систему, например изменить конфигурацию сети для защиты от атаки по принципу “отказ в обслуживании” (denial of service), запрещая запросы из определенных подсетей глобальной сети Internet, такие изменения вместе с любыми другими внеплановыми и плановыми изменениями фиксируются в системе управления изменениями. В компании RingCentral в качестве системы управления изменениями используется CMP.

Все изменения классифицируются по следующим типам:

Изменение уведомительного типа (notification type of change) - хорошо известное и опробованное изменение, не несущее значимых рисков для работоспособности распределенной сети. Такое изменение не требует большого количества согласований и может быть выполнено, как правило, в любое удобное для инженера, вносящего это изменение, время.

Стандартное изменение (standard change) - любое изменение, которое несет риски для работоспособности системы, однако эти риски минимизированы за счет того, что изменение выполняется в то время, когда количество запросов минимально (например, ночью), также за счет того, что изменение хорошо протестировано и имеет сценарий отката к исходному состоянию. Такие

изменения требуют согласования с руководством службы эксплуатации, а также уведомления пользователей и партнеров компании.

Крупное изменение (major change) - изменение, которое либо несет существенные риски для работоспособности системы, которые невозможно минимизировать, либо явно предполагает отказ в обслуживании пользователей либо деградацию качества предоставляемых услуг на время выполнения этого изменения. К такого рода изменениям относятся, например, изменения на корневом маршрутизаторе, обеспечивающим сетевую связность внутренней сети RingCentral с сетью Internet. Эти изменения должны делаться, уведомив пользователей и партнеров минимум за 2 недели до плановой даты выполнения. Также требуется согласование высшего руководства службы эксплуатации компании.

В качестве исполняющей системы, осуществляющей непосредственное внесение изменения, используется Automated Deployment System

Для управления версиями обновлений в компании RingCentral используется информационная система Roadmap. С ее помощью планируются этапы развертывания изменений. Наличие резервных серверов, которые должны быть задействованы в случае выхода из строя основных, подразумевает что внесение изменения нецелесообразно делать сразу и на основные мощности и на резервные, так как процесс внесения изменения или само изменение могут привести к сбою. Также, наличие сегментов сети позволяет минимизировать риски, связанные с внесением изменения, проверив его сперва только на одном сегменте (канареечное развертывание). В результате для внесения одного изменения появляется последовательность задач, каждая из которых оформляется как отдельный запрос на изменение в системы управления изменениями. Набор и последовательность этих задач формируются в системе управления версиями обновлений.

Помимо последовательности внесения одного изменения, система управления версиями обновлений также формирует последовательность внесения

разных изменений в систему в соответствии с техническими требованиями разработчиков системы.

Управление проблемами осуществляется с использованием информационно-управляющей системы Production Support Portal (PSP) и системы управления заявками Jira [JIRA]. Процесс управления проблемами запускается сразу после устранения сбоя в системе. Производится анализ последовательности событий, приведших к сбою, выявляется *первопричина* - событие, нештатное событие, либо нарушение процедур эксплуатации. Для того, чтобы избежать повторения такого же сбоя в будущем, формулируются корректирующие действия - это могут быть изменения в процедуре эксплуатации, обучение персонала, исправления ошибок программного кода, а также реализация новой функциональности информационно-управляющих систем.

Каждое корректирующее действие оформляется в виде заявки в системе Jira, назначается на ответственное лицо для приоритизации и исполнения. Осуществляется периодический контроль за прогрессом выполнения заявок.

Взаимосвязь информационно-управляющих систем показана на Рисунке 1.6

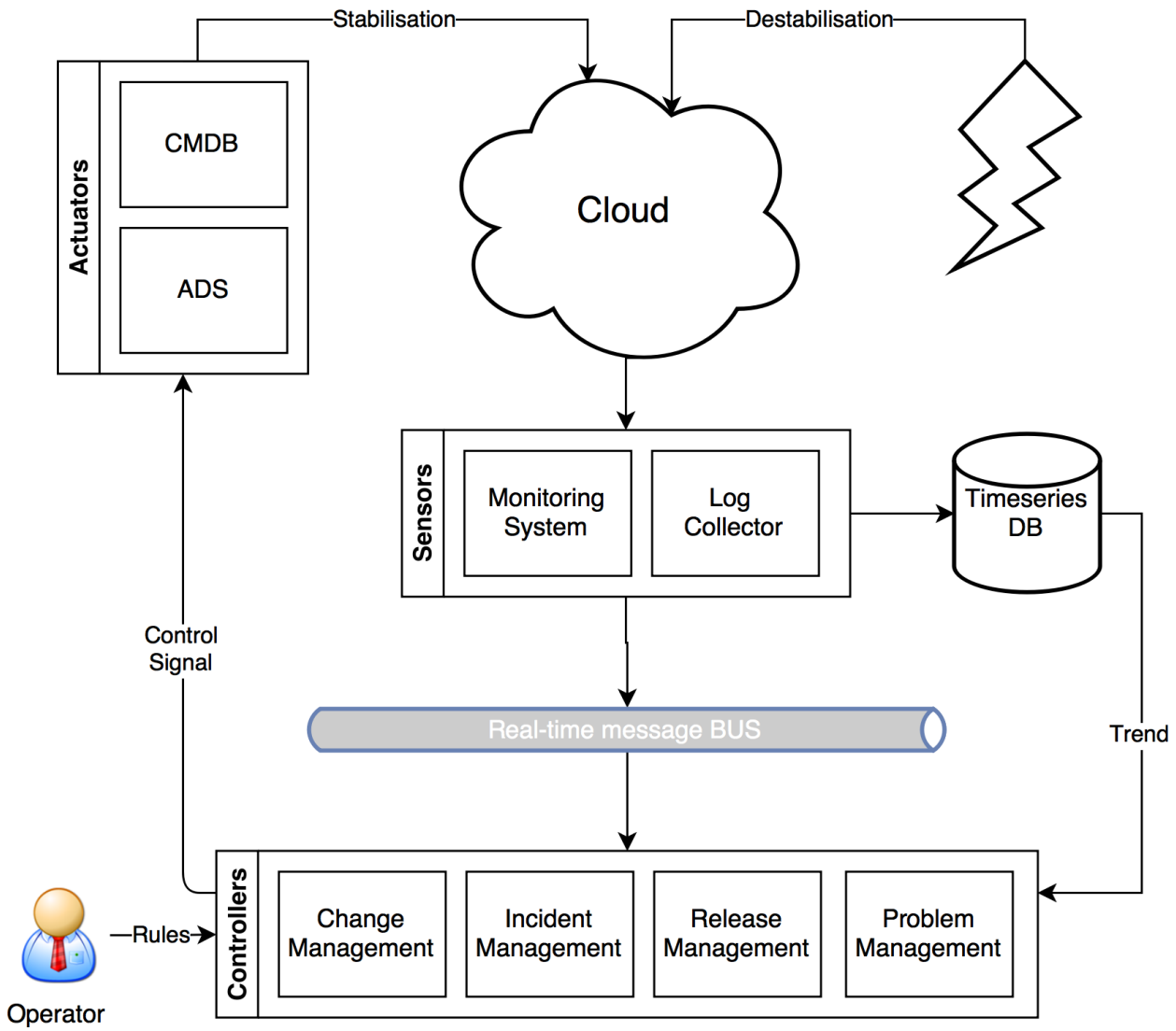

Рисунок 1.6: взаимосвязь информационно-управляющих систем в процессе эксплуатации “облачного” сервиса компании RingCentral

Одним из сценариев работы комплекса информационно-управляющих систем является управление емкостью вычислительных ресурсов. В процессе эксплуатации “облачного” сервиса количество запросов пользователей может изменяться в связи с появлением новых пользователей и отказом от использования сервиса старыми. Также на количество запросов может влиять меняющийся профиль использования сервиса одними и теми же пользователями.

При меняющемся профиле нагрузки система должна в любой момент времени удовлетворять требованиям по доступности всей системы (*reliability of system*, R_s). Растущая нагрузка дестабилизирует систему, повышая вероятность отказа в обслуживании из-за выхода из строя одного или нескольких серверов

приложений из-за нехватки ресурсов сервера. Наиболее распространенным случаем является нехватка оперативной памяти или вычислительных ресурсов центрального процессора.

Мониторинговая система обнаруживает превышение занятой оперативной памяти либо использованных вычислительных ресурсов процессора выше порогового значения и генерирует соответствующее мониторинговое событие. Для того, чтобы сохранить уровень доступности системы на необходимом уровне, требуется принять решение о том, какое количество ресурсов (серверов) необходимо добавить в систему. Это вычисляется с помощью следующей формулы:

$$R_s = \sum_{i=1}^{n} (1 - \prod_{j=1}^{m_j} (p_{ij}/2))/n, \qquad (1.1)$$

где R_s - требуемая общая надежность системы, p_{ij} - вероятность выхода из строя или перегрузки одного из серверов приложений (вычисляется на основе опытных данных и является фиксированной для сервера приложений, использование ресурсов которого не привышает пороговых значение, а также для тех, у которых это значение превышено), n - количество сегментов системы, m - количество серверов приложений в сегменте. Вероятность выхода из строя сервера приложений делится на коэффициент (в данном случае 2) для того, чтобы компенсировать резервирование вычислительных ресурсов в системе (в данном случае двукратное резервирование).

Производится расчет количества серверов приложений при котором общая надежность системы оказывается в пределах заданных значений. Добавление нового сервера приложений - это изменение в системе и для его выполнение требуется пройти процесс управления изменениями. В случае, если надежность системы отклонилась от требуемого уровня незначительно, либо если требуется убрать из системы лишние сервера приложений для понижения слишком высокого значения надежности системы, такое изменение делается планово. В случае же

если показатель надежности системы сильно отклонился от требуемого, изменение делается как экстренное в рамках процесса управления сбоями.

1.4 Выводы по главе 1

В первой главе приведено определение “облачных” услуг, данное национальным институтом стандартов и технологий США, сделан краткий экскурс в историю развития облачных услуг, первым крупным представителем которых считается компания SalesForce, основанная в 1999 году. Приведены оценки аналитических агентств Gartner и FinancesOnline о тенденциях развития облачных услуг, такие как рост количество людей, занятых в этой индустрии с 206 миллионов человек по всему миру в 2018 году до 380 миллионов к 2021 году. Приведен квадрант аналитического агентства Gartner, показывающий лидеров облачных услуг (телекоммуникации).

Исследована типовая архитектура глобально распределенной вычислительной сети, выявлены подходы к проектированию архитектуры: резервирование серверов приложений и баз данных, сегментирование данных, единая точка входа с маршрутизацией запросов к необходимому сегменту данных.

Научно обоснован выбор глобально распределенной вычислительной сети компании RingCentral в качестве объекта исследования на основании следующих фактов:

- компания RingCentral (США) является одной из ведущих в мире в сфере облачных телекоммуникационных услуг по результатам исследования аналитического агентства Gartner
- вычислительная сеть компании RingCentral имеет типовую архитектуру с сегментами, расположенными в центрах обработки данных по всему миру
- вычислительная сеть управляет информационными потоками между сегментами данных и внутри сегментов данных и использует стандартные механизмы репликации, аварийного и штатного переключения между основными и резервными серверами приложений и базами данных

Приведен аналитический обзор программных средств и информационных процессов эксплуатации “облачных” сервисов, основанных на лучших практиках ITIL (управление изменениями, управление сбоями, управление версиями обновлений, мониторинг). Приведены основные сценарии использования данных информационно-управляющих систем. Показана необходимость их интеграции на примере процесса планирования вычислительных ресурсов, который требует взаимодействия системы мониторинга, систем управления изменениями, сбоями и версиями обновлений.

Глава 2. Разработка информационной модели интеграции разнородных ИУС

2.1. Сравнительный анализ существующих методов и моделей интеграции данных

Интеграция данных - задача, давно известная. Doan A. H., Halevy A., Ives Z. в своем труде Principles of data integration [8] описывают цель интеграции данных как связывание данных из разнородных источников, контролируемых разными людьми, в одну схему. Они выделяют 3 области применения принципов интеграции данных: корпоративные приложения, научные данные и данные в сети интернет. Кратко рассмотрим каждую из них.

Неотъемлемой частью любой компании являются ее сотрудники, численность которых, для больших компаний может достигать десятков тысяч человек. Работа подобного большого количества сотрудников для достижения общей цели компании требует организации - формирования групп (отделов), выполняющих определенные функции. Из-за специфики работы, каждый отдел зачастую предпочитает использовать свой собственный узко-специализированный набор информационных систем. Таким образом по мере роста и развития организации, в компании появляется набор информационных систем, работающих независимо друг от друга.

В то же время, отделы компании работают над общими целями - так или иначе задачи разных команд связаны между собой. Возможность получать общую отчетность и прослеживать цепочки задач от отдела к отделу рождают необходимость в интеграции данных.

Группы исследователей в разных частях земного шара независимо друг от друга получают данные о последовательностях ДНК, особенностях поведения растений и животных, физических и химических свойствах новых материалов. Объект исследования один и возможность интеграции этих данных может дать важную дополнительную информацию для анализа.

Третья область применения принципов и методов интеграции данных - данные, которые появляются ежесекундно в глобальной сети Интернет. Это данные о товарах в различных интернет-магазинах, данные о публичных местах и

общественном транспорте и так далее и так далее. Источником данных могут быть частные лица, коммерческие организации, муниципальные и государственные структуры. Возможность получить сводную информацию об одном и том же объекте позволяет отфильтровать потенциально недостоверную информацию, а также избавляет от необходимости сложного поиска полных данных в разных источниках.

В чем сложность интеграции данных?

Во-первых, можно выделить сложность с логикой интеграции. Данные, которые очевидно относятся к одному и тому же объекту - большая редкость, зачастую разные источники данных по-разному называют один и тот же объект, что делает проблематичным сопоставление этих объектов. Более того, данные одного источника могут относится к части объекта, который описывается в другом источнике данных. Примером здесь может быть указание различных названий одних и тех же организаций и высших учебных заведений в научных публикациях в случае их переименования, что затрудняет индексирование в базах РИНЦ и Scopus.

Во-вторых, выделяется технологическая сложность. В зависимости от способа и цели интеграции (они рассмотрены далее), возникают технические сложности, которые связаны с тем, что источниками данных могут являться совершенно разные платформы, а формирование запроса к разным платформам может оказаться технически сложной задачей. Также стоит учесть, что источники данных - это независимые системы и каждый будет обрабатывать такой гетерогенный запрос со своей скоростью и своей надежность, что требует правильной обработки ошибок и объединения результатов параллельных вычислений.

В-третьих, выделяется социальная сложность. В случае с интеграцией данных корпоративных информационных систем, отдельной сложностью может стать обнаружение необходимой информационной системы, содержащей нужные данные, а также получение разрешения на доступ к этим данным. Ограничениями

могут быть политика информационной безопасности в компании, охрана личных данных пользователей и клиентов компании.

Таким образом, интеграция данных - это сложная задача. И будет ли она когда-нибудь решена полностью? Идеальной системой для интеграции данных представляется такая, которая смогла бы самостоятельно обнаружить и использовать новый источник данных и дала бы возможность формулировать такие запросы к данным, которые бы позволили получать данные как из старых так и из этого нового источника данных. В настоящее время создание подобной системы интеграции не представляется возможным и, судя по вышеописанной сложности, подобная задача потребовала бы участие сильного искусственного интеллекта.

Вместо этого, задачами, которыми занимается наука, исследующая интеграцию данных, являются следующие: во-первых, это снижение трудоемкости интеграции, а также повышение надежности интеграционной системы и доступности данных.

Выделяют 2 архитектурных подхода к интеграции данных. **Виртуальная** интеграция и интеграция с использованием **хранилища** данных.

Отличительной особенностью виртуальной интеграции является обращение к источникам данных по мере необходимости, как показано на Рисунке 2.1

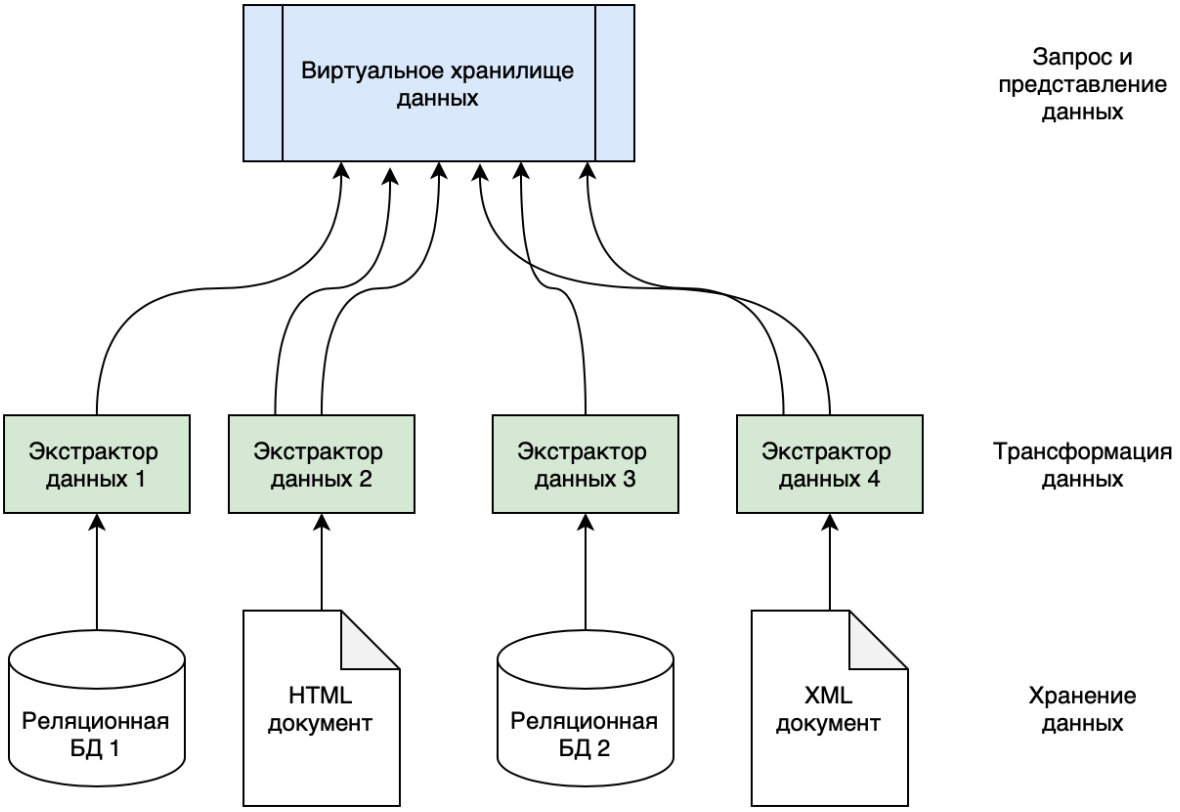

Рисунок 2.1: архитектура виртуальной интеграции данных.

При получении запроса интегрированных данных системой интеграции, она анализирует этот запрос и формирует необходимое количество подзапросов к источникам данных. Полученные данные далее, в среде системы интеграции, объединяются и выдаются клиенту.

В случае использования хранилища данных, данные, независимо от запросов клиентов, в постоянном режиме загружаются из источников данных и сохраняются в обработанном виде в хранилище данных. При получении системой интеграции запроса, она обращается не к источникам данных, а к хранилищу данных, как показано на рисунке 2.2.

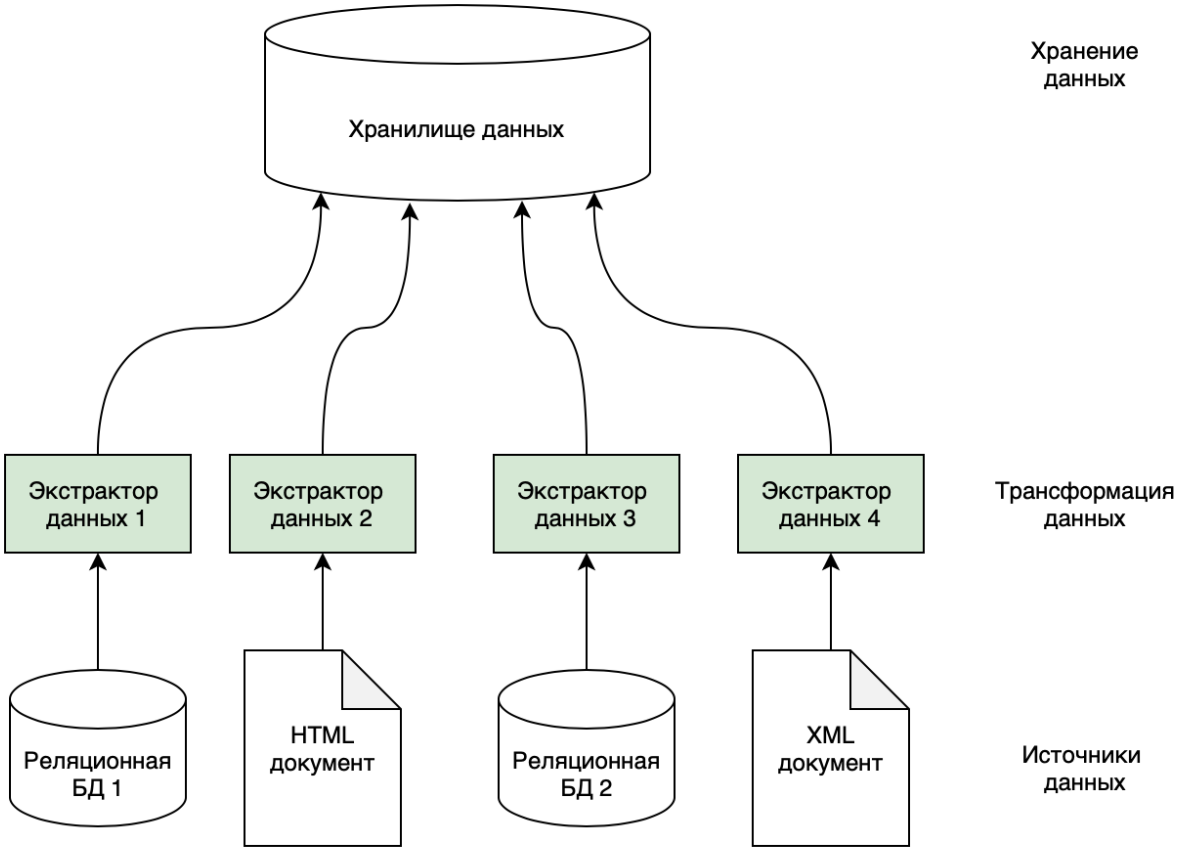

Рисунок 2.2: интеграция данных с использованием хранилища данных.

Два архитектурных подхода имеют свои преимущества и недостатки, но стоит обратить внимание, что в обоих случаях сохраняется сложность объединения семантически различных данных, происходящих из гетерогенных источников.

Недостатком использования хранилища данных является необходимость в проектировании структуры данных для этого хранилища, а также учитывать требования по отказоустойчивости и производительности этого хранилища, что может занять значительное время и средства.

Вторым недостатком хранилища данных является то, что данных в нем находятся в неактуальном состоянии и немного “запаздывают” по сравнению с источником данных, так как загрузка обновленных данных в хранилище происходит только после того, как они будут сохранены и даже в случае если процесс загрузки запускается сразу после сохранения данных, он занимает

какое-то время. Чаще же загрузка данных осуществляется по расписанию, что создает существенное “запаздывание”.

Этих недостатков лишена архитектура виртуальной интеграции, однако у нее есть другие недостатки. Во-первых, каждый запрос к интеграционной системе генерирует в свою очередь запрос или запросы к источникам данных, что может создать непредвиденную большую нагрузку и вывести источник данных из строя в случае если источник данных не был спроектирован для обработки такого большого количества запросов.

Вторым недостатком виртуальной интеграции является низкая скорость ответа. Работа с удаленными источниками данных - это запрос по сети, который, к тому же, может быть неуспешным и потребуется повторный запрос. В случае же обращения к хранилищу данных, интеграционная система может быть спроектирована таким образом, чтобы ответ на обращение происходил с минимальной задержкой благодаря размещению хранилища в одной подсети с интеграционной системой.

Исследование проблем интеграции данных помимо рассмотрения принципиальной архитектуры интеграции требует также рассмотрения технологии интеграции.

Конечным результатом интеграции данных является возможность эффективно (за короткое время) получить выборку, составленную из данных, находящихся в разных источниках, по определенным параметрам, относящимся также к данным из разных источников. Это требует следующих этапов:

1. Формирования интеграционной модели данных, описывающей итоговую схему данных, состоящую из набора различных сущностей, их атрибутов и взаимосвязей. Эта модель данных моделирует предметную область, в которой происходит интеграция.

2. Описания источников данных. Включает описание доступа к данным, схемы данных источника и правила конвертации этих данных в интеграционную модель. Позволяет определить куда обращаться за какими данными, а также сформировать запрос на языке источника данных

3. Разработка механизмов сопоставления гетерогенных данных. Наиболее частым способом сопоставления является выявление строковых ключей с последующим сравнением строк.

4. Для виртуальной интеграции требуется разработка механизма преобразования запроса пользователя к интегрированным данным в набор запросов к источникам данных. А также механизма выполнения запросов к гетерогенным источникам данных.

Стоит также упомянуть сложность интеграции данных, связанную с неопределенностью данных. Говоря о данных принято понимать определенное состояние объекта (рост человека, температура воздуха на улице) и большинство баз данных проектируются для того, чтобы описывать это состояние. Однако в реальности мы часто сталкиваемся с неопределенностью данных (волосы преступника либо светлые, либо рыжие; температура воздуха колеблется от одного значения до другого). В случае же необходимости интеграции данных из различных источников, неопределенность может усиливаться за счет того, что один и тот же параметр объекта может иметь разные значения в разных базах.

Исследование интеграции неопределенных данных находится только в начале своего пути. Однако на данный момент выделяются два подхода. В первом для работы с неопределенными данными, интеграционную модель данных расширяют таким образом, чтобы описывать не одно состояние мира, а набор возможных состояний. В этом случае сохраняется полнота данных.

Во втором подходе вводится вероятностная модель. Интеграционная модель стремится устранить неопределенность, для этого для каждого источника данных, потенциально создающего неопределенность данных, вводится вероятность, с которой данные из этого источника будут приняты за истинные.

2.2. Концепция создания информационной модели интеграции без модификации структур БД

В крупной международной IT компании, предоставляющей облачные услуги в сфере телекоммуникаций, существует потребность в обслуживании огромного парка вычислительных ресурсов (физических серверов, виртуальных машин, программного обеспечения [9]), глобально распределенных в удаленных центрах обработки данных на разных континентах. В данной работе использованы реальные сведения об американской компании RingCentral [10], предоставляющей облачные телекоммуникационные услуги в режиме 24/7 в 27 странах Северной Америки, Западной Европы и Юго-Восточной Азии.

Компания обслуживает более 10 тысяч серверов, расположенных в 6 удаленных центрах обработки данных. С каждого удаленного сервера собирается информация в централизованную систему мониторинга событий. Суммарный поток данных составляет более 13 тысяч значений в секунду. Каждое событие анализируется для выявления аномального состояния, например, завышенного расхода ресурсов ЦПУ. Таких аномалий в системе фиксируется и устраняется около 3 тысяч в день.

Аномалия может привести к потере резервирования вычислительных ресурсов, снижению производительности и даже к отключению сервиса и отказу в обслуживании пользователей. В среднем фиксируется 3 таких инцидента в день.

Повторяющиеся инциденты – это проблема, которая требует выявления первопричины и исправления базовой ошибки, после чего система должна быть обновлена для предотвращения повторения инцидента в будущем. Среднее количество изменений в глобально распределенной инфраструктуре компании RingCentral, включая аварийное исправление ошибок и плановое добавление новых возможностей системы, – 30 в день, причем ежегодно наблюдается рост практически в 2 раза (рис. 2.3).

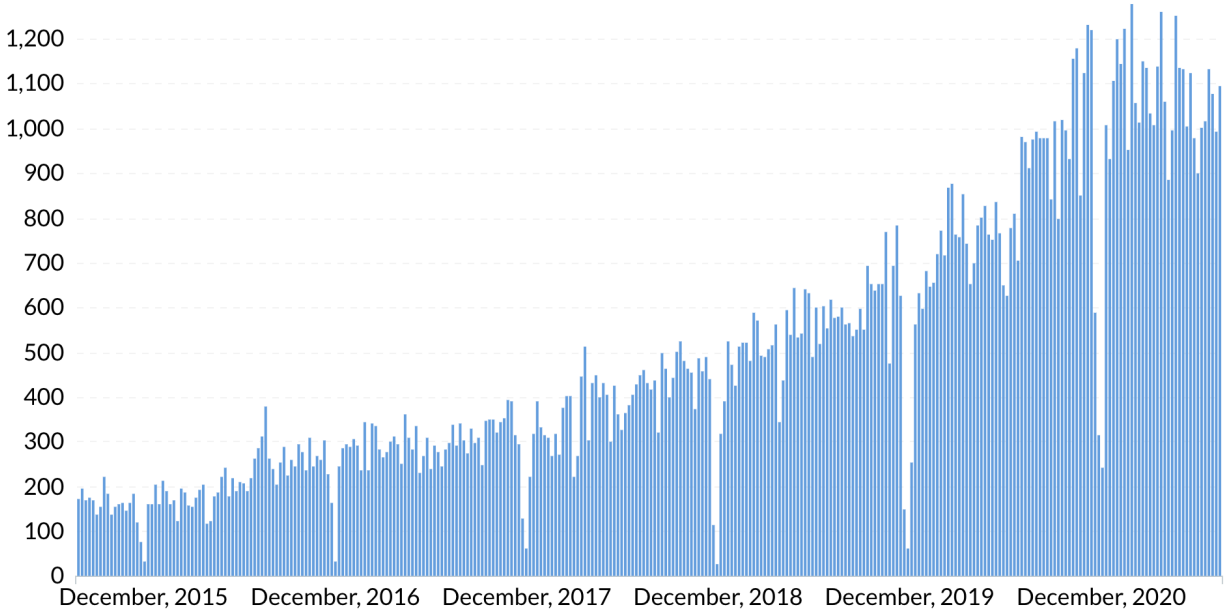

Рисунок 2.3: количество изменений в неделю в сервисе компании RingCentral со второй половины 2015 года по первую половину 2021 года

Учитывая глобальный масштаб всей инфраструктуры, большое количество событий, аномалий и изменений в системе, а также динамику их роста, возникла объективная необходимость повышения эффективности разрозненных процессов эксплуатации посредством их интеграции.

Анализ процессов эксплуатации показал:

а) они связаны логически, имея в виду что один процесс может инициировать другой (изменение в системе порождает новые события, события сигнализируют об инциденте, инцидент инициирует процесс анализа проблемы, который в свою очередь заканчивается внесением определенных изменений в систему);

б) процессы связаны с точки зрения данных, задействованных в этих процессах (результаты одного процесса являются исходной информацией для другого, как показано на рис. 2.4).

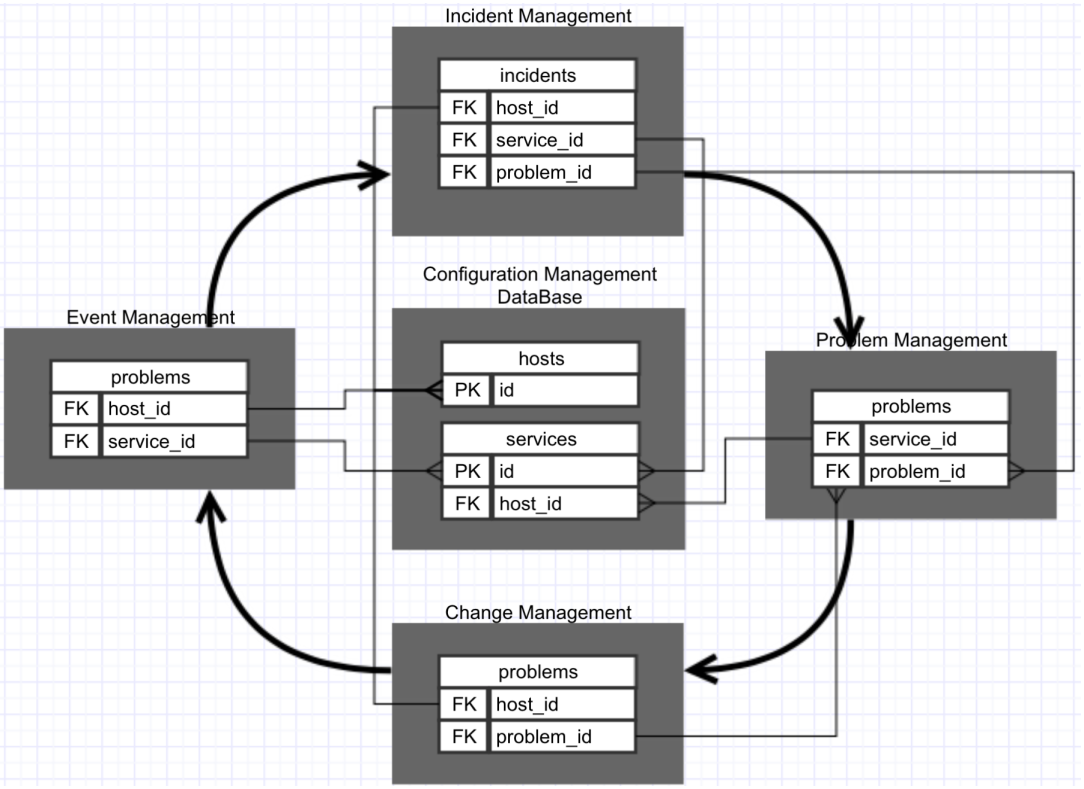

Рисунок 2.4: взаимосвязь процессов эксплуатации облачных сервисов

Учитывая выявленные взаимосвязи процессов (рис. 2.4), сформулированы следующие требования к задаче интеграции с целью повышения эффективности эксплуатации:

1) требуется уметь быстро и надежно определить наиболее подходящее время для внесения изменений в систему с учетом других планируемых обновлений, а также текущего состояния системы – наличия ошибок и проблем;

2) для эффективного контроля за проблемами требуется видеть, на каком этапе находится решение проблемы, включая недавние и текущие инциденты, связанные с данной проблемой, планируемые изменения для устранения причины проблемы;

3) для быстрого и правильного устранения проблемы требуется видеть взаимосвязь между недавними изменениями и текущими аномалиями, чтобы в случае ошибок быстро отменить изменение, вызвавшее отказ.

Одним из решений поставленной задачи является внедрение готового интегрированного продукта, например BMC Cloud Management или VMware Cloud Management [1]. Готовые решения крупных поставщиков имеют два существенных недостатка: дороговизна и необходимость использовать информационные системы одного производителя. Более того, переход на новые средства интеграции является революционным и крайне затруднен в условиях динамично развивающихся информационных технологий.

В компании RingCentral было принято решение интегрировать имеющиеся информационные системы различных поставщиков, которые зарекомендовали себя как надежные программные средства и накопили в базах данных многолетнюю историю всех изменений инфраструктуры. Главными преимуществами такого подхода являются минимальные затраты на интеграцию и сохранение привычных технологий работы технических специалистов в командах без необходимости переобучения новым интеграционным проектам. Единственный недостаток состоит в необходимости дополнительного программирования веб приложения и незначительной модификации структуры каждой базы данных, подверженной интеграции.

В данной работе предложен новый подход к интеграции глобально распределенных информационных систем на основе идеи проектного и ресурсного срезов данных.

Главной интеграционной сущностью является проект – совокупность задач, направленных на достижение поставленной цели. Задачи отдельных команд учитываются в разных информационных системах и объединяются в сущность «проект». Проектный срез данных – это совокупность всех активностей различных команд в рамках одного проекта.

Другой интеграционной сущностью является «ресурс» – это учетная единица сотрудника, команды, организации, либо единица оборудования, привлекаемая к выполнению проекта. Ресурсный срез – это совокупность всех активностей и задач, в которых задействован данный ресурс.

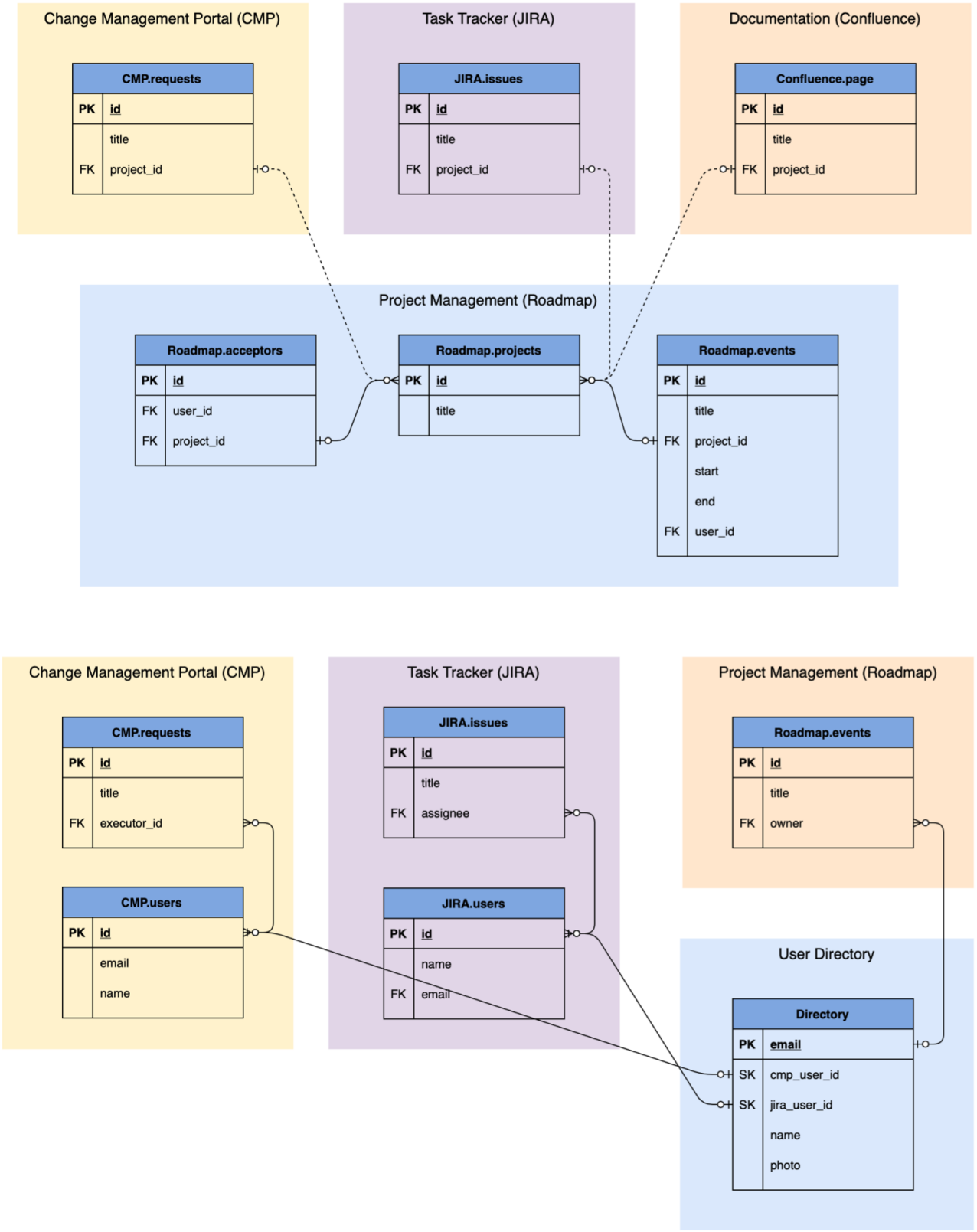

Рисунок 2.5: модели данных для проектного и ресурсного срезов

Ресурсные и проектные срезы данных на основе установления ссылочных связей (рис. 2.5) – это известный способ интеграции различных информационных источников для дальнейшей обработки, анализа и визуализации [12].

В глобально распределенной архитектуре информационных систем, использующих гетерогенные реляционные и нереляционные базы данных, автоматическое объединение таблиц БД физически невозможно. Создание специализированного интеграционного приложения является трудоемкой задачей. В данной работе предложен следующий подход:

Использован известный стандарт программных интерфейсов API доступа к внутренним данным информационных систем, в частности для поиска всех задач по идентификационному номеру ресурса или проекта. В качестве такого интерфейса использован стандарт REST API, который поддерживается всеми современными информационными системами на уровне приложений или программных библиотек.

2.3. Практическая реализация информационной модели интеграции в компании RingCentral

На международной конференции DCCN [13] были представлены результаты интегральной оценки производительности и качества облачных информационных сервисов на примере телекоммуникационной компании RingCentral [10].

Для обслуживания такой большой глобально распределенной инфраструктуры в департаменте по эксплуатации сформированы команды технических специалистов в конкретных областях:

1. Архитектура и виртуализация серверов;
2. Сетевое оборудование;
3. Системные администраторы ОС и ПО;
4. Администраторы БД и их приложений;
5. Телекоммуникационное оборудование и специализированное ПО;

6. Мониторинг облачной инфраструктуры;

7. Информационная безопасность.

Каждая из команд международной компании использует свои специализированные средства автоматизации бизнес-процессов и физически расположена в разных офисах разных стран и континентов. В этих условиях выполнение совместных проектов и программ, требующих вовлечения разных команд, и управление ими становится затруднительным из-за необходимости использования исполнителями и руководителями проектов большого количества разрозненных информационных систем. Возникает объективная потребность интеграции существующих систем с целью повышения эффективности управления и информационного обмена между всеми подразделениями глобально распределенной корпоративной сети.

Для реализации поставленной задачи разработана информационная модель интеграции с использованием инструментальных средств моделирования БД Gliffy [14, 15] (рис. 2.7), в основе которой положена предыдущая работа авторов [16].

Согласно новой интегрированной структуре данных, облачная инфраструктура состоит из системных единиц (System Unit), обладающих определенными свойствами. Каждая системная единица имеет системные связи (System Relation). Системная связь объединяет две системных сущности так, что их класс соответствует типу данной связи. В интеграционной модели данных реализованы следующие реляционные связи с логикой «родитель-потомок»: сетевой интерфейс системной единицы, физическое соединение, сетевой интернет интерфейс, роутинг интерфейс, сервис доменных имен, сервисный интернет интерфейс, сервисное соединение, системное соединение, отображение портов, реверсивное проксирование.

Системный компонент (System Component) является единственным родителем для всех системных единиц, обладающих одинаковыми свойствами и их значениями. Системный компонент также может представлять сущность без

дочерних системных единиц – пользовательский агент (User Agent Client). Реализованы следующие свойства системного компонента:

1. Сервисная роль: сервисный шлюз, реверсивный прокси, база данных, кэш, ведущий кластера, ведомый кластера, поставщик данных, шлюз партнера и прочие;

2. Тип: виртуальная машина, контейнер, сервер, устройство хранения данных, коммутатор, маршрутизатор, источник питания, стойка, центр обработки данных и прочие;

3. Производитель, поставщик;

4. Модель, тип, торговая марка, брэнд;

5. Операционная система, прошивка, версия.

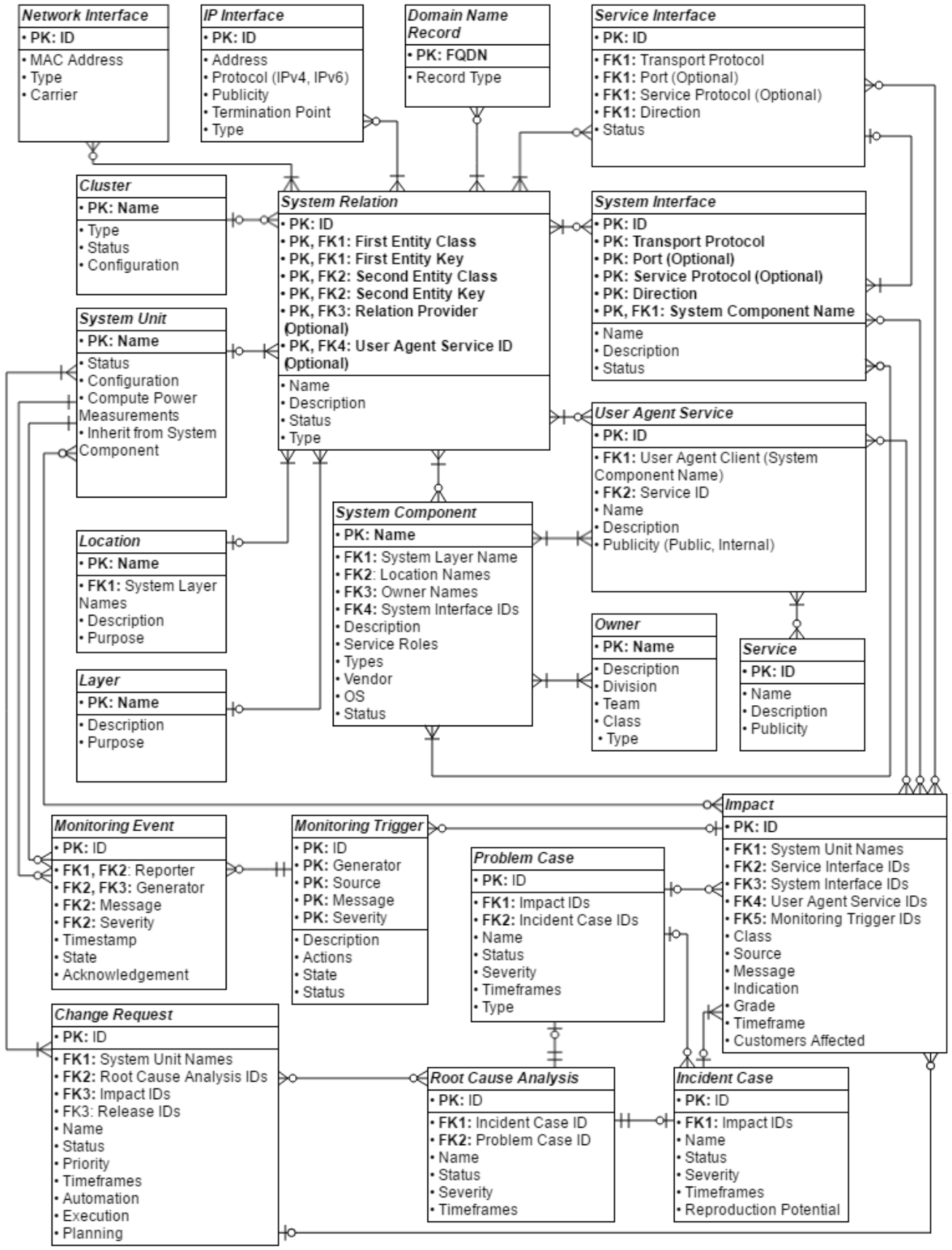

Рисунок 2.6: интеграционная модель данных

Системный компонент может быть представлен в нескольких экземплярах (Location), имеющих разные предназначения: обслуживание конечных пользователей, тестирование, разработка. Системный компонент представляет один и только один системный уровень (Layer), выполняющий заданную задачу по обработке пользовательских запросов.

Пользовательский сервис (User Agent Service) – это абстрактная сущность, призванная отображать системные связи (сервисное соединение, системное соединение, отображение портов, реверсивное проксирование), используемые для предоставления того или иного сервиса компоненту типа пользовательский агент (User Agent Client).

Системная единица может произвести мониторинговое событие (Monitoring Event) при помощи генератора событий (Generator). У мониторингового события может быть только один генератор (системная единица). Мониторинговый триггер (Monitoring Trigger) это виртуальная сущность, введенная для отображения мониторинговых событий, которые уже произошли (история мониторинга) и которые могут произойти в будущем (спецификация мониторинга). Мониторинговое событие, представленное мониторинговым триггером и отметкой времени, индицирует воздействие (Impact) на пользовательский сервис, системную единицу, системное или сервисное соединение.

Воздействие также может быть индицировано по пользовательским обращениям. Воздействие имеет временной диапазон и насчитывает следующие градации: прерывание, сокращение качества, обрыв сессий, потеря избыточности, потеря емкости. Воздействие может затрагивать пользователей или нет. Если воздействие не затрагивает пользователя, оно расценивается как потенциальное и должно принадлежать проблеме (Problem Case). Если же воздействие затрагивает пользователей, то оно должно принадлежать инциденту (Incident Case). Если по результатам расследования причин (Root Cause Analysis) инцидент имеет большую вероятность повторения, он должен стать частью проблемы.

Проблема имеет собственное расследование причин. Результатом расследования причин инцидента является экстренный запрос на изменение

(Change Request). Результатом расследования причин проблемы может являться как экстренный, так и запланированный запрос на изменение. В рамках запроса на изменение происходит обновление конфигурации системных единиц. Запрос на изменение может быть вызван не только расследованием причин проблемы или инцидента, вне зависимости от источников изменений они способны вызвать воздействие.

2.4. Выводы по главе 2

В данной главе произведен сравнительный анализ существующих методов интеграции данных. Приведены области, где актуально применение методов интеграции данных: корпоративные приложения, научные исследования, данные в сети Интернет.

Показано два архитектурных подхода к интеграции данных - виртуальная интеграция и хранилище данных. Приведены преимущества и недостатки каждого из подходов.

Также показаны этапы интеграции данных и упомянута открытая проблема интеграции (неопределенность данных).

Средствами интеграции решается задача повышения эффективности эксплуатации облачных сервисов крупной международной IT компании RingCentral (США) с глобально распределенной инфраструктурой, большим количеством изменений, аномалий, инцидентов и хронически повторяющихся проблем, а также быстрой динамикой роста данных показателей. Сделан анализ существующих процессов эксплуатации и выявлены взаимосвязи между ними. Разработана новая информационная модель интеграции данных, которая использована для создания программных приложений, внедренных в существующую глобально распределенную инфраструктуру компании и эффективно эксплуатируются для управления облачными сервисами.

Дальнейшей проработки требует вопрос интеграции других разрозненных средств автоматизации, используемых в различных процессах установки и

тестирования облачных сервисов, которые в настоящее время никак не связаны между собой.

Главной причиной низкой эффективности управления проектами в большой глобально распределенной IT компании является разрозненность корпоративных информационных систем. В данной работе исследована проблема повышения эффективности управления проектами, выполняемыми разными исполнителями в корпоративной сети с облачной архитектурой. Взамен дорогостоящих решений известных поставщиков, предложен нетрадиционный подход к интеграции имеющихся разрозненных информационных систем. Для этого реализовано логическое соединение сущностей по критериям «проект» и «ресурс» стандартными программными средствами API.

Предложенный подход к интегральной оценке эффективности корпоративных систем управления успешно внедрен в международной телекоммуникационной компании RingCentral с целью интеграции используемых информационных систем Atlassian Jira и Confluence, Jenkins, Roadmap, SalesForce и др.

Глава 3. Разработка программных приложений интеграции разрозненных ИУС

3.1. Методика оценки интенсивности технического обслуживания ГРИС.

Современные «облачные» сервисы, такие как электронная почта, телефония, видеосвязь и социальные сети, завоевывают все большую популярность. «Облачными» сервисами начинают пользоваться не только дома, в личных целях, но и на работе, как часть рабочего процесса. Предприятия заменяют свои традиционные информационные системы на «облачные».

Первым важным критерием для бизнеса при переходе на облачный сервис является уровень доступности - сервис должен отвечать на запросы в режиме «24/7» «при любой погоде», вплоть до локальных катастроф (землетрясение, ураган, падение метеорита). При этом качество предоставления сервиса, такое как время обработки запроса, должно оставаться неизменным даже при большом количестве одновременных обращений. Так, например, компания RingCentral Inc. гарантирует доступность своего телекоммуникационного сервиса на уровне 99,999%. Компания Microsoft - доступность Office 365 на уровне 99,9% [17].

Такой высокий уровень доступности достигается за счет высокой степени резервирования вычислительных ресурсов (рис. 3.1).

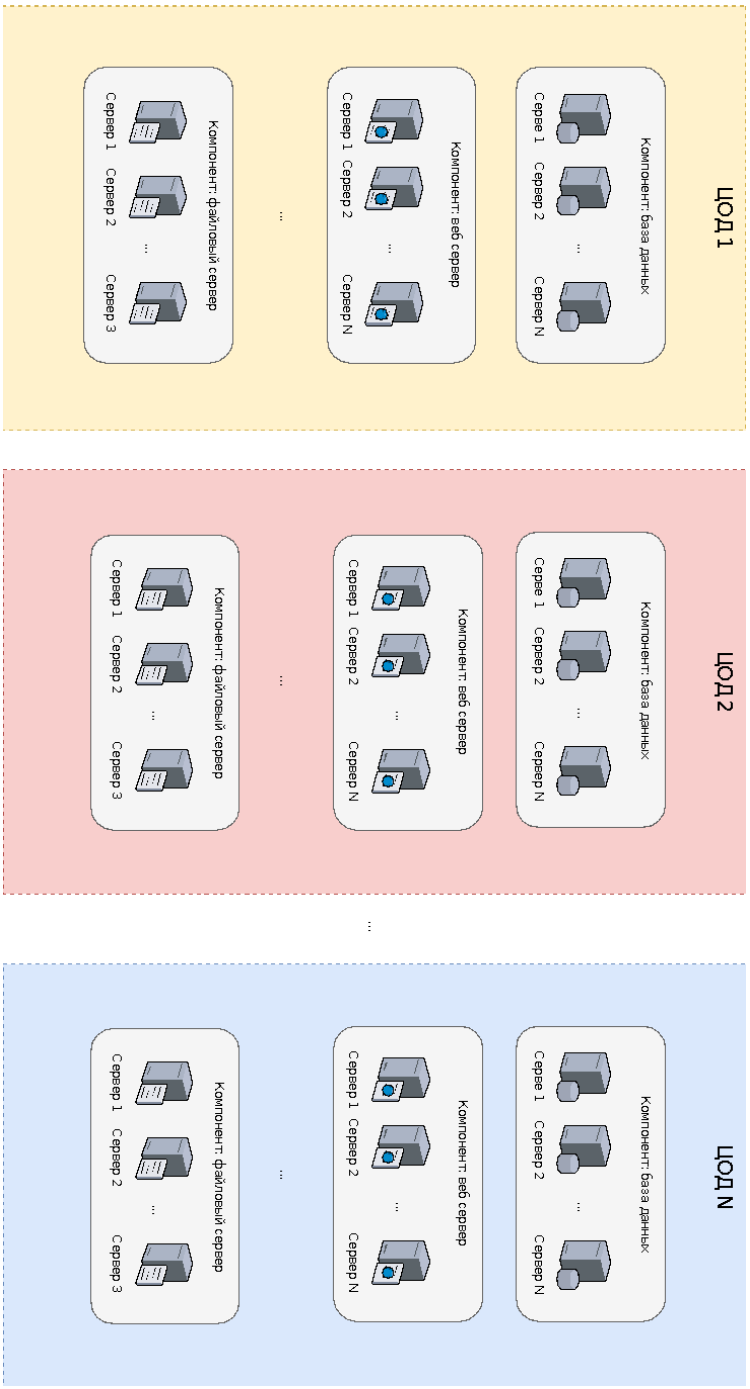

Рисунок 3.1: схема географически распределенной информационной системы

Второй важный критерий при выборе поставщика облачных услуг – набор предоставляемых им сервисов и богатство их функциональности, так называемый «пакет» услуг. Компании стремятся создавать новые сервисы и расширять функциональность, добавляя, например возможность отправить факсы через электронную почту или возможность получать информацию о расходах средств на звонки в реальном времени. Технологически это заключается в проектировании и

разработке новых алгоритмов вычислений и системами хранения данных. Появляются вычислительные машины или серверы, специально выделенные для реализации этих вычислений, серверы новых систем хранения данных. Группа серверов, реализующих одни и те же вычисления, функционально однородная группа, называется компонентом системы. В период с 2010 по 2017 годы у компании RingCentral Inc. значительно расширился «пакет» предоставляемых услуг: в дополнение к телефонии за это время у компании появились сервисы обмена мгновенными сообщениями, сервис анализа качества звонков в реальном времени и прочие. Количество компонентов облачного сервиса компании RingCentral Inc. с 2010 по 2017 год выросло с 50 до 200 [18].

Параллельно с ростом сложности информационной системы за счет добавления новых компонентов существует еще одна тенденция - рост скорости внедрения инноваций. Существенным конкурентным преимуществом является быстрая реакция на нужды пользователей и возможность быстро компенсировать потенциальное отставание от конкурентов [19].

Однако скорость вносимых изменений в ИС может негативно отразиться на предоставлении информационных услуг, так например 24 сентября 2010 года крупнейшая социальная сеть Facebook стала недоступна на 2,5 часа [20]. Инцидент был вызван плановым обновлением одного из компонентов системы и затронул всех пользователей. Чтобы не подвергать риску сразу всех пользователей, компании в настоящее время часто делят их на группы и выпускают обновление для них по очереди, отслеживая динамику состояние системы и удовлетворенности пользователей. Это называется поэтапное развертывание ПО.

Все больше компаний адоптируют «гибкие» методологии разработки, такие как Scrum, которые позволяют получать обновление функциональности для пользователей не раз в год, как раньше, при использовании методологии Waterfall, а раз в 3 недели. Существует и дальнейшая тенденция к ускорению доставки новой функциональности конечным потребителям – методология разработки Kanban допускает выпуск обновлений несколько раз в день.

В итоге, имея систему с а) большим количеством серверов для обеспечения необходимого уровня резервирования, б) большим количеством компонентов для обеспечения широкого набора услуг и в) большим количеством обновлений системы для обеспечения высокой скорости реакции на нужды пользователей, становится актуальным вопрос о том, каков предел интенсивности изменений в сложной глобально распределенной информационной системе [21, 22].

Целью данной главы является создание метода оценки интенсивности обслуживания глобально распределенной вычислительной системы, который позволяет определить потенциал увеличения скорости внесения изменений программного обеспечения в информационную систему.

В качестве объекта исследования выбрано одно из регулярных крупных обновлений ПО компании RingCentral «Версия ПО 10.1» [23]. Новая версия ПО состоит из 141 отдельного изменения и была установлена на ИС в период с февраля по сентябрь 2018 года. На рисунке 3.2 показана часть этапов развертывания, пришедшихся на июль 2018 года:

Рисунок 3.2: этапы развертывания изменений в рамках «Версии ПО 10.1» компании RingCentral Inc. в июле 2018 года

Изменение – это переход системы из старого состояния в новое, обновленное. 16-го августа 2018 года компания RingCentral выпустила крупное обновление своего сервиса обмена мгновенными сообщениями Glip, добавив в него возможность совершать голосовые звонки. Об этом обновлении было объявлено заранее, оно обсуждалось в средствах массовой информации и в профессиональной среде. Это был «маркетинговый ход», способ привлечь новых клиентов. Подобные обновления случаются и у других компаний, предоставляющих «облачные» сервисы, однако чаще обновления бывают не такие «большие» и происходят «тихо», например, компания Microsoft выпустила 40 обновлений Office 365 за 2017 год [24]. Это делается для того, чтобы такое «тихое» обновление ПО можно было бы так же «тихо» откатить, снижая степень разочарования клиентов в случае, если, например, обновленная система начнет работать со сбоями. Впрочем, даже такой откат – это серьезная проблема для компании, так как сбивает производственный график. Для решения этой проблемы компании дробят обновления на максимально «мелкие», откат которых не так болезнен. Так, компания RingCentral выпускает 80 «мелких» обновлений в месяц. Это, например, установка новой версии операционной системы на серверах или новой версии приложения, отвечающего за обработку телефонного трафика. Такого рода обновление применяется на минимальное количество компонентов системы за раз.

Например, типичные этапы развертывания обновления ПО в компании RingCentral следующие [25]:

1. Обновление серверов, обслуживающих небольшую группу пользователей, явно изъявивших желание получать обновления первыми, а также мало активных пользователей, для которых сбой в работе «облачного» сервиса будет наименее ощутим. Такую группу часто называют «бета-пользователи».

2. Наблюдение за системой. Если количество жалоб от бета-пользователей не возросло после изменения, а также на основании данных технического мониторинга принимается решение о том, можно ли продолжать развертывание изменения.
3. Обновление серверов, обслуживающих часть пользователей, активно использующих сервис. Это случайная выборка обычных пользователей.
4. Наблюдение за системой. По аналогии с пунктом 2, но с дополнительным вниманием к изменениям в производительности системы принимается решение о том, можно ли продолжать развертывание.
5. Обновление всех оставшихся серверов, относящихся к обновляемым компонентам.

Таким образом, время внесения одного изменения в систему – это сумма временных интервалов, необходимых на каждый из этапов (формула 3.1)

$$T_C = \sum_{i=1}^{n} T_{Pi}, \qquad (3.1)$$

где T_C - время внесения одного изменения; T_P – время одной фазы изменения.

Однако, сложная глобально распределенная информационная система несмотря на функциональную декомпозицию по компонентам продолжает оставаться единой системой. В обработке запроса пользователя может быть задействовано большое количество компонентов, таких как:

1. Серверы аутентификация пользователей;
2. Серверы авторизации пользовательского запроса;
3. Серверы взаимодействия с базой данных;
4. Базы данных;
5. Журналы запросов;
6. Серверы приложений, формирующие ответ пользователю.

В случае внесения изменения в интерфейс компонента API, его взаимодействие с другими компонентами в процессе обработки пользовательского

запроса может измениться и даже сломаться. Наиболее типичным примером здесь является изменение структуры таблиц в базе данных. Подобное изменение может быть безопасно выполнено только после обновления серверов приложений, взаимодействующих с этой базой данных – сервера приложений должны заранее «научиться» работать с новой структурой таблиц данных. Таким образом существуют изменения, которые должны быть выполнены в строго определенной последовательности. Появляются зависимости между развертываниями и эти зависимости ограничивают интенсивность внесения изменений, так как зависимые развертывания нельзя делать параллельно, а нужно делать последовательно.

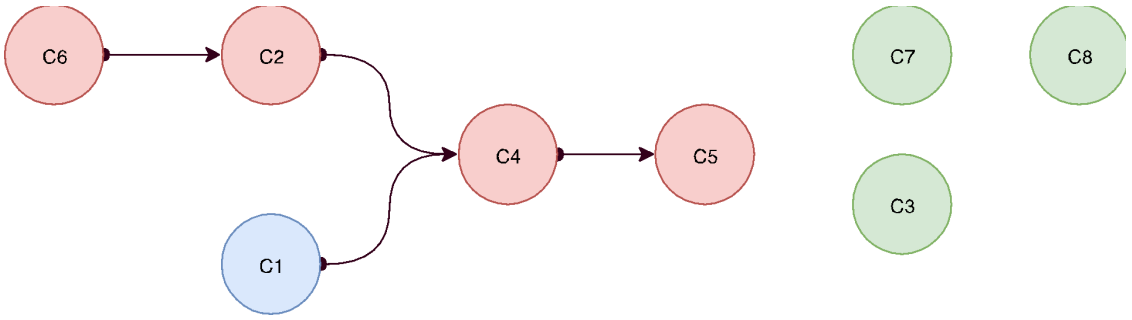

Рисунок 3.3: пример зависимостей в группе из 8 изменений. Зеленым обозначены независимые изменения, красным – самая длинная цепь зависимых изменений, синим – остальные.

На рис. 3.3 показан пример зависимостей в группе из 8 изменений. Красным цветом выделена группа из 4 изменений которая является самой длинной цепочкой зависимостей. Эти 4 изменения можно применять только одно за другим. Зеленым – те, которые можно применять независимо от остальных. Синим выделено изменение C1, оно должно предшествовать изменению C4, но не является частью самой длинной цепи изменений. Таким образом, время внесения серии (множества) изменений ПО может быть записано следующим образом:

$$T_{\{C_1 \dots C_m\}} = \sum_{i=1}^{n} T_{Ci'} \qquad (3.2)$$

где $T_{\{C_1 \ldots C_m\}}$- время внедрения множества изменений, n - размер самой длинной цепочки зависимостей

В таблице 3.1 представлена самой длинная цепочка зависимости между 141 изменением в рамках инсталляции «Версии ПО 10.1» в ИС компании RingCentral Inc., составляющими одно крупное обновление ПО.

Таблица 3.1: пример экспериментальных данных

Порядок применения	Название изменение	Суммарная длительность всех этапов изменения
1	10.1 SDI (Configuration Update for JWS)	5 дней
2	10.1 Billing Release	4 дня
3	10.1 In-place upgrade	5 дней
4	10.1 Swat Patch 1	3 дня
5	10.1.1 deployment sprint	3 дня
6	10.1.3 Deployment sprint	3 дня
7	10.1.6 deployment sprint	3 дня
8	10.1.7 deployment sprint	3 дня
9	10.1.8 deployment sprint	3 дня
10	10.1.9 Deployment sprint	3 дня
11	10.1.10 Deployment sprint	3 дня

Таким образом, минимальная длительность обновления равна сумме длительностей применения каждого из изменений в самой длинной цепочке, что составляет 38 дней. В реальности же «Версия ПО 10.1» была развернут в период времени с 15 февраля по 7 сентября 2018 года, что составляет 134 рабочих дня.

Соответственно скорость развертывания потенциально может быть увеличена примерно в 3,5 раза.

3.2. Интегрированный интерфейс визуализации процесса автоматического развертывания виртуальных сервисов в облачной инфраструктуре

Облачная инфраструктура позволяет заменить физические серверы на виртуальные без потери производительности. Технологии облачных вычислений позволили IT компаниям строить сложные компьютерные комплексы достаточно быстро. Например, на одном физическом сервере в настоящее время можно развернуть 40-100 виртуальных машин, при этом достигается экономия ресурсов ЦОД, таких как полезная площадь автозалов, количество потребляемой электроэнергии, систем кондиционирования воздуха и других. Затраты на облачную инфраструктуру ЦОД примерно в десять раз меньше стоимости аналогичного оборудования при создании традиционных автозалов на физическом оборудовании.

Значительная часть IT компаний, занимающих лидирующее положение на рынке телекоммуникационных и информационных услуг, используют облачную инфраструктуру как основную технологию построения ЦОД в различных географических зонах земного шара. Наиболее крупные вычислительные комплексы построены именно в облачной среде и включают от десятков до сотен тысяч VM (virtual machines) [26]. Однако пропорциональное наращивание обслуживающего персонала делает деятельность компании неэффективной ввиду того, что ручная инсталляция ПО на сотнях тысяч VM зачастую приводит к ошибкам работы компьютерного сервера по причине человеческого фактора [27].

Все это привело к созданию программных средств автоматической установки ПО, которые обеспечивают единообразный алгоритм обслуживания облачной среды на множестве серверов. Наиболее полный сравнительный анализ таких средств автоматизации дан в работе Placette [28]. На основе полученных результатов авторы пришли к выводу о невозможности использования единого

универсального средства для установки и обновления множества ПО в облачной среде, включая автоматическое создание VM и обновление операционных систем, особенно семейства MS Windows [29], ввиду несовместимости процедур установки ПО разных производителей.

Для повышения эффективности обслуживания и обновления ПО в облачной инфраструктуре реализована новая интегрированная система автоматического развертывания ПО, позволяющая эффективно управлять и централизованно обслуживать множество разнородных облачных ресурсов и процессов.

За основу разработки взяты текущие информационные процессы и средства обслуживания типового ПО в компании RingCentral (США) в различных географических зонах земного шара [10]. Обслуживание распределенной облачной инфраструктуры RingCentral базируется на следующих программных системах (рис. 3.4):

1. Инвентаризация сетевой инфраструктуры (Puppet, Forman, RT)
2. Развертывание VM и установка ПО (ADS)
3. Контроль и управление задачами обслуживающего персонала (Jira, CMP, Wiki)
4. Мониторинг технического состояния облачных сервисов (Zabbix, Jenkins)

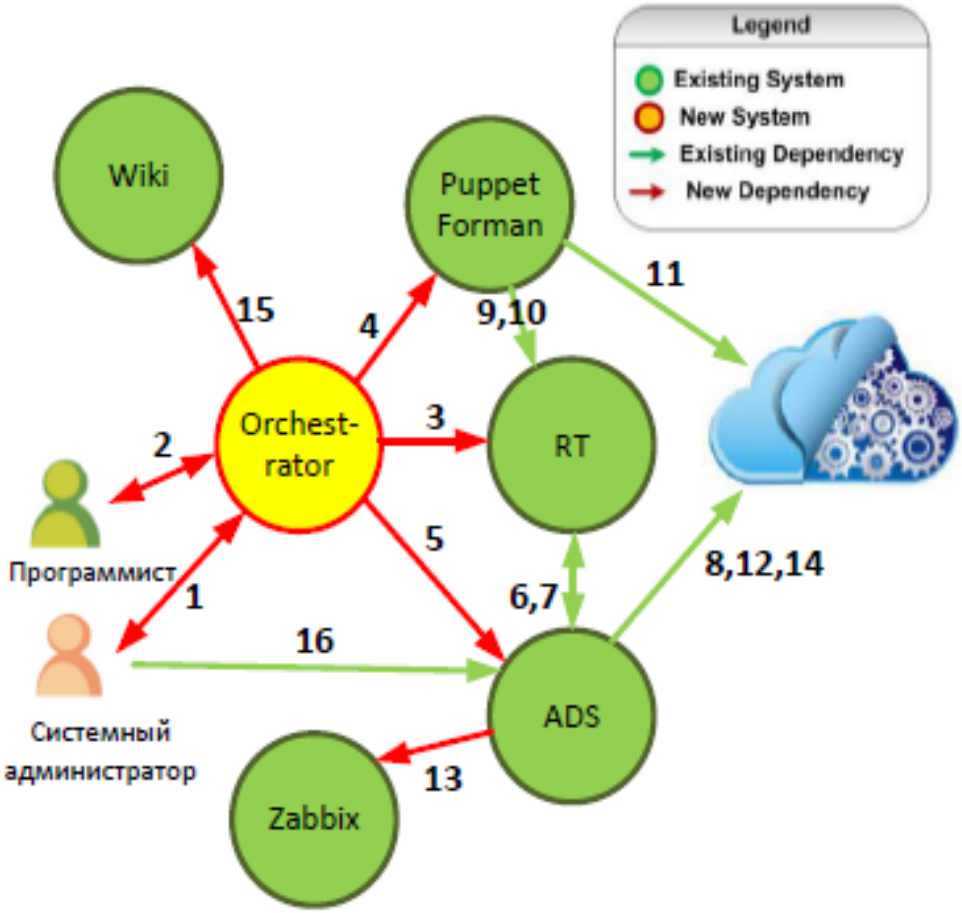

Рисунок 3.4: информационные системы и процесс обслуживания ГРИС

На первом этапе исследований разработана алгоритмическая модель, описывающая процесс инсталляции ПО в полуавтоматическом режиме. В результате получены аналитические данные, включая время инсталляции и деинсталляции ПО, обновления операционных систем и создания новых сред на основе существующих шаблонов. В ходе последующего анализа дана оценка эффективности обслуживания как количество обновлений в единицу времени, ограниченном регламентированным периодом сервисных мероприятий в облачной среде.

На втором этапе исследований изучена эффективность процессов инсталляции ПО при внедрении новой интегрированной системы, управляющей всеми остальными компонентами, не взаимодействующими между собой.

Новый алгоритм работы состоит из следующих действий (рис. 3.4):

1. Системный администратор вводит данные о VM

2. Загрузка данных о среде в виде описательного программного манифеста
3. В случае построения нового сервера создается запись о нем в БД инвентаризации (RackTables, RT)
4. Резервирование ресурсов к запуску новых идентичных VM через Puppet/Forman
5. Подача команды в систему автоматического развертывания (ADS)
6. ADS загружает данные о вновь создаваемых VM из RT
7. ADS получает необходимые ключи доступа к облачной среде
8. ADS запускает клонирование VM из сохраненного в БД образа
9. Puppet загружает манифест из RT
10. Puppet получает необходимые ключи доступа к облачной среде
11. На вновь клонированный сервер Puppet устанавливает заданный манифест
12. ADS устанавливает заданное ПО
13. ADS посылает параметры готового сервера в систему Zabbix для начала мониторинга
14. Настройка внешних сервисов облачной среды: Domain Name Service (DNS), Session Border Controller (SBC), Load Balancer (LBR)
15. Передача и создание веб страницы в базе данных документации
16. Возможность проведения аудита ADS

Интегрированная система отображает результаты работы в режиме реального времени на централизованном вэб интерфейсе. Пример экранной формы представлен на рис. 3.5.

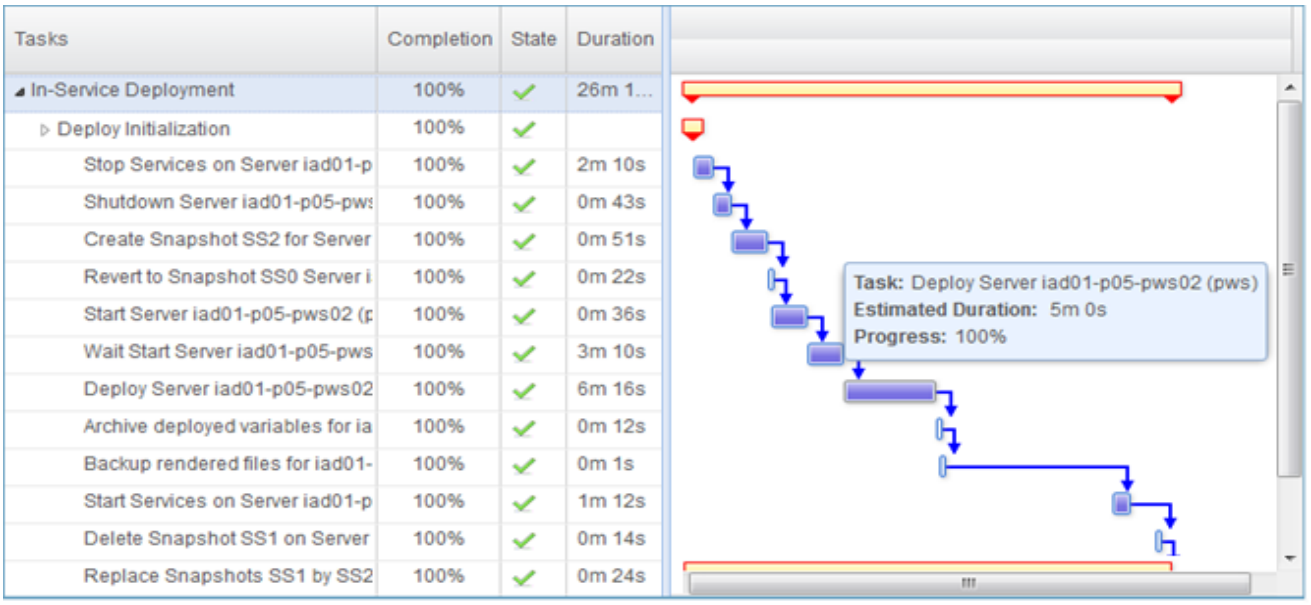

Рисунок 3.5: интегрированный интерфейс работы системы управления в режиме реального времени

В результате работы создан интегрированный интерфейс, позволяющий в режиме реального времени визуально контролировать работу системы централизованного управления программными средствами автоматической установки ПО в облачной инфраструктуре.

3.3. Интеграция облачных сервисов с использованием БД управления конфигурацией Configuration Management DataBase

Современные глобально распределенные облачные сервисы, такие как социальные сети, поисковики, телефония и видеосвязь, обслуживают пользователей по всему миру. Серверы, обрабатывающие запросы пользователей, располагаются в ЦОД на разных континентах. Как правило, к облачным сервисам предъявляются высокие требования по доступности, например общепринятым для телефонии является требование по доступности 99,999%. Это достигается за счет высокой степени функциональной декомпозиции и резервирования, однако делает систему сложной [30]. Другим важным требованием является незамедлительное уведомление клиентов о сбоях в работе сервиса. При этом проинформированы

должны быть только те пользователи, которые действительно пострадали, иначе, в случае если все пользователи будут знать о всех сбоях, у них может сложиться ложное негативное представление о надежности сервиса. Выборочная незамедлительная нотификация в сочетании со сложностью системы, - трудоемкая задача, которая требует автоматизации и визуализации проекта автоматизированной системы.

Поэтому была поставлена задача об уведомлении пострадавших пользователей с целью выявить способ автоматизации. А также визуализировать схему их взаимодействия, которая может быть применена различными компаниями вне зависимости от специфики их деятельности.

Для этого были проанализированы общепринятые практики и процессы эксплуатации информационного сервиса ITIL [6], среди которых выявлены следующие:

Каталог сервисов. Представляет из себя исчерпывающий перечень сервисов компании: как внешних (пользовательских), так и внутренних (технических), а также их взаимосвязи. Позволяет определить, какой из пользовательских сервисов перестает обрабатывать запросы в случае сбоя в одном из технических сервисов.

Конфигурационная база данных (Configuration Management DataBase, CMDB). Содержит информацию о том, какие серверы отвечают за какой сервис, а также о том, какие серверы обслуживают каких пользователей.

Управление инцидентами. Процесс, целью которого является минимизация ущерба пользователям в результате сбоя системы, а также их своевременное информирование.

Мониторинг и управление событиями. Инструментарий и процесс сбора метрик о состоянии серверов с целью максимально раннего обнаружения сбоя в работе системы.

Также был изучен опыт автоматизации эксплуатации компанией RingCentral (США), которая предоставляет сервис голосовой и видеосвязи, обмена мгновенными сообщениями, обмена файлами и др. в Северной Америке, Европе и

Азиатско-Тихоокеанском регионе. В результате была получена следующая схема взаимодействия (рис. 3.6).

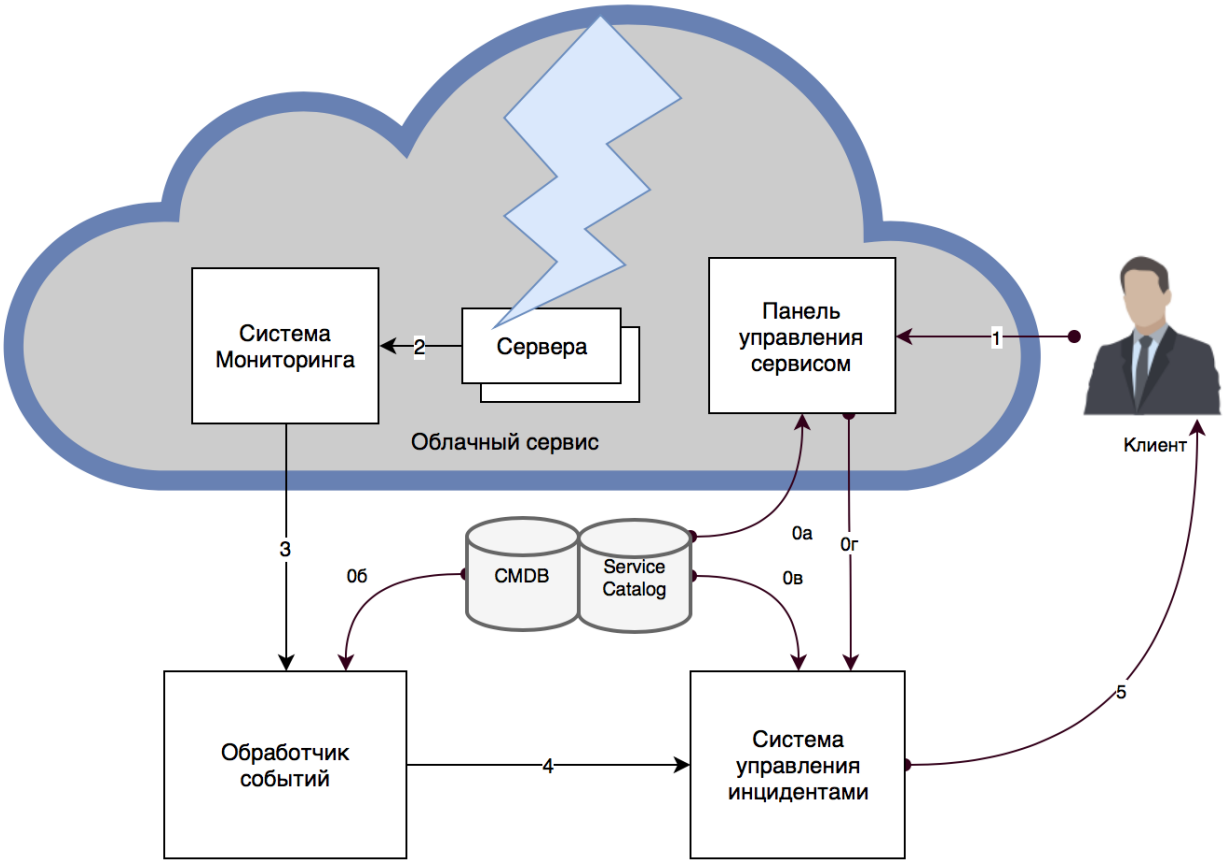

Рисунок 3.6: схема взаимодействия средств автоматизации эксплуатации с конфигурационной базой и каталогом сервисов для автоматической нотификации клиента.

1. Клиент подписывается на уведомления о недоступности сервиса. Панель управления содержит список сервисов из каталога (стрелка 0а).

2. В систему мониторинга приходит мониторинговое событие, связанное со сбоем на одном или нескольких серверах [31].

3. Мониторинговая система фиксирует его и передает в обработчик событий.

4. Обработчик событий, на основании информации о том, какие серверы обслуживают какой сервис, полученной из CMDB (стрелка 0б), делает заключение о том, какой технический сервис сбоит и передает эту информацию в систему управления инцидентами.

5. Система управления инцидентами, обладая информацией о подписках клиентов (стрелка 0г) и о том, какой пользовательский сервис зависит от сбоящего технического сервера, определяет список клиентов, которых нужно нотифицировать и выполняет нотификацию.

На основании предложенной схемы взаимодействия системы мониторинга, обработчика событий, системы управления инцидентами, панелью управления сервисом, CMDB и каталогом сервисов удалось реализовать автоматическое уведомление клиентов о сбоях в работе сервисов, которыми они пользуются.

3.4. Практическая реализация программных приложений интеграции в компании RingCentral

Разработанная информационная модель интеграции процессов обслуживания облачных сервисов реализована на практике и внедрена в глобально распределенной инфраструктуре компании RingCentral, где показала свою эффективность в обеспечении непрерывного цикла обслуживания и обновления облачных сервисов в условиях их быстрого развития и постоянной модернизации.

На рис. 3.7 приведена архитектура интегрированной системы обслуживания облачных сервисов компании RingCentral, где реализованы подходы:

1. Использование единого механизма авторизации (single sign on).

2. Микро-сервисная архитектура виртуальных серверов.

3. Использование шины передачи сообщений по принципу оповещения клиентов для обмена данными между микро-сервисами.

4. Доступ к данным через абстракцию API.

Для новой архитектуры разработан интегрированный интерфейс (рис. 3.8) с целью визуализации всех текущих событий в системе, их корреляции между собой, с последними изменениями и известными хроническими проблемами.

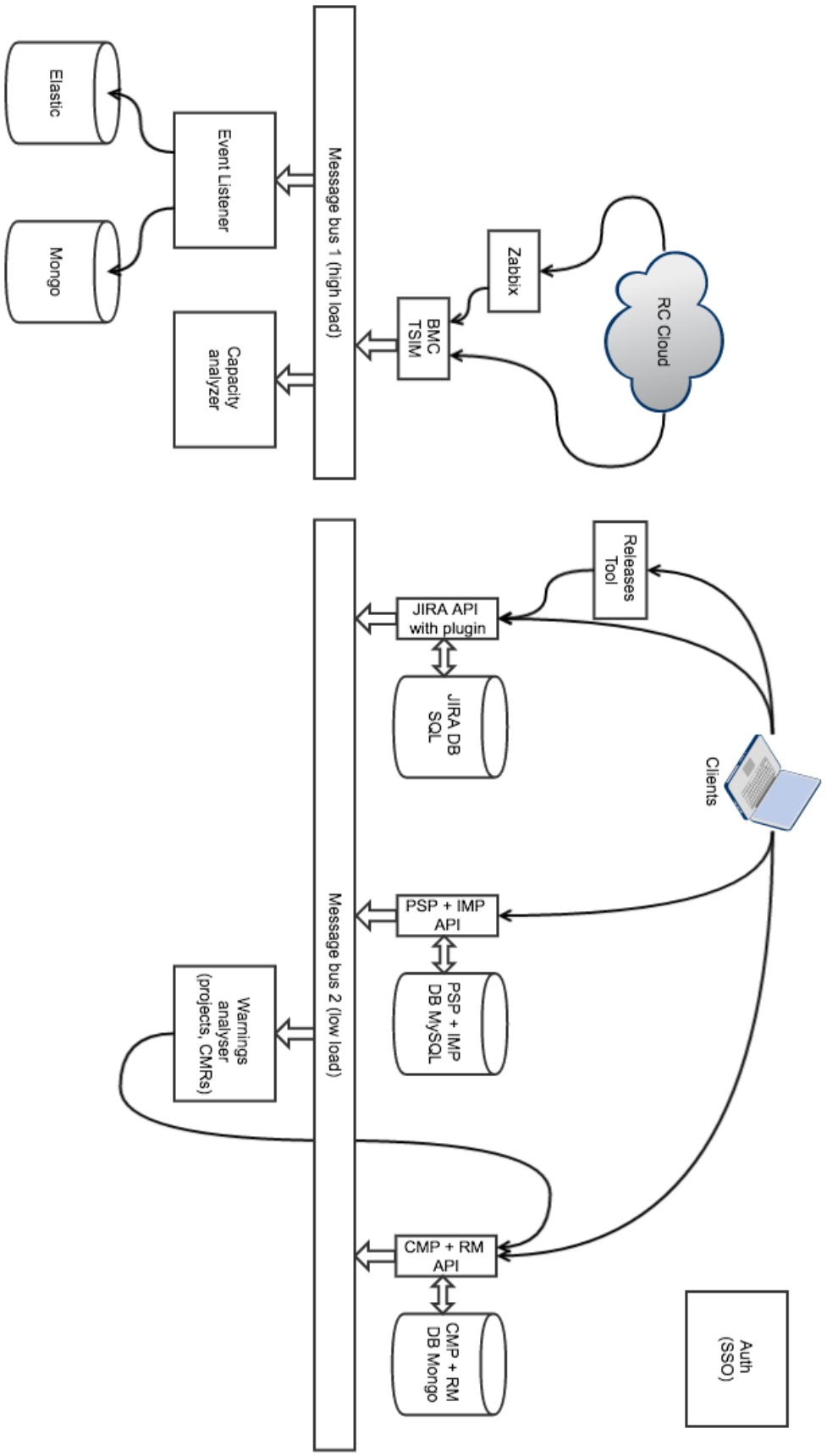

Рисунок 3.7: архитектура интегрированной системы обслуживания облачных сервисов компании RingCentral

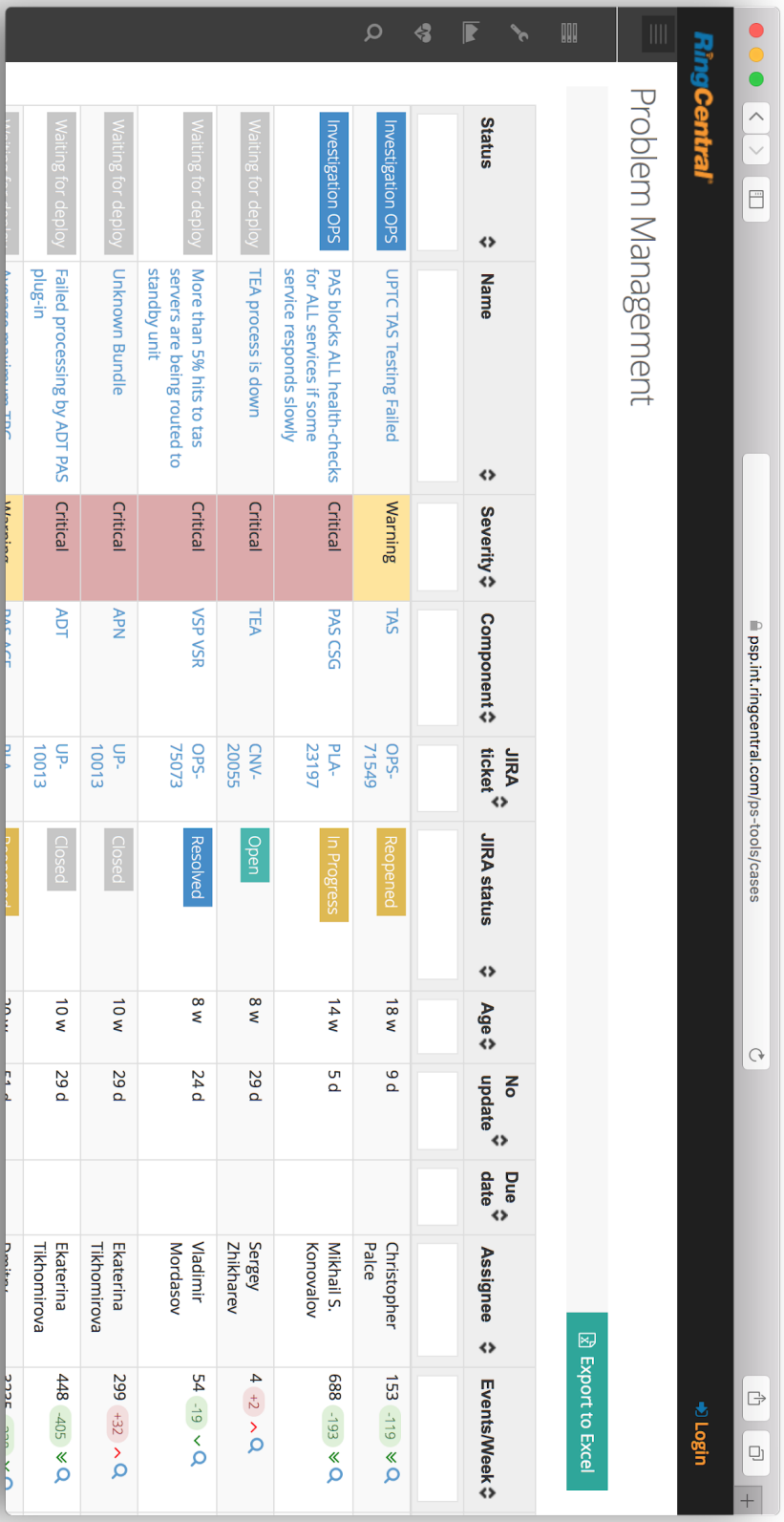

Рисунок 3.8: интегрированный интерфейс контроля за проблемами, показывающий название проблемы (Name), соответствующую заявку на исправление ошибки

Разработана универсальная программная библиотека API функций на языках JavaScript и HTML (рис. 3.9), которая интегрируется на уровне веб приложения с любой информационной системой в архитектуре клиент-сервер и реализует возможность идентификации проектов и ресурсов посредством стандартного поля ввода.

```
26  initSelection: function(element, callback) {
27      var id=AJS.$(element).val();
28      if (id!=="") {
29          AJS.$.ajax("https://roadmap.ringcentral.com/projects/" + id + ".json", {
30            type: 'get',
31              data: {
32                  ids: id
33              },
34              dataType: "jsonp",
35              error: function (xhr, status, error) {
36          notify("Error: " + error + ".", false);
37          }
38          }).done(function(data) { callback(data); });
39      }
40  },
```

Рисунок 3.9: фрагмент программного кода библиотеки на языке JavaScript

Визуализация интегрированной информации о проектах и ресурсах реализована в специально разработанном веб приложении посредством проектных и ресурсных срезов данных (рис. 3.10).

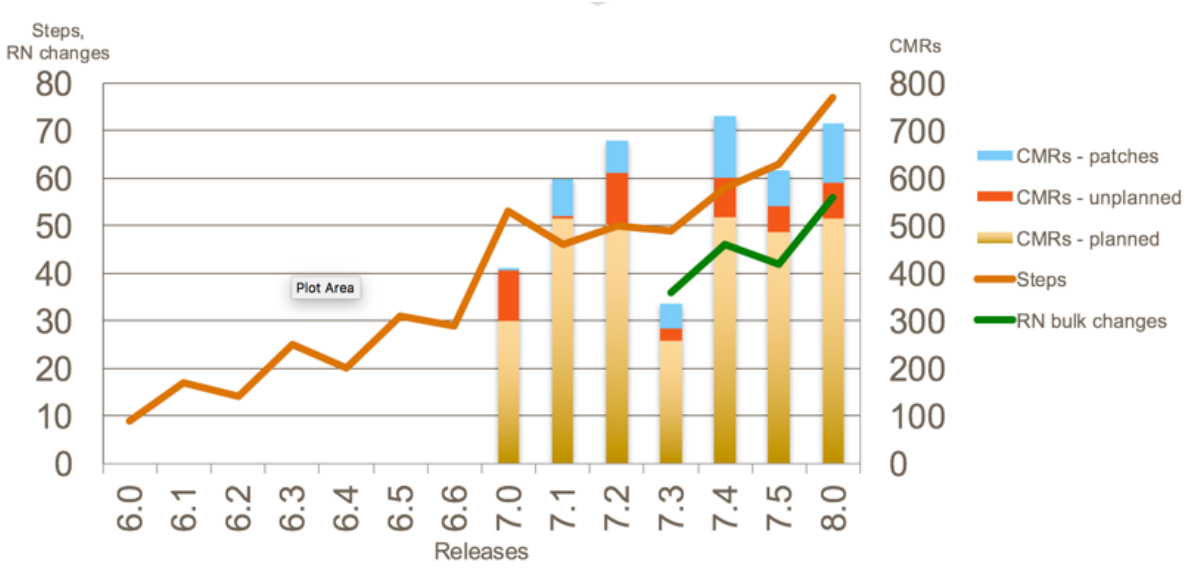

Рисунок 3.10: анализ интегрированной информации о проектах

3.5. Выводы по главе 3

Внедрение интегрированных автоматических средств обслуживания облачной инфраструктуры, включая единый централизованный интерфейс визуализации и мониторинга, позволило радикально поднять производительность автоматической установки ПО и тем самым значительно повысить эффективность регламентных работ по обслуживанию облачной инфраструктуры.

Визуализированная автоматическая система позволяет незамедлительно, без задержки на обработку информации и принятие решения человеком, уведомить клиентов. Уведомления точечно уходят только тем клиентам, которые используют сервис, испытывающий сбой, и подписались на уведомления. Это позволяет бизнесу выполнять свои обязательства перед клиентами, при этом, не создавая у остальных пользователей ложного негативного впечатления о надежности сервиса.

В работе был проведен анализ опыта внесения изменений в облачный сервис компании RingCentral. Было показано, что внесение изменения состоит из нескольких этапов, а крупные обновления, как правило, состоят из серии изменений, применяемых в определенной последовательности. Авторы работы предложили метод оценки интенсивности внесения изменений, потенциала увеличения скорости обновления.

Был проведен расчет минимальной продолжительности серии изменений фактически развернутого компанией RingCentral Версии ПО 10.1 в 2018 году. Расчет показал, что существует потенциал повышения интенсивности развертывания примерно в 3,5 раза.

Дальнейшего исследования требуют другие ограничения, которые влияют на интенсивность внесения изменения, такие как ограниченность ресурсов (как человеческих, так и электронно-вычислительных), различные технологические ограничение.

Глава 4. Разработка методов и моделей количественной оценки эффективности обслуживания ГРИС на основе объективных данных мониторинга

4.1. Типовая архитектура системы мониторинга ГРИС

Мониторинг - это система постоянного наблюдения за явлениями и процессами, происходящими в системе, результаты которого служат для обоснования управленческих решений [32]. Применимо к ГРИС, система мониторинга служит обоснованием для принятия таких решений как увеличение или уменьшение ресурсов сети в связи с изменением количества запросов пользователей; решений о замене оборудования в связи с выходом его из строя и пр.

Состояние ГРИС характеризуется набором ее параметров, которые называются метриками. По способу сбора метрики делятся на несколько видов:

Исторически мониторинг вычислительных сетей начался с наблюдения над сетевым оборудованием, таким как маршрутизаторы, сетевые экраны, принтеры и пр. с целью обнаружения нештатных событий в работе оборудования. Для удаленного управления сетевым оборудованием используется стандартный протокол SNMP (Simple Network Management Protocol), при этом один из способов взаимодействия управляемого устройства с сервером - это отправка “ловушки” (trap).

Недостатком метода сбора метрик состояния через прием “ловушек” является зависимость от наблюдаемого устройства. Сообщение о нештатной ситуации генерируется на устройстве и в ряде случаев (например, в случае аварийного выключения) устройство не может сгенерировать сообщение. Для минимизации подобных рисков, а также в том случае, если устройство не умеет самостоятельно генерировать сообщение, на устройство устанавливается дополнительное приложение - агент системы мониторинга. Задачей агента является наблюдение за определенными параметрами устройства и отправка с

определенной периодичностью значений этих параметров в систему мониторинга. При данном методе решение о том, произошло ли нештатное событие принимает не наблюдаемое устройство, а система мониторинга, анализируя поток поступающих значений метрик. О сбое может сигнализировать определенное значение метрики (например, запущено ли на сервере определенное приложение или нет), превышение порогового значения метрики (например, высокий уровень использования оперативной памяти), а также отсутствие новых значений метрик, что может указывать на полный выход из строя наблюдаемого оборудования.

Еще одним методом сбора метрик является их запрос системой мониторинга. Это удобно в случае, когда установка агента системы мониторинга на сетевое устройство невозможна, а также упрощает обнаружение его полного выхода из строя. При данном методе система мониторинга сама отправляет запрос на устройство и ожидает ответ в определенном формате. Это может быть простая проверка на доступность устройства по сети и ответом в таком случае может быть простое “ОК”. Также ответом может быть список текущих значений метрик наблюдаемого устройства.

В данной работе использована мониторинговая система Zabbix - это бесплатный продукт, который разработан для сбора метрик, отправляемых zabbix-агентами, установленными на удаленных серверах (Рис. 4.1).

В случае, когда количество наблюдаемых серверов становится достаточно большим, возникает риск того, что система мониторинга не сможет принять и обработать такое большое количество сообщений из-за ограниченных вычислительных ресурсов. В результате, архитектура системы Zabbix была усложнена и добавлены Zabbix-прокси серверы, каждый из которых принимает часть сообщений. Далее zabbix-прокси передает пакет из объединенных данных на центральный сервер, на котором данные сохраняются и в дальнейшем анализируются для обнаружения сбоев.

Рисунок 4.1: архитектура системы мониторинга

Сообщения, отправляемые Zabbix-агентами сопровождаются информацией о том, с какого сервера было отправлено сообщение (host), меткой времени сбора метрики (clock), а также ключом, по которому можно выстроить последовательность значений метрики во времени (key) (рисунок 4.2).


```
{
    "host":"<hostname>",
    "key":"log[/home/zabbix/logs/zabbix_agentd.log]",
    "lastlogsize":112,
    "value":" 19845:20140621:141708.521 Starting Zabbix Agent [<hostname>]. Zabbix 2.4.0 (revision 50000).",
    "id": 2,
    "clock":1400675595,
    "ns":77053975
},
{
    "host":"<hostname>",
    "key":"vfs.fs.size[/nono]",
    "state":1,
    "value":"Cannot obtain filesystem information: [2] No such file or directory",
    "id": 3,
```


Рисунок 4.2: сообщение агента системы мониторинга, содержащее указание на сервер (host), с которого отправлена информация

Для каждой метрики в системе мониторинга задается пороговое значение. В случае, если оно превышается, такое сообщение считается сигналом о возможном сбое в работе ГРИС и оно выводится информационное табло с целью информирования оператора службы обслуживания (рисунок 4.3).

Time	Severity	Recovery time	Status	Info	Host	Problem	Duration	Ack
06:01:04	Information		PROBLEM		sjc01-c01-HDV	Vertica: to many monitoring (mon_zabbix) sessions	2m 17s	No
06:00:20	Critical		PROBLEM		sjc01-c01-dos01	CountOfJennePostNewOrderFailedRequests grows	3m 1s	No
06:00:05	Critical		PROBLEM		sjc01-p01-aws02	HTTP ping to 8088 port lost	3m 16s	No
06:00								
05:59:20	Critical		PROBLEM		sjc01-p01-bil02	There is no data about BIL health during 10 minutes or more	4m 1s	No
05:58:59	Warning		PROBLEM		sjc01-p01-fax04	Zabbix Agent is not available	4m 22s	No
05:58:56	Warning		PROBLEM		sjc01-p01-fax05	Zabbix Agent is not available	4m 25s	No
05:58:54	Warning		PROBLEM		sjc01-p01-fax06	Zabbix Agent is not available	4m 27s	No
05:58:19	Warning		PROBLEM		sjc01-c01-scr02	Remote out	5m 2s	No
05:57:09	Information		PROBLEM		sjc01-c01-crs03	Routing policy changed	6m 12s	No

Рисунок 4.3: информационное табло системы мониторинга Zabbix

4.2. Показатели оценки эффективности технического обслуживания облачных сервисов

Достижение качества предоставления услуг, определенного в соглашении о качестве услуг (SLA), требует наличия информационной системы, контроля качества при разработке ее обновлений, а также бесперебойной эксплуатации и обслуживания.

В крупной компании эти работы выполняются разными группами сотрудников. Для оценки эффективности обслуживания ИС существует ряд показателей, ключевым из которых является среднее время, требуемое для восстановления штатной работы ИС после сбоя (Mean Time to Restore, MTTR).

Время, требуемое для восстановления штатной работы ИС после - это время, которое прошло с момента возникновения сбоя до момента, когда подтверждено штатное состояние ИС. Какое-то время требуется на обнаружение сбоя. Мониторинговая система собирает информацию о состоянии ИС с задержкой и между событием отказа в обслуживании пользователей и появлением соответствующего сообщения в мониторинговой системе проходит время - это время называется временем обнаружения (Mean Time to Detect, MTTD) и является

ключевым показателем работы мониторинговой системы. В случае, когда мониторинговая система собирает недостаточное количество метрик состояния ИС, либо неправильно анализирует значения этих метрик, сбой может не быть обнаружен автоматически и служба эксплуатации будет оповещена о сбое через жалобы пользователей в службу поддержки, что негативно скажется на MTTR.

Крупная глобально распределенная информационная система состоит из различных серверов запущенных в множественном количестве экземпляров (необходимое количество экземпляров определяется их производительностью и количеством запросов пользователей к ним, а также необходимостью в резервировании для обеспечения бесперебойной работы). В результате, инженеры службы эксплуатации, занимающиеся обслуживанием и восстановлением системы, вынуждены специализироваться на ограниченном наборе компонентов системы. При возникновении сбоя, требуется проинформировать и привлечь к восстановлению сервиса именно того инженера, который специализируется на вышедшем из строя компоненте. Также необходимо учесть рабочий график инженеров - для обслуживания информационной системы в режиме 24/7 инженеры формируют график дежурств. Получив сообщение о сбое, инженер службы эксплуатации должен установить безопасное соединение со своего рабочего места (если он находится в офисе компании), либо из того места, где он сейчас находится (если он, например, работает из дома), выйти на связь и быть готовым начать работы по восстановлению сервиса. В результате, следующим этапом восстановления сервиса, требующим определенного времени, является время, требуемое на коммуникацию (Mean Time to Communicate, MTTC). В случае, если инженер службы эксплуатации недоступен, либо не имеет возможности приступить к восстановлению системы (например, из-за проблем с выходом в сеть Интернет при работе из дома), MTTC может занять значительное время и сильно повлиять на MTTR.

Далее инженер либо группа инженеров службы эксплуатации приступает к восстановлению работоспособности сервиса. В случае, когда шаги, требуемые для восстановления работоспособности сервиса неизвестны, этот этап может занять

значительное время, на протяжении которого пользователи будут испытывать отказ в обслуживании. Поэтому в IT компаниях заранее разрабатываются и проходят пробные испытания сценарии восстановления работоспособности ИС. Основным сценарием является переключение на резервные мощности. Другими заранее заготовленными сценариями являются перезагрузка серверов и добавление дополнительных серверов, а также блокирование избыточных запросов пользователей.

Список показателей эффективности эксплуатации, а также их взаимосвязь показаны на рисунке 4.4.

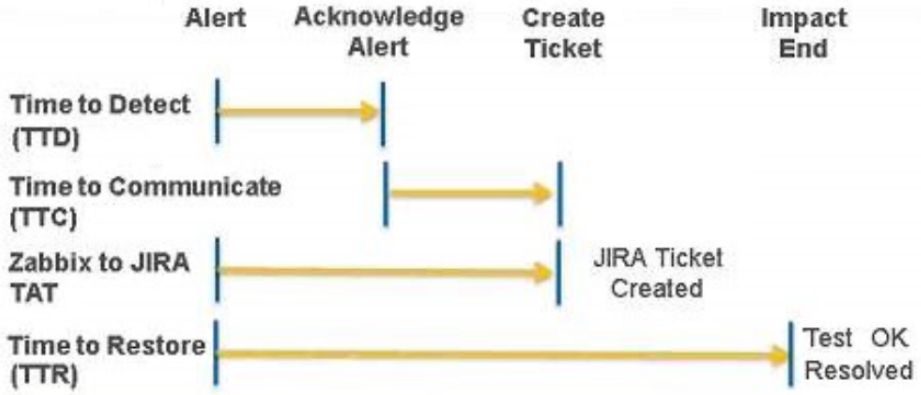

Рисунок 4.4: показатели эффективности работы эксплуатации

4.3. Практическая реализация методов и моделей количественной оценки эффективности обслуживания в компании RingCentral

Телекоммуникационная компания RingCentral (RC) предоставляет до 150 облачных сервисов по всему миру [10]. Учитывая огромный парк из 10 тыс виртуальных машин (Virtual Machines, VMs), а также устойчивый годовой рост, возникла объективная потребность в сравнительной оценке качества сервиса (Quality of Service, QoS) на основе общепринятых во всем мире ключевых показателей производительности (Key Performance Indicators, KPIs).

Далее будут описаны метрики KPI, которые вычисляются автоматически на базе модели интеграции различных источников данных, предложенной в работе [33], и реализованы в единой информационно-управляющей системе масштаба крупной компании RingCentral.

Если раньше Интернет провайдеры обеспечивали пользователям каналы связи и передачу по ним потоков данных, то в настоящее время все больше телекоммуникационных компаний предоставляют доступ к внутренним ресурсам (IaaS) и/или информационным сервисам (SaaS), которые реализуются по облачным технологиям виртуализации.

В компании RC функционирование всей глобальной облачной инфраструктуры отслеживается специальным оперативным центром (Global Network Operating Center, GNOC), который работает 24 часа в сутки 7 дней в неделю и несет ответственность за бесперебойность облачных сервисов в соответствии с принятым соглашением об уровне обслуживания (SLA) более 300 тыс корпоративных клиентов. Все данные в реальном времени и критические события на многочисленных удаленных серверах отслеживаются мониторинговой системой Zabbix в единой базе данных и автоматически отображаются вэб приложением на централизованном табло (Network Monitoring Console, NMC) с целью быстрого обнаружения и реагирования на возможные аномалии. Анализ оперативных и исторических данных в БД Zabbix позволяет оценить динамику, определить повторяющиеся аномалии, предсказать и заранее предотвратить многие проблемы не допуская полного отказа распределенных сервисов в обслуживании (Distributed Denial of Service, DDoS).

В таблице 4.1 приведены источники информации и методы отслеживания критических событий в RC сервисах. В случае обнаружения аномалии действия GNOC, как и в любой другой компании в сфере предоставления SaaS/IaaS, включают следующие главные этапы:

Таблица 4.1: Методы обнаружения аномалий в компании RingCentral.

Источники и способы обнаружения аномалий	Средняя статистика RC

Автоматические оповещения в мониторинговой системе Zabbix	90%
Сообщения от внешних партнеров и поставщиков Интернет услуг	4%
Функциональные тесты, выполняемые вручную с заданной периодичностью	3%
Уведомления от службы поддержки корпоративных клиентов	2%
Прочие источники	1%

1. Обнаружение и подтверждение аномалии посредством NMC либо одним из способов, указанных в таблице 4.1.

2. Уведомление других членов команды и протоколирование сообщений об ошибке в одной из систем документирования. В компании RC для этих целей используется популярная в мире информационная система JIRA [34], а также инструментальные средства собственной разработки с отдельной базой данных инцидентов Incident Management Portal (IMP).

3. Восстановление SaaS имеющимися автоматизированными средствами GNOC, если проблема хорошо известна и для нее предусмотрено оперативное решение, например перезагрузка VM или перенаправление рабочей нагрузки на запасной сетевой ресурс.

4. Когда SaaS не может быть восстановлен средствами GNOC, инцидент эскалируется инженерам службы технической поддержки. Если в процессе решения проблемы модифицируется программное обеспечение или инфраструктура, то все изменения отражаются в отдельной информационной системе собственной разработки Change Management Portal, связанной с системой планирования проектов Roadmap.

Для ряда хронических проблем, относящихся к внешним поставщикам и не зависящих от внутренних ресурсов RC, предусмотрены автоматические процедуры восстановления SaaS. В таких случаях участие GNOC на этапах 1 и 2

не требуется, и документ JIRA также создается автоматически. Примером такой реализации является автоматическое восстановление Java сервисов в результате общеизвестной проблемы утечки памяти при пиковой пользовательской нагрузке на приложения с целью предотвращения DDoS, подробно рассмотренные в работе [35].

При оценке эффективности процессов в IMP/CMP/JIRA/Zabbix/Roadmap с помощью KPI главная проблема заключается в разрозненности перечисленных источников данных, хранящихся в отдельных БД реляционного типа. Поскольку в БД мониторинговой системы Zabbix сконцентрирована вся информация об удаленных серверах, то было решено использовать встроенные средства Zabbix для интеграции всех БД и автоматизации вычислений KPI. С этой целью разработаны гетерогенные SQL запросы, которые направляются по заданному расписанию с прокси-сервера Zabbix на распределенные серверы БД, и результат их выполнения возвращается в Zabbix как отдельные значения метрик по каждой из систем IMP/CMP/JIRA/Roadmap.

В таблице 4.2 в качестве примера приведены некоторые метрики и их ежедневные значения для системы IMP. Интеграция данных в Zabbix в последствии позволяет создавать новые метрики KPI на основе агрегированных выражений любой степени сложности, которые уже вычисляются локально на стороне сервера Zabbix и сохраняются в его БД. Для интегральной оценки QoS в RC реализованы метрики KPI, описанные в предыдущем разделе:

1. Время обнаружения аномалии (TTD). Традиционно TTD определяется как разница между фактическим событием на удаленном сервере и временем его фиксации в мониторинговой системе [36]. В реальности существует незначительная задержка, вызванная объективной потребностью передачи самых последних данных с удаленного сервера через прокси-сервер в БД мониторинговой системы Zabbix. Поэтому в компании RC принято оценивать TTD как время реакции GNOC на критическое событие в NMC с максимальным SLA 2 минуты.

Таблица 4.2: Пример ежедневного отчета по метрикам IMP в Zabbix.

Наименование метрики IMP в Zabbix	Значение за последний день
% инцидентов, автоматически обнаруженных в Zabbix	84
% инцидентов, эскалированных в службу поддержки	0
% инцидентов по результатам часовых ручных тестов	11
% инцидентов, выявленных корпоративными клиентами	0
% инцидентов, обнаруженных другими источниками	5
Среднее время устранения инцидента, минут	6
Общее количество инцидентов	19
Максимальная продолжительность инцидентов, минут	40
Максимальное число затронутых клиентов	0
Максимальное число оборванных соединений	500

2. Время уведомления об аномалии (TTC). Под TTC понимается именно письменное уведомление заинтересованных лиц [37]. В компании RC для этих целей эффективно используются ИУС IMP и JIRA. Для документирования инцидента в GNOC установлен SLA 2 минуты, но на практике это занимает не более 1 минуты, поскольку многие процессы передачи информации в цепочке Zabbix- JIRA-IMP автоматизированы.

3. Время восстановления SaaS (TTR). TTR – это базовая мера оценки периода недоступности SaaS для пользователей от момента времени, когда об этом стало известно, независимо от способа его восстановления и степени автоматизации [38]. Некоторые провайдеры ошибочно интерпретируют TTR как время реакции на инцидент (time to react), однако это некорректно, поскольку SaaS

недоступен для пользователей до тех пор, пока процедура восстановления не закончена и QoS не протестировано.

На практике часто оперируют усредненными понятиями MTTD, MTTC, MTTR, которые являются частью SLA [39] и в компании RC вычисляются согласно схемы на рисунке 4.4 по формулам:

$$MTTD = \sum (T_2 - T_1) / N \tag{4.1}$$

$$MTTC = \sum (T_3 - T_2) / N \tag{4.2}$$

$$MTTR = \sum (T_4 - T_3) / N , \tag{4.3}$$

где *MTTD* – среднее время обнаружения аномалии; *MTTC* – среднее время уведомления об аномалии; *MTTR* – среднее время восстановления SaaS; T_1 – время фактического события на удаленном сервере; T_2 – момент времени обнаружения аномалии в GNOC; T_3 – время создания документа JIRA/IMP; T_4 – время завершения процедуры восстановления SaaS и перевода инцидента JIRA/IMP в статус “Resolved/Closed”; *N* – количество инцидентов.

4. Время полного цикла изменения/восстановления SaaS (TAT). TAT – это категория мирового класса, используемая для общей оценки эффективности мероприятий по техническому обслуживанию и эксплуатации производственных систем независимо от отрасли экономики [40]. В IT сфере TAT означает общее время обработки заявки на сервисное обслуживание [41]. В компании RC согласно политики CMP в полный цикл TAT включен весь процесс создания, планирования, согласования изменений и их внедрение в работающую систему (рис. 4.5). Применительно к IMP эскалациям TAT определяется как длительность полного цикла обработки инцидента от момента его возникновения в работающей системе до окончательного восстановления SaaS для пользователей.

Таким образом, справедливо одно из следующих выражений в зависимости от процедуры восстановления SaaS:

$$TAT = MTTD + MTTC + MTTR \quad (4.4)$$

$$TAT = MTTD + MTTR \quad (4.5)$$

Выражение (4.5) применимо при наличии полностью автоматических процедур обнаружения аномалии и восстановления SaaS, когда MTTC и MTTR происходят параллельно и ручного вмешательства в процесс IMP эскалации не требуется.

Анализ статистики инцидентов в IMP показывает, что они во многих случаях являются следствием изменений инфраструктуры, поэтому их контроль в CMP с оценкой KPI имеет важное значение для достижения требуемого уровня SLA. Любое изменение всегда выполняется как часть одного из проектов, планируемых в Roadmap, с установленным рабочим циклом (рис. ?), который включает разработку SaaS (Development, DEV), анонсирование новых функциональных возможностей (Release Notes Meeting, RNM), контроль развертывания SaaS (Deployment Tracking, DT) сначала на тестовой и затем на реальной системе (Production, PRO), управление внедрением (Release Management, RM), тестирование и контроль QoS.

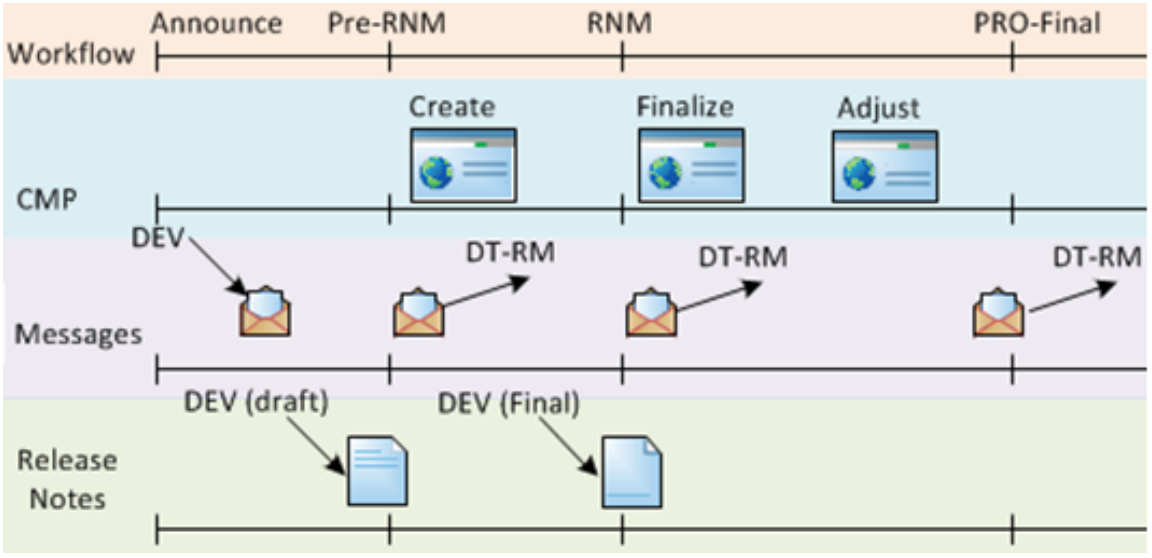

Рис. 4.5: Рабочий цикл выполнения заявок CMP.

Заявка с детальным описанием изменений, включая плановую дату и время реализации, пошаговый план работ, целевые VM, план тестирования и отката в случае ошибки и т. п., подается на рассмотрение через CMP. В RC как крупной компании этот цикл согласования заявки в CMP составляет 8 дней, что позволяет провести техническую экспертизу, проверку плана реализации, анализ возможных последствий, подтверждение на уровне вовлеченных менеджеров отделов и совета директоров (Change Management Board, CMB). Статус заявки, CMB и весь процесс согласования отслеживается в CMP в реальном времени и фиксируется в БД CMP для последующего анализа и объективной оценки эффективности CMP.

С этой целью в мониторинговой системе Zabbix созданы метрики CMP KPI аналогично метрикам IMP, рассмотренным в предыдущем разделе. Пример ежедневного отчета CMP KPI в Zabbix приведен в таблице 4.3, а на графике рис. 4.6 показаны исторические данные за месяц по одной метрике, вычисляющей длительность обслуживания заявок CMP. Данная метрика полезна для анализа загруженности ежедневной плановой профилактики. На графике видно, что 4-часовое окно используется в среднем менее чем на 50%, а выполнение одной заявки превысило максимальный предел 4 ч, но поскольку это произошло в выходной день в часы низкой пользовательской нагрузки, то это не считается инцидентом.

Таблица 4.3: Пример ежедневного отчета по метрикам CMP в Zabbix.

Наименование метрики CMP в Zabbix	Значение за последний день
% изменений, вызванных инцидентами	4
% изменений, выполненных в запланированное время	84
% срочных внеплановых изменений	12
Среднее время выполнения заявки, минут	67

Всего подано заявок	25
Всего отказано в обработке заявок	1
Всего обработано заявок	24

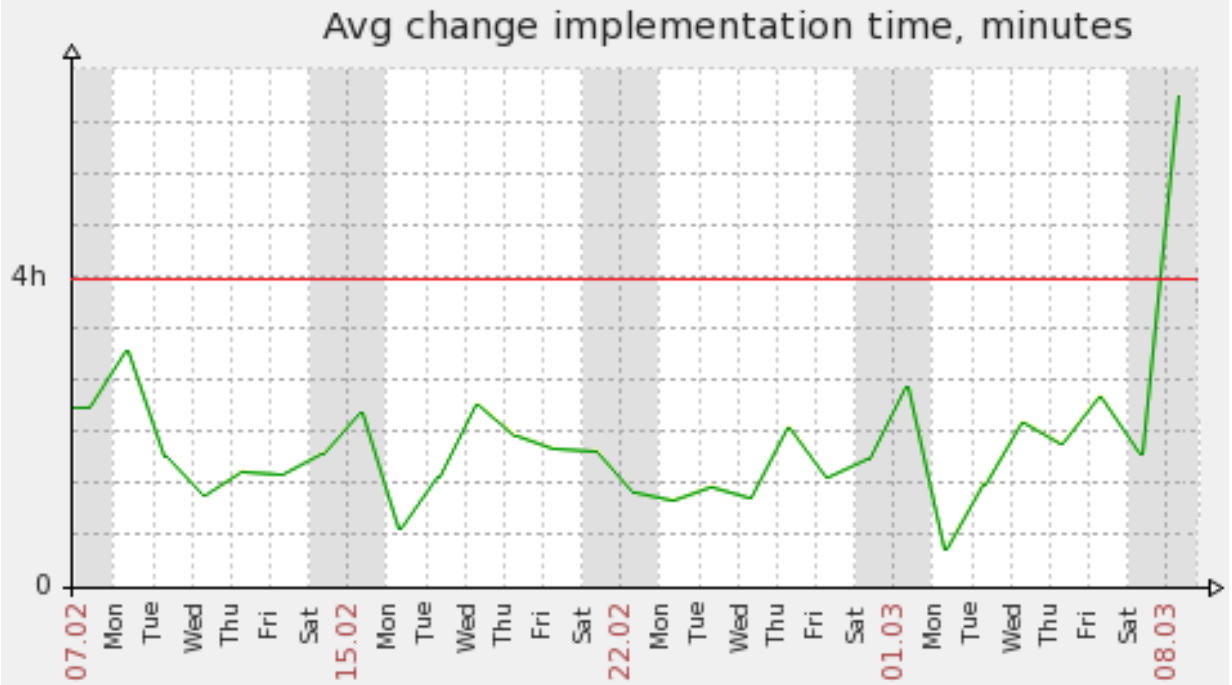

Рисунок 4.6: исторические данные Zabbix о времени выполнения заявок CMP.

В работе [42] представлены другие примеры успешной реализации KPI с целью повышения эффективности IMP/CMP в условиях непрерывности процесса обновления SaaS/IaaS в динамично развивающейся многофункциональной облачной инфраструктуре RC.

4.4. Выводы по главе 4

Главная проблема в повышении эффективности управления большой IT компанией, предоставляющей инфраструктурные и информационные сервисы в глобальной сети Интернет, состоит в разрозненности корпоративных информационных систем различного назначения. С целью интеграции и создания единого информационного пространства в компании RingCentral предложено

использовать мониторинговую систему Zabbix масштаба предприятия, поскольку в ее централизованной базе данных в реальном режиме времени собираются данные о производительности, загруженности ресурсов, пользовательской активности и других системных событиях на многочисленных удаленных серверах и сетевых каналах связи в облачной инфраструктуре. Исторические данные БД Zabbix используются не только при решении текущих задач восстановления сервисов после инцидентов, но и для анализа тенденций дальнейшего развития, прогнозирования темпов масштабирования и потребности в системных ресурсах.

В данной главе предложено использовать преимущества автоматизированных средств мониторинга Zabbix для интегральной количественной оценки эффективности корпоративных систем управления на основе общепринятых в мире ключевых показателей производительности, таких как TTD, TTC, TTR, TAT, а также их усредненных значений. Внедрение KPI и анализ полученных результатов позволили улучшить качество и доступность бизнес сервисов компании RingCentral с 99.995% до уровня 99.998% за счет более эффективного использования 4-часовой профилактики, что является относительно высоким показателем в IT сфере.

Заключение

В ходе выпускной квалификационной работы проведено исследование информационно-управляющих систем, с помощью которых производится целенаправленное изменение состояния ГРИС. Произведен аналитический обзор современных “облачных” технологий, показана типовая архитектура информационных потоков в “облачных” системах. Проанализированы современные методы и средства управлением состоянем ГРИС. Произведен аналитический обзор методов интеграции данных.

На основании проведенного анализа были выявлены ограничения, накладываемые на ИУС, управляющие состоянием ГРИС, была показана необходимость интеграции данных о состоянии ГРИС из разрозненных информационных систем и предложен набор организационно-технических принципов интеграции баз данных этих систем.

В качестве иллюстрации реализации этих принципов была предложена модель интеграции данных ИУС компании RingCentral. Inc, которая следует стандарту ITIL. Были показаны методы и алгоритмы оценки эффективности обслуживания ГРИС на основе данной модели.

Также, на основе предложенной модели, были предложены методы и алгоритмы агрегации данных о состоянии ГРИС с целью получения необходимой информации для принятия решения о целевом состоянии ГРИС.

Были разработаны программы для ЭВМ, реализующие предложенные выше методы интеграции, агрегации и оценки эффективности эксплуатации. Данные программы для ЭВМ используются в компании RingCentral (США).

Внедрение KPI в RingCentral выявило низкую эффективность (50%) использования часов профилактики, а также позволило повысить уровень доступности облачных сервисов до мирового уровня 0,99999.

Список использованной литературы

1. Mell, Peter and Grance, Timothy. The NIST Definition of Cloud Computing. Recommendations of the National Institute of Standards and Technology. http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf

2. Arif Mohamed. A history of cloud computing (англ.). Computer Weekly (27 March 2009).

3. Anthony, James. “70 SaaS Statistics You Must Learn: 2020/2021 Market Share & Data Analysis.” FinancesOnline.com, https://financesonline.com/saas-statistics/.

4. https://www.grandviewresearch.com/industry-analysis/unified-communications-as-a-service-market

5. https://www.uctoday.com/unified-communications/ucaas/gartner-magic-quadrant-for-ucaas-2020/

6. AXELOS. ITIL Foundation: ITIL 4 Edition. London: The Stationery Office, 2019. 212 p.

7. Olups R. Zabbix Network Monitoring. – Packt Publishing Ltd, 2016.

8. Doan A. H., Halevy A., Ives Z. Principles of data integration. – Elsevier, 2012.

9. Bernstein D. Containers and Cloud: From LXC to Docker to Kubernetes. IEEE Cloud Computing, Vol. 1, Issue 3, 2014. – http://ieeexplore.ieee.org/document/7036275/

10. RingCentral Inc. – http://ringcentral.com

11. VMware Cloud Management.

http://blogs.vmware.com/management/tag/change-managemen

12. Volkov A., Efimov V., Mescheryakov S., Shchemelinin D. Integrated Data Model for Managing a Multi-service Dynamic Infrastructure / Proceedings of the 3rd International Conference on Computer Modeling and Simulation.– St. Petersburg, Polytechnic University, 2014. http://dcn.icc.spbstu.ru/index.php?id=344&L=2

13. Ефимов В., Мещеряков С., Щемелинин Д. Интегральная оценка производительности и качества информационных услуг в облачной

инфраструктуре / Распределенные компьютерные и теле-коммуникационные сети (DCCN-2015): Материалы 18 междунар. науч. конф. – М.: ИПУ РАН, 2015. http://www.dccn2015.com/

14. Lanubile F., Ebert C., Prikladnicki R., and Vizcaino A. Collaboration Tools for Global Software Engineering. IEEE Software, Vol. 27, Issue 2, 2010. – http://ieeexplore.ieee.org/abstract/document/5420797/

15. Мещеряков С.В., Иванов В.М. Методы оптимального проектирования баз данных производственного оборудования. – СПб: Изд-во Политехн. ун-та, 2012. – http://gpupress.ru/

16. Volkov A.N., Efimov V.V., Mescheryakov S.V., Shchemelinin D.A. Integrated Data Model for Managing a Multi-Service Dynamic Infrastructure. Computer Modeling and Simulation: труды междунар. науч.-техн. конф. КОМОД-2014, СПб, Изд-во Политехн. ун-та, 2014. – http://dcn.icc.spbstu.ru/index.php?id=344

17. Volume Licensing. Соглашение об уровне обслуживания для Вэб-служб Microsoft [Электронный ресурс] // Microsoft, 12 июля 2018 г. – С. 11-18, Режим доступа:
http://www.microsoftvolumelicensing.com/Downloader.aspx?DocumentId=14043
(Дата обращения: 20.10.2018)

18. V.V. Efimov, S.V. Mescheryakov,D.A. Shchemelinin. Integration data model for continuous service delivery in cloud computing system// Communications in Computer and Information Science, 2017, P. 87-97 Режим доступа: https://www.springer.com/gp/book/9783319668352 (Дата обращения: 20.10.2018)

19. Ефимов В. В., Щемелинин Д. А., Яковлев К. А. Интеграционная модель данных для управления непрерывным обслуживанием глобально распределенных вычислительных систем [Электронный ресурс]// труды междунар. науч.-техн. конф. КОМОД-2017,-СПб, Изд-во Политехн. ун-та, 2017, Режим доступа: http://dcn.icc.spbstu.ru/fileadmin/userfiles/Documents/Erasmus/Sbornik_Comod_2017/COMOD-2017_paper_14.pdf (Дата обращения: 20.10.2018)

20. R. Johnson. More Details on Today’s Outage [Электронный ресурс]// Facebook September 23, 2010, Режим доступа:

https://www.facebook.com/notes/facebook-engineering/more-details-on-todays-outage/431441338919/ (Дата обращения: 20.10.2018)

21. Мещеряков С. В., Щемелинин Д. А. Технология быстрого обновления программных интерфейсов в условиях высокой загруженности веб приложений// Неделя науки СПбПУ, Санкт-Петербург, 01-06 декабря 2014 г., С. 177-179, Режим доступа: https://elibrary.ru/item.asp?id=23713968 (Дата обращения: 25.10.2018)

22. S. Mescheryakov, D. Shchemelinin, V. Efimov. Adaptive Control of Cloud Computing Resources in the Internet Telecommunication Multiservice System // The 6th International Congress on Ultra Modern Telecommunications and Control Systems. St. Petersburg, Russia, 2015-January, P. 287-293

23. D. Yin. This is just in: RingCentral 10.1 Release! [Электронный ресурс] //RingCentral News, May 29, 2018 Режим доступа: https://www.ringcentral.co.uk/blog/ringcentral-10-1-release/ (Дата обращения: 25.10.2018

24. A. Moss, Olprod. История обновлений Office 365 профессиональный плюс (по дате) [Электронный ресурс] // Microsoft, 17 октября 2018 г. Режим доступа: https://docs.microsoft.com/ru-ru/officeupdates/update-history-office365-proplus-by-date (Дата обращения: 25.10.2018)

25. S.V. Mescheryakov, A.O. Rudenko, D.A. Shchemelinin. ASE International Conference on Big Data Science and Computing// Научно-технические ведомости СПбГПУ 1'(212)2015. Информатика. Телекоммуникации. Управление. P. 110-119, Режим доступа: https://cyberleninka.ru/article/v/ase-international-conferences-on-big-data-science-and-computing (Дата обращения: 25.10.2018)

26. Батура Т., Мурзин Ф., Семич Д. Облачные технологии: основные понятия, задачи и тенденции развития.- Программные продукты, системы, выпуск №1, 2014. http://swsys-web.ru/cloud-computing-basic-concepts-problems.html

27. Smith C. Top 5 Big Benefits of Automated Deployment. http://thefutureofdeployment.com/5-big-benefits-automated-deployment/

28. Placette D. 48 Best Cloud Tools for Infrastructure Automation, 2015. https://blog.profitbricks.com/48-best-cloud-tools-for-infrastructure-automation/

29. Shinder T. Automate and Manage Windows Operating System Deployments Now Online, 2015. http://blogs.technet.com/b/privatecloud/archive/2014/10/15/automate-and-manage-windows-operating-system-deployments-now-online.aspx

30. Dragoni N. et al. (2017) Microservices: Yesterday, Today, and Tomorrow. In: Mazzara M., Meyer B. (eds) Present and Ulterior Software Engineering. Springer, Cham

31. Кучерова К.Н., Мещеряков С.В., Щемелинин Д.А. Сравнительный анализ систем мониторинга глобально распределенных вычислительных комплексов // Сравнительный анализ в проектировании и управлении: сб. науч. тр. XX Международной науч.-практич. конф., СПб, СПбПУ, 2016. - С. 303-309.

32. Мониторинг//Гражданская защита: Энциклопедия в 4-х томах. Т. II (К – О) — М.: ФГБУ ВНИИ ГОЧС (ФЦ), 2015).

33. Volkov A., Efimov V., Mescheryakov S., Shchemelinin D. Integrated Data Model for Managing a Multi-service Dynamic Infrastructure. Proceedings of the 3rd International Conference on Computer Modeling and Simulation, St. Petersburg, Polytechnic University Publishing House, 2014, http://dcn.icc.spbstu.ru/index.php?id=344&L=2

34. Atlassian JIRA, https://www.atlassian.com/software/jira

35. Mescheryakov S., Shchemelinin D. Capacity Management of Java-based Business Applications Running on Virtualized Environment. Proceedings of the 39th International Conference for the Performance and Capacity by CMG. La Jolla, CA, USA, 2013, http://www.cmg.org/conference/cmg2013/

36. Fault Detection and Isolation. Wikipedia, Free Encyclopedia, http://en.wikipedia.org/wiki/Fault_detection_and_isolation

37. Nonverbal Communication. Wikipedia, Free Encyclopedia, http://en.wikipedia.org/wiki/Nonverbal_communication

38. Mean Time to Repair. Wikipedia, Free Encyclopedia, http://en.wikipedia.org/wiki/Mean_time_to_repair

39. Service Level Agreement. Wikipedia, Free Encyclopedia, http://en.wikipedia.org/wiki/Service-level_agreement

40. Performance Engineering. Wikipedia, Free Encyclopedia, http://en.wikipedia.org/wiki/Performance_engineering

41. Turnaround Time. Performance Management. Wikipedia, Free Encyclopedia, http://en.wikipedia.org/wiki/Turnaround_time

42. Mescheryakov S., Shchemelinin D. Big Software Deployments in a Big Enterprise Environment. Proceedings of the 2nd ASE International Conference on Big Data Science and Computing, Stanford University, CA, USA, 2014, http://www.ase360.org/handle/123456789/135/

Перечень сокращений

ИУС - информационно-управляющая система

ГРИС - глобально-распределенный вычислительный комплекс

Мбит/с - мегабит в секунду

USW - United States West, западная часть Соединенных Штатов

USE - United States East, восточная часть Соединенных Штатов

AMS - Amsterdam, Амстердам

ZRH - Zurich, Цюрих

IMP - Incident Management Portal, информационная система управления изменениями

CMP - Change Management Portal, информационная система управления изменениями

PM - Project Management, управление проектами

ITIL - Information Technology Infrastructure Library, перечень практик эксплуатации информационных систем

CRM - client relationships management, управление взаимоотношениями с клиентами

IaaS - infrastructure as a service, инфраструктура как сервис

PaaS - platform as a service, платформа как сервис

SaaS - software as a service, программный продукт как сервис

НИОКР - научно-исследовательские и опытно-конструкторские работы

БД - база данных

ЦОД - центр обработки данных

POD - Point of Data

BGP - Border Gateway Protocol

SLA - Service Level Agreement

VM - Virtual Machine

CMDB - Configuration Management DataBase

Приложения

Приложение 1. Программные коды SQL/API для реализации модели интеграции и запросов к разнородным таблицам БД

```
const { sendRequest, handleError, arrayToObj } = require("./utils");
const moment = require("moment");
const CONFIG = require("./../config");
const logger = require("simple-node-logger").createSimpleLogger({
  logFilePath: CONFIG.LOG_PATH,
  timestampFormat: CONFIG.LOG_FORMAT,
});
const {
  EndPoints,
  Statuses,
  Relations,
  TimeframesTypes,
  DictTypes,
  MaintenanceTypes,
} = require("./../../../lib/modules/constants");

const PolicyKey = "MAJOR_CMR_POLICY";
let policyUserId;
let commentTypeId;

/**
 * main method to run
 * @returns {Promise<void>}
 */
async function main() {
  logger.info(`Starting ${PolicyKey}`);

  const {
    cmrs,
    concernedReviewsByCmr,
    timeframesByCmr,
    pendingReviewsByCmr,
  } = await fetchBasicData();
  const conflicts = await calculateConflicts(cmrs, timeframesByCmr);
  await applyPolicy(conflicts, concernedReviewsByCmr, pendingReviewsByCmr);

  logger.info(`Finishing ${PolicyKey}`);
}

/**
 * fetches data needed for further work
    * @returns {Promise<{concernedReviewsByCmr: (Object|void), cmrs: (Object|void), pendingReviewsByCmr:
(Object|void), timeframesByCmr: (Object|void)}>}
 */
async function fetchBasicData() {
  const [cmrs, concernedReviewsByCmr] = await Promise.all([
    getMajorCmrs(),
    getConcernedReviews(),
    setPolicyUserId(),
    setCommentTypeId(),
  ]);

  const [timeframesByCmr, pendingReviewsByCmr] = await Promise.all([
    getCmrTimeframes(Object.keys(cmrs)),
```

```
    getPendingReviews(),
  ]);

  return { cmrs, concernedReviewsByCmr, timeframesByCmr, pendingReviewsByCmr };
}

/**
 * GETs MAJOR CMRs
 * @returns {Promise<Object | void>}
 */
function getMajorCmrs() {
  const endPoint = `${EndPoints.maintenances}/?`;
  const filters = [
    `filter[${Relations.status}.key][in]=${Statuses.approved},${Statuses.pendingApproval}`,
    `filter[${Relations.type}.key][eq]=${MaintenanceTypes.major}`,
    `filter[${Relations.timeframes}.start_at][gte]=${moment().format(
      "yyyy-MM-DD"
    )}`,
  ];
  const url = endPoint + filters.join("&");
  return sendRequest({ url })
    .then((data) => arrayToObj(data))
    .catch((error) => handleError(error));
}

/**
 * GETs TIMEFRAMES for CMRs
 * @param {Array} ids
 * @returns {Promise<Object | void>}
 */
function getCmrTimeframes(ids = []) {
  if (!ids || !ids.length) {
    return {};
  }
  const endPoint = `${EndPoints.timeframes}/?`;
  const filters = [
    `filter[${Relations.context}.id][in]=${ids.join(",")}`,
    `filter[${Relations.context}.type][eq]=${EndPoints.maintenances}`,
    `filter[${Relations.type}.key][eq]=${TimeframesTypes.expected}`,
  ];

  const url = endPoint + filters.join("&");
  return sendRequest({ url })
    .then((data) => {
      const tfByCmr = {};
      data.data.forEach((tf) => {
        const cmrId = tf.relationships[Relations.context].data.id;
        tfByCmr[cmrId] = {
          start_at: tf.attributes.start_at,
          end_at: tf.attributes.end_at,
        };
      });
      return tfByCmr;
    })
    .catch((error) => handleError(error));
}

/**
 * GETs REVIEWS with concern from Policy Bot
 * @returns {Promise<Object | void>}
 */
function getConcernedReviews() {
  const endPoint = `${EndPoints.reviews}/?`;
```

```
  const filters = [
    `filter[${Relations.context}.type]=${EndPoints.maintenances}`,
    `filter[${Relations.noVotes}.id][ne]=null`,
    `filter[${Relations.policy}.key][eq]=${PolicyKey}`,
  ];
  const url = endPoint + filters.join("&");
  return sendRequest({ url })
    .then((data) => placeReviewsByContext(data.data))
    .catch((error) => handleError(error));
}

/**
 * GETs REVIEWS with pending votes from Policy Bot
 * @returns {Promise<Object | void>}
 */
function getPendingReviews() {
  const endPoint = `${EndPoints.reviews}/?`;
  const filters = [
    `filter[${Relations.context}.type]=${EndPoints.maintenances}`,
    `filter[${Relations.pendingVotes}.id][in]=${policyUserId}`,
    `filter[${Relations.policy}.key][eq]=${PolicyKey}`,
  ];
  const url = endPoint + filters.join("&");
  return sendRequest({ url })
    .then((data) => placeReviewsByContext(data.data))
    .catch((error) => handleError(error));
}

/**
 * places REVIEW by its context (1-to-1)
 * @param {Array} data
 * @returns {{}}
 */
function placeReviewsByContext(data = []) {
  const returnObj = {};
  data.forEach((item) => {
    returnObj[item.relationships[Relations.context].data.id] = item;
  });
  return returnObj;
}

/**
 * sets user ID of Policy Bot after it GETs one
 * @returns {Promise<void>}
 */
async function setPolicyUserId() {
  const endPoint = `${EndPoints.accounts}/?`;
  const filters = [`filter[login]=${CONFIG.BOT_USER}`];
  const url = endPoint + filters.join("&");
  policyUserId = await sendRequest({ url })
    .then((data) => data.data.length && data.data.pop().id)
    .catch((error) => handleError(error));
}

/**
 * sets user ID of Policy Bot after it GETs one
 * @returns {Promise<void>}
 */
async function setCommentTypeId() {
  const endPoint = `${EndPoints.dictionaries}/?`;
  const filters = [`filter[key]=${DictTypes.comment}`];
  const url = endPoint + filters.join("&");
  commentTypeId = await sendRequest({ url })
```

```
    .then((data) => data.data.length && data.data.pop().id)
    .catch((error) => handleError(error));
}

/**
 * fetches aaffected MAJOR CMR's ID
 * @param {{start_at: string, end_at: string}} time
 * @param {string} envId
 * @returns {Promise<*[]>}
 */
async function fetchAffectedMajorIds(time, envId) {
  const timeframeFiltersArr = generateAffectedTimeframesFilters(time);
  const promisesArr = timeframeFiltersArr.map((group) =>
    getAffectedMajors(envId, group)
  );
  const arrayOfArr = await Promise.all(promisesArr);
  return [...new Set([].concat(...arrayOfArr))];
}

/**
 * GETs CMRs within $time interval with
 * @param {string} envId
 * @param {array} addFilters
 * @returns {Promise<Array | void>}
 */
function getAffectedMajors(envId, addFilters = []) {
  const endPoint = `${EndPoints.maintenances}/?`;
  const filters = [
    `filter[${Relations.status}.key][in]=${Statuses.approved},${Statuses.pendingApproval}`,
    `filter[${Relations.type}.key][eq]=${MaintenanceTypes.major}`,
    `filter[${Relations.environment}.id][eq]=${envId}`,
    ...addFilters,
  ];
  const url = endPoint + filters.join("&");
  return sendRequest({ url })
    .then((data) => data.data.map((item) => item.id))
    .catch((error) => handleError(error));
}

/**
 * generates array of filters for proper timeframes search
 * @param {{start_at: string, end_at: string}} time
 * @returns {string[][]}
 */
function generateAffectedTimeframesFilters(time) {
  const startStr = new Date(time.start_at).toISOString();
  const endStr = new Date(time.end_at).toISOString();
  return [
    [
      `filter[${Relations.timeframes}.start_at][lte]=${startStr}`,
      `filter[${Relations.timeframes}.end_at][gte]=${endStr}`,
    ],
    [
      `filter[${Relations.timeframes}.start_at][gte]=${startStr}`,
      `filter[${Relations.timeframes}.start_at][lt]=${endStr}`,
      `filter[${Relations.timeframes}.end_at][gte]=${startStr}`,
    ],
    [
      `filter[${Relations.timeframes}.start_at][lte]=${endStr}`,
      `filter[${Relations.timeframes}.end_at][lte]=${endStr}`,
      `filter[${Relations.timeframes}.end_at][gt]=${startStr}`,
    ],
  ];
```

```
}

/**
 * calculates Major CMR conflicts
 * @param {Object} cmrs
 * @param {Object} timeframesByCmr
 * @returns {Promise<{}>} {cmrid: array-of-conflicting-cmrIds, ...}
 */
async function calculateConflicts(cmrs, timeframesByCmr) {
  const returnObj = {};
  const cmrKeys = Object.keys(cmrs);

  for (const cmrId of cmrKeys) {
    const tf = timeframesByCmr[cmrId];
    const envId = cmrs[cmrId].relationships[Relations.environment].data.id;
    const affectedIds = await fetchAffectedMajorIds(tf, envId);
    affectedIds &&
      affectedIds.forEach((item) => {
        if (item === cmrId) {
          return;
        }
        returnObj[item] = returnObj[item] || [];
        returnObj[item].push(cmrId);
      });
  }
  return returnObj;
}

/**
 * method for REVIEW voting and commenting
 * @param {Object} conflicts
 * @param {Object} concernedReviewsByCmr
 * @param {Object} pendingReviewsByCmr
 * @returns {Promise<void>}
 */
async function applyPolicy(
  conflicts,
  concernedReviewsByCmr,
  pendingReviewsByCmr
) {
  const cmrsToConcern = [];
  for (const cmrId in conflicts) {
    if (concernedReviewsByCmr[cmrId]) {
      logger.info(`Updating CMR-${cmrId}`);
      const text = generateMajorConcernText(conflicts[cmrId]);

      await updateConcernComment(concernedReviewsByCmr[cmrId].id, text);
      delete concernedReviewsByCmr[cmrId];
      continue;
    }
    if (pendingReviewsByCmr[cmrId]) {
      delete pendingReviewsByCmr[cmrId];
    }
    cmrsToConcern.push(cmrId);
  }
  const reviewsToConcern = await getReviewsToConcern(cmrsToConcern);
  await voteConcern(reviewsToConcern, conflicts);
  await voteNoObjections({ ...concernedReviewsByCmr, ...pendingReviewsByCmr });
}

/**
 * method for concern COMMENTS updating
 * @param {string} reviewId
```

```
 * @param {string} text
 * @returns {Promise<void>}
 */
async function updateConcernComment(reviewId, text = "") {
  const commentIds = await getConcernComments(reviewId);
  for (const id of commentIds) {
    await patchConcernComment(id, text);
  }
}

/**
 * GETs bot's REVIEWS from CMR
 * @param {Array} cmrIds
 * @returns {Promise<Array | void>}
 */
function getReviewsToConcern(cmrIds = []) {
  if (!cmrIds || !cmrIds.length) {
    return;
  }

  const endPoint = `${EndPoints.reviews}/?`;
  const filters = [
    `filter[${Relations.context}.id][in]=${cmrIds.join(",")}`,
    `filter[${Relations.context}.type]=${EndPoints.maintenances}`,
    `filter[${Relations.policy}.key][eq]=${PolicyKey}`,
  ];
  const url = endPoint + filters.join("&");

  return sendRequest({ url })
    .then((data) => data.data)
    .catch((error) => handleError(error));
}

/**
 * method for concerning CMRs and COMMENTing on them
 * @param {Array} reviews
 * @param {Object} dataObj
 * @returns {Promise<void>}
 */
async function voteConcern(reviews = [], dataObj = {}) {
  for (const review of reviews) {
    const reviewId = review.id;
    const cmrId = review.relationships[Relations.context].data.id;
    const text = generateMajorConcernText(dataObj[cmrId]);
    try {
      logger.info(`Concerning CMR-${cmrId}`);
      await postConcern(reviewId);
      await postConcernComment(reviewId, text);
    } catch (e) {
      handleError(e);
    }
  }
}

/**
 * POSTs user to 'no_votes' relation of the $reviewId
 * @param {string} reviewId
 * @returns {Promise<T | void>}
 */
function postConcern(reviewId) {
  const method = "POST";
  const data = { data: [{ type: EndPoints.accounts, id: policyUserId }] };
  const url = `${EndPoints.reviews}/${reviewId}/relationships/${Relations.noVotes}`;
```

```
  return sendRequest({ url, data, method })
    .then((data) => data.data)
    .catch((error) => handleError(error));
}

/**
 * POSTs COMMENT with concern description
 * @param {string} reviewId
 * @param {string} text
 * @returns {Promise<T | void>}
 */
function postConcernComment(reviewId, text = "") {
  const method = "POST";
  const options = {
    method,
    data: generateCommentBody(text, reviewId, method),
    url: `${EndPoints.comments}/`,
  };
  return sendRequest(options)
    .then((data) => data.data)
    .catch((error) => handleError(error));
}

/**
 * method for COMMENT body generation
 * @param {array} cmrIds
 * @returns {Object} JSON-like object
 */
function generateMajorConcernText(cmrIds) {
  const cmrStr = cmrIds
    .map((id) => `[${id}](${CONFIG.CMP_HOST}/${EndPoints.maintenances}/${id})`)
    .join(", ");
  return `Major CMR is planned on this date: ${cmrStr}.`;
}

/**
 * method for COMMENT body generation
 * @param {Object} text
 * @param {string} reviewId
 * @param {string} method
 * @returns {Object} JSON-like object
 */
function generateCommentBody(text, reviewId = "", method = "") {
  const comment = {
    data: {
      type: EndPoints.comments,
      attributes: {
        text,
      },
    },
  };

  if (method === "POST") {
    comment.data.relationships = {
      context: {
        data: {
          type: EndPoints.reviews,
          id: reviewId,
        },
      },
      type: {
        data: {
          type: EndPoints.dictionaries,
```

```
        id: commentTypeId,
      },
    },
  };
 }

 return comment;
}

/**
 * GETs not started PA/A CMRs
 * @param {Array} ids
 * @returns {Promise<Object | void>}
 */
function getActualCmrs(ids = []) {
 const endPoint = `${EndPoints.maintenances}/?`;
 const filters = [
  `filter[${Relations.status}.key][in]=${Statuses.approved},${Statuses.pendingApproval}`,
  `filter[${Relations.timeframes}.start_at][gte]=${moment().format(
   "yyyy-MM-DD"
  )}`,
  `filter[id][in]=${ids.join(",")}`,
 ];
 const url = endPoint + filters.join("&");
 return sendRequest({ url })
  .then((data) => arrayToObj(data))
  .catch((error) => handleError(error));
}

/**
 * method for removing concern and description COMMENTS
 * @param {Object} reviews
 * @returns {Promise<void>}
 */
async function voteNoObjections(reviews = {}) {
 const cmrIds = Object.keys(reviews);
 if (!reviews || !cmrIds.length) {
  return;
 }
 const actualCmrs = await getActualCmrs(cmrIds);
 for (const cmrId in reviews) {
  if (!actualCmrs[cmrId]) {
   continue;
  }
  try {
   logger.info(`Removing concern from CMR-${cmrId}`);
   await postNoObjections(reviews[cmrId].id);
   await removeConcernComments(reviews[cmrId].id);
  } catch (e) {
   handleError(e);
  }
 }
}

/**
 * POSTs user to 'yes_votes' relation of the $reviewId
 * @param {string} reviewId
 * @returns {Promise<Array | void>}
 */
function postNoObjections(reviewId) {
 const method = "POST";
 const data = { data: [{ type: EndPoints.accounts, id: policyUserId }] };
 const url = `${EndPoints.reviews}/${reviewId}/relationships/${Relations.yesVotes}`;
```

```
  return sendRequest({ url, data, method })
    .then((data) => data.data)
    .catch((error) => handleError(error));
}

/**
 * method for concern COMMENTS removing
 * @param {string} reviewId
 * @returns {Promise<void>}
 */
async function removeConcernComments(reviewId) {
  const commentIds = await getConcernComments(reviewId);
  for (const id of commentIds) {
    deleteComment(id);
  }
}

/**
 * GETs concern comments from $reviewId
 * @param {string} reviewId
 * @returns {Promise<T | void>}
 */
function getConcernComments(reviewId) {
  const endPoint = `${EndPoints.comments}/?`;
  const filters = [
    `filter[${Relations.type}.id]=${commentTypeId}`,
    `filter[${Relations.context}.type]=${EndPoints.reviews}`,
    `filter[${Relations.context}.id]=${reviewId}`,
  ];
  const url = endPoint + filters.join("&");
  return sendRequest({ url })
    .then((data) => data.data.map((item) => item.id))
    .catch((error) => handleError(error));
}

/**
 * PATCHes concern COMMENT
 * @param {string} commentId
 * @param {string} text
 * @returns {Promise<T | void>}
 */
function patchConcernComment(commentId, text = "") {
  const options = {
    method: "PATCH",
    data: generateCommentBody(text),
    url: `${EndPoints.comments}/${commentId}`,
  };
  return sendRequest(options)
    .then((data) => data.data)
    .catch((error) => handleError(error));
}

/**
 * DELETEs COMMENT
 * @param {string} id
 * @returns {Promise<T | void>}
 */
function deleteComment(id) {
  const method = "DELETE";
  const url = `${EndPoints.comments}/${id}`;
  return sendRequest({ url, method })
    .then((data) => data.data)
    .catch((error) => handleError(error));
```

```
}

module.exports = { main };
```

Приложение 2. Программный модуль веб приложения интеграции разрозненных ИУС

```
"use strict";
const CONFIG = require("./config");
let api = undefined;
if(CONFIG.isUsingV3){
   console.log('Using v3-api');
   api = require("./v3-api");
}else{
   console.log('Using cmp legacy api');
   api = require("./api");
}
const moment = require('moment');
module.exports = {
   parsePendingApprovalCMRs(response) {
      let cmrSet = new Set();
      let irSet = new Set();
      let singleCMR = {}
      if (typeof response === "string") {
         response = JSON.parse(response);
      }
      if (response.error) {
         console.log(response.error);
         return null;
      }
      response.forEach((value) => {
         singleCMR["id"] = value.id;
         singleCMR["title"] = value.title;
         singleCMR["votes"] = value.votes;
         Object.keys(value.votes).forEach(item => {
            if (item.includes("Internal Resources")) {
               value.votes[item].forEach(ele => {
                  irSet.add(ele);
               });
            }
         });
         singleCMR["internal_resources"] = Array.from(irSet);
         cmrSet.add(singleCMR);
         singleCMR = Object.create(null);
         irSet = new Set();
      });
      return cmrSet;
   },
   isAmerican(country) {
      try{
         let flag = false;
         for (let ele of CONFIG.usa) {
            if (country.trim().toUpperCase() === ele.trim().toUpperCase()) {
               flag = true;
```

```
                break;
            }
        }
        return flag;
    }catch(e){
        return false;
    }

},
findAndAddComment(response){
    let users = new Set();
    if((typeof response)!=='object'){
        return []
    }
    Object.keys(response).forEach((item=>{
        response[item].forEach(ele=>{
            users.add(ele);
        })
    }))
    return Array.from(users);
},
async voteRestrictedCMRByID(cmrId,hosts=undefined){
    let activeHosts =  hosts; //Got all us host
    if(!activeHosts){
        activeHosts = await api.getAllActiveHosts();
    }
    let singleCMRHost = await api.findHostsWithCMRID(cmrId); //Get host via cmr-id
    let cmr = await api.getSingleCMRObject(cmrId);
    if(cmr['error']){
        console.log(`CMR ${cmrId} not found.`);
        return {"response":`CMR ${cmrId} not found.`,"error":true};
    }
    let voteList = await api.getVoteListByCmrID(cmrId);
    let voteFlag = false;
    let alreadyApproved = false;
    let addIR2CMR = new Set();
    voteList.forEach(item=>{
        if(item['login']===CONFIG.cmp.user && CONFIG.groupRegexp.test(item['groupTitle'])){
            if(item['isApproved']===0){
                voteFlag = true;
            } else if (item['isApproved'] === 1) {
                alreadyApproved = true;
            }
        }
        if(item['groupTitle'].toLowerCase().includes("Internal Resources".toLowerCase())){
            addIR2CMR.add(item);
        }
    })
    cmr['componentOwnerNotifier'] = false;
    cmr['internal-resources'] = Array.from(addIR2CMR);
    if(singleCMRHost.length===0 && cmr['environment'].title!=='ATT RingCentral.biz' && !alreadyApproved){
        console.log(`${cmr['environment'].title} CMR-${cmrId} without host.`)
        return await api.voteCMRByID(cmrId,false,cmr);
    }
    try{
        singleCMRHost = JSON.parse(singleCMRHost);
    }catch(e){
        singleCMRHost = [];
    }
    cmr.singleCMRHost = singleCMRHost;
    cmr = await this.checkActiveHosts(cmr, activeHosts);

    // ATT RingCentral.biz or Glip Ringcentral.biz
```

```
    let bizEnvsFlag = CONFIG.restrictEnvironments.find(element => cmr['environment'].title.includes(element))
    if(bizEnvsFlag){
      console.log(`${cmr['environment'].title} CMR-${cmrId}`)
      cmr = await this.checkExcecutorAndIR(cmr);
    }
    //End

    if(cmr['against'] && cmr['against'][0]){
      if(!voteFlag){
        return await api.voteCMRByID(cmrId,true,cmr);
      }
      console.log(`This CMR ${cmrId} already voted with 'Raise concern'`);
      return {
        error: true,
        response:{
          message: `This CMR ${cmrId} already voted with 'Raise concern'`
        }
      }
    }else{
      if (alreadyApproved) return {};
      return await api.voteCMRByID(cmrId,false,cmr);
    }
  },
  async checkExcecutorAndIR(cmrFlag){
    try{
      let exemptionUsers = await api.getExemptionName();
      exemptionUsers = new Set(exemptionUsers);
      const personal = [];
      const componentOwnerGuys = cmrFlag['internal-components-owners'] || [];
      const exceptionHosts = CONFIG.specialHosts
      // check executor whether come from USA

      /* CMP-4364: additional rules, applied for Policy bot without component bot, to listed hosts only */
                                    if (!componentOwnerGuys.length && cmrFlag.singleCMRHost.some(host =>
exceptionHosts.includes(host.hostname.trim()))) {
        cmrFlag['against'] = await this.getExceptionHostsConcerns(cmrFlag, exemptionUsers) || [];
        return cmrFlag;
      }


if(!exemptionUsers.has(cmrFlag['executor'].login.toLowerCase())&&!this.isAmerican(cmrFlag['executor'].country)){
        personal.push({id: cmrFlag['executor'].id,name:cmrFlag['executor'].login});
        // https://jira.ringcentral.com/browse/CMP-4141
        if(componentOwnerGuys && componentOwnerGuys.length > 0){
          let hasUS = componentOwnerGuys.filter((item)=>{return item.country.toUpperCase() === 'US'});
          if(hasUS.length>0){
              console.log("CMP-4141:Policy bot should vote 'No objections' because there is an US-based resource in a
list of approvers");
          }else{
            cmrFlag['against'] = personal;
          }
        }else{
          cmrFlag['against'] = personal;
        }

      }
      //check internal resources whether come from USA
      let atLeastOne = cmrFlag['internal-resources'].some(item=>{
        return this.isAmerican(item.country);
      })
      if(atLeastOne){
          console.log(`Policy bot should vote 'no objection' because CMR-${cmrFlag.id} has at least 1 US based Internal
Resource.`);
```

```
            return cmrFlag;
        }else{
            for(let flag of cmrFlag['internal-resources']){
                if(!exemptionUsers.has(flag.login.toLowerCase())&&!this.isAmerican(flag.country)){
                    personal.push({id:flag.id,name: flag.login});
                    console.log(`${flag.login} is not US-Based.`);
                    cmrFlag['against'] = personal;
                }
            }
            return cmrFlag;
        }

    }catch(e){
        console.log("checkExcecutorAndIR",e.message);
        return cmrFlag;
    }

},
async policyBotForAllPN(){
    let activeHosts = await api.getAllActiveHosts(); //Got all us host
    console.log(`${moment().format()} activeHosts`,activeHosts.length);
    let pendingCMRs = await api.findAllPendingApprovalCMRs();
    pendingCMRs = Array.from(this.parsePendingApprovalCMRs(pendingCMRs)); // Get all pending cmrs
    console.log(`${moment().format()} pendingCMRs`,pendingCMRs.length);
    for(let cmr of pendingCMRs){
        if(!cmr.id){
            continue;
        }
        await this.voteRestrictedCMRByID(cmr.id,activeHosts);
    }
},
async voteOkCMRs(cmrs){
    for (let cmr of cmrs){
        let voters = this.findAndAddComment(cmr['votes']);
        let alreadyApproved = false;
        for (let user of voters){
            if (CONFIG.cmp.policybotUserID.indexOf(`${user.user_id}`)==-1) continue;
            if (user['vote'] === 'approve') {
                alreadyApproved = true;
                break;
            }
        }
        if (!alreadyApproved){
            await api.voteCMRByID(cmr.id,false,cmr);
        }
    }
},
async voteConcernCMRs(cmrs){
    for(let cmr of cmrs){
        let voters = this.findAndAddComment(cmr['votes']);
        let need2Vote = false;
        for(let user of voters){
            if(CONFIG.cmp.policybotUserID.indexOf(`${user.user_id}`)!=-1){
                if(user['vote']!=='reject'){
                    need2Vote = true;
                    console.log(`This is ${user.vote} CMR-${cmr.id}, Will do voting with Raise concern.`);
                    continue;
                }else{
                    console.log(`The CMR-${cmr.id} already has ${user.vote} vote`);
                }
            }
        }
        if(need2Vote){
```

```
            console.log(`Adding Raise concern to CMR-${cmr.id}`);
            await api.voteCMRByID(cmr.id,true,cmr);
        }
    }
},
/**
 * checks for CMP-4364-case concerns
 * @param {Object} cmr
 * @param {Set} exemptionUsers
 * @returns {{name: *, id: *}[]|undefined}
 */
async getExceptionHostsConcerns(cmr, exemptionUsers) {
    const hostTeamRules = await this.getHostToDepartmentMap(CONFIG.specialHosts);
    console.log(`CMR-${cmr.id}: checking the CMP-4364-case`);

    /* If executor is US-based - policy bot votes no objections */
    if (exemptionUsers.has(cmr['executor'].login.toLowerCase()) || this.isAmerican(cmr['executor'].country)) {
        console.log(`CMR-${cmr.id}: executor is US-based - No objections`);
        return;
    }

        /* If executor is non-US and there is at least one US-based person from hostTeamRules department as internal
resources - policy bot votes no objections */
    for (const ruleHost in hostTeamRules) {
        if (!cmr.singleCMRHost.some(cmrHost => cmrHost.hostname.trim() === ruleHost)) {
            continue;
        }

        const owner = hostTeamRules[ruleHost];
        const check = cmr['internal-resources'].some(resource => {
            if (!this.isAmerican(resource.country)) {
                return;
            }
            const group = resource.groupTitle.split(' | ').pop();
            return owner === group;
        });

        if (!check) {
            /* If executor is non-US and Owner member is missing from the IR - policy bot votes concern */
            console.log(`CMR-${cmr.id}: executor is non-US and no IR for '${ruleHost}' - Concern`);
            return [{id: cmr['executor'].id, name:cmr['executor'].login}];
        }
    }

    console.log(`CMR-${cmr.id}: executor is non-US but all required IRs are present - No objections`);
},
/**
 * checking active hosts here, separated into a method for a proper exit
 * @param {Object} cmr
 * @param {Array} activeHosts []
 * @returns {Promise<*|undefined>|*}
 */
checkActiveHosts(cmr, activeHosts = []) {
    for (let item of activeHosts){
        for (let ele of cmr.singleCMRHost) {
            // check whether us host
            if (ele.hostname && item.hostname && ele.hostname.trim() === item.hostname.trim()) {
                cmr['componentOwnerNotifier'] = true;
                return this.checkExcecutorAndIR(cmr);
            }
        }
    }
    return cmr;
```

```
    },
    /**
     * maps hosts with CMP departments
     * @param {Array} hosts
     * @returns {Promise<{}>}
     */
    async getHostToDepartmentMap(hosts = []) {
        const returnObj = {};
        if (!hosts || !hosts.length) return returnObj;
        const components = this.getComponentsFromHosts(hosts);
        const componentMapping = await this.getComponentToDepartmentMap(components);
        hosts.forEach(host => {
            const component = this.getComponentsFromHosts([host]).pop();
            returnObj[host] = componentMapping[component];
        });
        return returnObj;
    },
    /**
     * parses components from hosts
     * @param {Array} hosts
     * @returns {*[]}
     */
    getComponentsFromHosts(hosts = []) {
            const components = hosts.map(host => host.split('-').pop().match(/[a-z]+/ig).pop().toUpperCase()).filter(item =>
!!item);
        return [...new Set(components)];
    },
    /**
     * maps components to CMP departments
     * @param {Array} components
     * @returns {Promise<{}>}
     */
    async getComponentToDepartmentMap(components = []) {
        const returnObj = {};
        if (!components || !components.length) {
            return returnObj;
        }
        const componentOwners = await api.getComponentOwners(components) || [];
        const departmentMappings = await api.getDepartmentMappings() || [];
        const ownersObj = this.arrayToObject(componentOwners, 'componentName');
        const mappingsObj = this.arrayToObject(departmentMappings, 'planitTeamID');
        components.forEach(item => {
            const team = ownersObj[item] && ownersObj[item]['teamId'];
            if (!team) return;
            const department = mappingsObj[team];
            if (!department) return;
            returnObj[item] = department['cmpDepartmentName'];
        })
        return returnObj;
    },
    /**
     * converts array into an object with specified key as "id"
     * @param {Array} payload
     * @param {String} key
     * @returns {{}}
     */
    arrayToObject(payload = [], key = '') {
        const returnObj = {};
        payload.forEach(item => {
            const lowerKey = item[key];
            returnObj[lowerKey] = item;
        })
        return returnObj;
```

```
  },
};
```

Приложение 3. Программный модуль формирования последовательности внесений изменений в глобально распределенную информационную систему

РОССИЙСКАЯ ФЕДЕРАЦИЯ

ФЕДЕРАЛЬНАЯ СЛУЖБА
ПО ИНТЕЛЛЕКТУАЛЬНОЙ СОБСТВЕННОСТИ

RU2021663522

ГОСУДАРСТВЕННАЯ РЕГИСТРАЦИЯ ПРОГРАММЫ ДЛЯ ЭВМ

Номер регистрации (свидетельства):
2021663522
Дата регистрации: **18.08.2021**
Номер и дата поступления заявки:
2021662772 10.08.2021
Дата публикации и номер бюллетеня:
18.08.2021 Бюл. № 8

Автор(ы):
Буйневич Михаил Викторович (RU),
Вострых Алексей Владимирович (RU),
Ефимов Вадим Вячеславович (RU)
Правообладатель(и):
Федеральное государственное бюджетное образовательное учреждение высшего образования «Санкт-Петербургский университет Государственной противопожарной службы Министерства Российской Федерации по делам гражданской обороны, чрезвычайным ситуациям и ликвидации последствий стихийных бедствий (RU)

Название программы для ЭВМ:
Программный модуль формирования последовательности внесений изменений в глобально распределенную информационную систему

Реферат:
Назначением разработанного программного модуля является формирование последовательности внесения изменений в глобально распределенную информационную систему (далее - ГРИС) с учетом её текущей конфигурации и стратегий безопасного внесения изменений, принятых в той или иной организации. Это позволяет обеспечить полноту внесений изменений, что актуально для ГРИС с динамической конфигурацией, подстраивающейся для обработки изменяющегося во времени количества запросов пользователей, а также придерживаться безопасной стратегии внесений изменений и оптимизировать время внесения изменений. Тип ЭВМ: программный модуль запускается с любого стационарного или переносного электронно-вычислительного устройства, такого как ноутбук или персональный компьютер. ОС: MS Windows, Mac OS, Linux, Android, IOS.

Язык программирования: JavaScript

Объем программы для ЭВМ: 25 151 Б