МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РЕСПУБЛИКИ КАЗАХСТАН
Некоммерческое акционерное общество
АЛМАТИНСКИЙ УНИВЕРСИТЕТ ЭНЕРГЕТИКИ И СВЯЗИ
имени Гумарбека Даукеева

Кафедра «Телекоммуникационные сети и системы»
Специальность: 6М071900 «Радиотехника, электроника и телекоммуникации»

ДОПУЩЕН К ЗАЩИТЕ
Зав. кафедрой
PhD, доцент Темырканова Э.К.
(ученая степень, звание, ФИО)

_____
(подпись)
«_____»_____2020 г.

МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ
пояснительная записка

на тему: «Алгоритмы автоматизированного выявления и устранения сбоев в распределенных инфокоммуникационных системах»

Магистрант: Окшин В.П _____ группа МРЭТн 13-2
(Ф.И.О.) (подпись)
Руководитель: PhD, профессор _____ Чайко Е.В
(ученая степень, звание (подпись) (Ф.И.О.)

Рецензент_____ _____ _____
(ученая степень, звание) (подпись) (Ф.И.О.)

Консультант по ВТ PhD, профессор _____ Чайко Е.В
(ученая степень, звание) (подпись) (Ф.И.О.)

Нормоконтроль: PhD, профессор _____ Чайко Е.В
(ученая степень, звание) (подпись) (Ф.И.О.)

Алматы 2020

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РЕСПУБЛИКИ КАЗАХСТАН
Некоммерческое акционерное общество
АЛМАТИНСКИЙ УНИВЕРСИТЕТ ЭНЕРГЕТИКИ И СВЯЗИ
имени Гумарбека Даукеева

Институт Космической Инженерии и Телекоммуникаций

Специальность: 6М071900 «Радиотехника, электроника и телекоммуникации»

Кафедра: «Телекоммуникационные сети и системы»

ЗАДАНИЕ
на выполнение магистерской диссертации

Магистранту Окшину Валерию Павловичу
(фамилия, имя, отчество)

Тема диссертации «Алгоритмы автоматизированного выявления и устранения сбоев в распределенных инфокоммуникационных системах»

Утверждена Ученым советом университета № 122 от 25 октября 2018
Срок сдачи законченной диссертации «25» мая 2020г.
Цель исследования состоит в экспериментальном анализе возможности поиска и устранения неполадок в распределенных системах с помощью применения метода тематического моделирования

Перечень подлежащих разработке в магистерской диссертации вопросов или краткое содержание магистерской диссертации:

1. Изучение процесса возникновения сбоев в распределенных системах
2. Исследование возможности автоматизированного устранения сбоев в распределенных системах
3. Экспериментальный анализ событий с помощью метода тематического моделирования

Перечень графического материала (с точным указанием обязательных чертежей)

Рисунок 1.1 - Распределенная система, организованная на уровне программного обеспечения промежуточного слоя
Рисунок 1.3 - Многоуровневая архитектура для сетевых вычислительных систем
Рисунок 2.1 - Процесс порождения текстовой коллекции вероятностной тематической моделью
Рисунок 3.8 - График перплексии для модели PLSA
Рисунок 3.25 - Зависимость метрики SparsityTheta от коэффициента τ

Рекомендуемая основная литература

1. Макаренко С. И. Системы многоканальной связи. Вторичные сети и сети абонент¬ского доступа: учебное пособие / С.И. Макаренко, В.Е. Федосеев. - СПб.: ВКА имени А.Ф. Можайского, 2014. - 179 с

2. Гасымов И. Архитектура оптических сетей доступа FTTH (Fiber-to-the-Home) // Официальный документ компании CiscoSystemInc. 2007. - 12 с.

Г Р А Ф И К
подготовки магистерской диссертации

Наименование разделов, перечень разрабатываемых вопросов	Сроки представления научному руководителю	Примечание
1. Информационный обзор согласно теме	05.10.2018	
2. Основные виды и типы архитектур распределенных систем (теоретическая часть)	14.01.2018	
3. Исследование возможности применения метода тематического моделирования для выявления сбоев (исследовательская глава)	02.02.2018	
4. Эксперимент по выявлению тематик, описывающих проблему (расчетная часть)	18.10.2019	
5. Анализ полученных экспериментальных и расчетных данных	10.12.2019	

Дата выдачи задания_30 сентября 2018г._____

Заведующий кафедрой _____ (Темырканова Э.К.)
(подпись) (Ф.И.О.)

Научный
руководитель диссертации _____ (Чайко Е. В.)
(подпись) (Ф.И.О.)

Задание принял к исполнению
магистрант _____ (Окшин В.П.)
(подпись) (Ф.И.О.)

Аңдатпа

Бұл жұмыста телекоммуникациялық оператор оқиғаларының мониторингі деректерін талдау әдісі іске асырылды. Ұсынылатын әдістің негізі – тақырыптық модельдеу әдісі. Бұл әдіс табиғи тілдердің мәтіндерін талдау үшін кеңінен қолданылады. Мониторинг жүйесімен тіркелетін оқиғалар алдамшы сөз оқиғалардың атаулары немесе олардың идентификаторлары болып табылатын күрделі тіл ретінде қарастыруға болады.Бірнеше мониторинг объектілерінен ұзақ кезең ішіндегі оқиғалар туралы деректер болған кезде “мәтіндердің" D бастапқы матрицасы құрылады. Модельдеудің тақырыптық алгоритмін іске асыру ең көп таралған "тақырыптарды" бөліп, әрбір мәтінге есептелген ықтималдықпен беруге мүмкіндік береді. Бұл өз кезегінде нақты проблеманың қайнар көзін табуға мүмкіндік береді. Ұсынылып отырған әдіс жұмысының нәтижесінде мониторинг объектілері оқиғаларының ағынын жеке санаттарға жіктеуге болады, бұл диагностиканы және ақауларды жоюды орындайтын персоналдың жұмысын жеңілдетеді.

Аннотация

В настоящей работе предлагается метод анализа данных мониторинга событий телекоммуникационного провайдера. Основа предлагаемого метода – метод тематического моделирования, который широко используется для анализа текстов естественных языков. Поток событий, фиксируемый системой мониторинга, можно рассматривать как текст псевдоязыка, словами которого являются названия событий или их идентификаторы. Имея данные о событиях с нескольких объектов мониторинга за продолжительный промежуток времени, составляется исходная матрица «текстов» D. Реализация алгоритма тематического моделирования позволяет выделить распространенные «темы» и присваивает их к каждому тексту с вычисленной вероятностью. Это, в свою очередь, позволяет найти источник проблемы из текущей картины событий. В результате работы предлагаемого метода можно классифицировать поток событий объектов мониторинга на отдельные категории, что упрощает работу персонала, выполняющего диагностику и устранение неисправностей.

Abstract

In this work the method of analysis of monitoring data of events of the telecommunication operator is realized. The basis of the proposed method is the method of thematic modeling. This method is widely used for analyzing texts of natural languages. The event stream recorded by the monitoring system can be considered as the text of a pseudo-language whose words are the names of events or their identifiers. Having data about events from several monitoring objects over a long period of time, the initial matrix of "texts" D is compiled. The implementation of the thematic modeling algorithm allows you to select common "themes" and assign them to each text with a calculated probability. This, in turn, allows you to find the source of the problem from the current picture of events. As a result of the proposed method, it is possible to classify the event flow of monitoring objects into separate categories, which simplifies the work of personnel performing diagnostics and elimination of inconsistencies.

Содержание

Введение

При эксплуатации различных распределенных систем возникает задача не только их мониторинга и выявления текущих проблем в оных, но и поиск «первоисточников» этих проблем, что в виду сложности и масштабности таким систем, является не тривиальной задачей. Аппаратные компоненты распределенных систем рассчитаны на ограниченный срок службы. В свою очередь, любое программное обеспечение может аварийно завершить свою работу в самый неподходящий момент. Может показаться, что работоспособность компонента – это бинарный параметр: компонент либо работает, либо совсем не работает и не может быть использован. Однако, зачастую выходу из строя компонента может предшествовать ряд связанных предварительных событий. Сбой компонентов приводит не только к завершению их работы, но и наносит непредсказуемый, а иногда и несоразмерный вред системе. Следовательно, во избежание возникновения подобных ситуаций необходимо иметь возможность прогнозирования появления проблем в системе. Иными словами, задача состоит в том, чтобы, имея некую картину событий на данный момент, а также хронологическую последовательность событий до этого, проанализировать эту информацию и выявить взаимосвязи между событиями, которые и создали текущую картину.

Сам процесс обработки данных представляет собой цепочку «сбор-хранение-анализ». Самым сложным процессом в данной цепочке является анализ. Ведь сама картина представляет собой хронологическую цепочку событий. Отсюда возникает вопрос: каким образом можно выявить связи между событиями? Совершенно очевидно, что в потоке событий будут соседствовать события совершенно независимые между собой и в то же время события, имеющие непосредственную связь, могут быть значительно разнесены во времени. В данный момент задача установления связи между событиями, а равно и поиск источника проблем возложена на плечи специалиста. Только специалист с должной квалификацией может найти и проинтерпретировать связи между событиями. Разумеется, такой подход требует значительное количество времени на поиск и устранение инцидентов, что, в свою очередь, будет негативно сказываться на оперативности решения проблем и общем неудовлетворительном состоянии системы.

Соответственно, становится совершенно очевидной необходимость создания некого автоматизированного решения. Одним из путей решения такой задачи является применение алгоритмов машинного обучения.

Направление тематического моделирования активно развивается в контексте машинного обучения уже более 15 лет. Это современная тенденция в статистическом анализе данных. Задачей тематического моделирования является поиск ответа на вопрос к какой теме относится большая коллекция текстов или иных данных. В общем случае это сводится к анализу частоты появления слов в текстах.

Каждая тема характеризуется своим словарем с вероятностями терминов, $p(w \mid t)$ – частота термина w в теме t. Таким образом, один текст может содержать несколько тем с определенной долей вероятности.

До последнего времени тематическое моделирование использовалось для анализа и структурирования различных текстов. Решались задачи классификации и категоризации документов, автоматического аннотирования, тематической сегментации документов и пр.

Но, так как алгоритм проводит статистический анализ текстов, а не семантический, то можно предположить, что подобному анализу могут быть подвержены данные любого характера.

1 Распределенная система как объект наблюдения и анализа

1.1 Организация распределенной системы

Распределенная система представляет собой совокупность автономных вычислительных элементов, которые представляются ее пользователям в виде единой согласованной системы [1].

Это определение относится к двум характерным особенностям распределенных систем. Первое состоит в том, что распределенная система представляет собой совокупность вычислительных элементов, каждый из которых способен функционировать независимо от других. Вычислительный элемент, который обычно называется узлом, может быть аппаратным устройством или программным процессом. Вторая особенность заключается в том, что пользователи (будь то люди или приложения) полагают, что имеют дело с единой системой. Это означает, что так или иначе автономные узлы должны сотрудничать. Задача установки этого сотрудничество лежит в основе разработки распределенных систем. Типы узлов могут быть самыми различными. Даже в пределах одной системы они могут варьироваться от высокопроизводительных мэйнфреймов до небольших устройств в сенсорных сетях. Кроме того, узлы системы могут быть связаны между собой самым различным образом.

Современные распределенные системы часто состоят из разных типов узлов, от очень больших высокопроизводительных компьютеров до небольших подключаемых компьютеров или даже меньших устройств. Фундаментальный принцип заключается в том, что узлы могут действовать независимо друг от друга, хотя в то же время совершенно очевидно, что если узлы системы игнорируют друг друга, то нет смысла помещать их в одну распределенную систему. На практике узлы программируются для достижения общих целей, которые реализуются путем обмена сообщениями друг с другом. Узел реагирует на входящие сообщения, которые затем обрабатываются и, в свою очередь, ведут к дальнейшему взаимодействию между узлами посредством передачи сообщений.

Важным наблюдением является то, что вследствие работы группы независимых узлов у каждого узла будет свое представление о времени. Отсутствие общего эталона времени приводит к фундаментальным вопросам, касающимся синхронизации и координации в распределенной системе. Помимо этого также появляется задача управлять членством и организацией этой коллекции устройств. Другими словами, может потребоваться фиксирование информации о том, какие узлы могут принадлежать или не принадлежать системе, а также предоставить каждому участнику список узлов, с которыми он может взаимодействовать напрямую.

Управление членством в группе может быть чрезвычайно сложным, хотя бы по причинам контроля допуска. К примеру, мы делаем различие между открытыми и закрытыми группами. В открытой группе любому узлу разрешено присоединяться к распределенной системе, что фактически

означает, что он может отправлять сообщения любому другому узлу в системе. Напротив, в закрытой группе только члены этой группы могут общаться друг с другом, и для того, чтобы узел мог присоединиться или покинуть группу, необходим отдельный механизм.

Очевидно, что задача контроля допуска не является легкой. Во-первых, необходим механизм аутентификации узла. При неправильно спроектированном механизме, управление аутентификацией может легко создать узкое место масштабируемости. Во-вторых, каждый узел должен иметь возможность проверить, действительно ли он общается с другим членом группы, а не, например, с злоумышленником, стремящимся навредить системе. Наконец, учитывая тот факт, что участник может легко общаться не с участниками системы, при условии, что конфиденциальность является проблемой во взаимодействии в распределенной системе, во весь рост встает с проблемами доверия.

Что касается организации сети, практика показывает, что распределенная система часто организована как оверлейная сеть [12]. В этом случае узел, как правило, представляет собой программный процесс, снабженный списком других процессов, которым он может напрямую отправлять сообщения. Передача сообщений затем осуществляется по каналам TCP/IP или UDP, но, кроме того, также могут быть доступны средства более высокого уровня. Существует примерно два типа оверлейных сетей:

– структурированное наложение – в этом случае каждый узел имеет четко определенный набор соседей, с которыми он может общаться. Например, узлы организованы в виде дерева или логического кольца;

– неструктурированное наложение. В таком наложении каждый узел имеет ряд ссылок на случайно выбранные другие узлы.

В любом случае, оверлейная сеть, в принципе, должна быть всегда подключена, это означает, что между любыми двумя узлами всегда существует канал связи, позволяющий этим узлам направлять сообщения от одного к другому. Хорошо известный класс наложений формируется одноранговыми (P2P) сетями. Важно понимать, что организация узлов требует особых усилий и что иногда это одна из наиболее сложных частей управления распределенными системами.

Чтобы помочь разработке распределенных приложений, распределенные системы часто организованы так, чтобы иметь отдельный уровень программного обеспечения, который логически размещается поверх соответствующих операционных систем компьютеров, которые являются частью системы. Эта организация показана на рисунке 1.1, что приводит к так называемому промежуточному слою программного обеспечения [13].

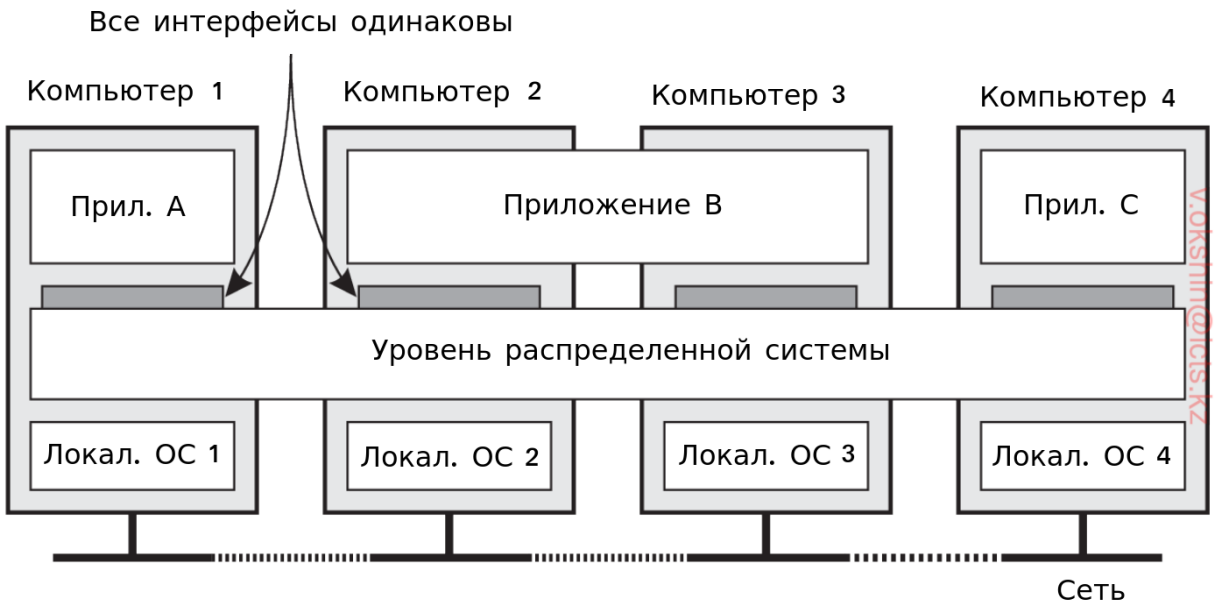

Рисунок 1.1 – Распределенная система, организованная на уровне программного обеспечения промежуточного слоя, которая распространяется на несколько компьютеров, предлагая каждому приложению один и тот же интерфейс

На Рисунке 1.1 показаны четыре сетевых компьютера и три приложения, из которых приложение В распределено по компьютерам 2 и 3. Каждому приложению предлагается один и тот же интерфейс. Распределенная система предоставляет средства для связи компонентов одного распределенного приложения друг с другом, но также позволяет взаимодействовать различным приложениям. В то же время она скрывает, насколько это возможно и разумно, различия в аппаратных средствах и операционных системах для каждого приложения.

В некотором смысле, промежуточное программное обеспечение для распределенной системы играет ту же роль, что и операционная система для компьютера, а именно роль менеджера ресурсов, предлагающего свои приложения для эффективного совместного использования и развертывания этих ресурсов в сети. Помимо управления ресурсами, он предлагает услуги, которые также можно найти в большинстве операционных систем, в том числе:

– средства для взаимодействия приложений;

– сервисы безопасности;

– маскировка сбоев и восстановление после них.

Основное программного обеспечения промежуточного слоя от операционных систем заключается в том, что функционирование промежуточного программного обеспечения осуществляется в сетевой среде. Также стоит обратить внимание, что большинство сервисов применимо для многих приложений. В этом смысле промежуточное программное обеспечение также можно рассматривать как контейнер часто используемых компонентов и функций, которые теперь больше не должны реализовываться

приложениями отдельно. Чтобы дополнительно проиллюстрировать эти моменты, кратко рассмотрим несколько примеров типичных сервисов промежуточного программного обеспечения:

– связь. Общим сервисом коммуникации является так называемый удаленный вызов процедур (RPC). Служба RPC позволяет приложению вызывать функцию, которая реализована и выполняется на удаленном компьютере, как если бы она была доступна локально. Для этого разработчику нужно просто указать заголовок функции, выраженный на специальном языке программирования, из которого подсистема RPC может затем сгенерировать необходимый код, который устанавливает удаленные вызовы;

– транзакции. Многие приложения используют несколько сервисов, которые распределены между несколькими компьютерами. Промежуточное программное обеспечение обычно предлагает специальную поддержку для выполнения таких сервисов в режиме «все или ничего», обычно называемом атомарной транзакцией. В этом случае разработчику приложения нужно только указать задействованные удаленные сервисы, и, следуя стандартному протоколу, промежуточное программное обеспечение гарантирует, что вызывается каждый сервис или вообще не один;

– состав услуг: становится все более распространенной разработка новых приложений путем объединения существующих программ и их «склеивания». Это особенно затрагивает многие веб-приложения, в частности те, которые являются веб-сервисами [14]. Промежуточное программное обеспечение на основе интернета может помочь, стандартизируя доступ к веб-службам и предоставляя средства для генерации их функций в определенном порядке. Простой пример того, как разворачивается состав службы, формируется гибридными приложениями: веб-страницы, которые объединяют данные из разных источников. Хорошо известными гибридными приложениями являются те, которые основаны на картах Google, в которых карты дополнены дополнительной информацией, такой как планировщики путешествий или прогнозы погоды в реальном времени;

– надежность. В качестве последнего примера было проведено множество исследований по предоставлению улучшенных функций для создания надежных распределенных приложений. Инструментарий Horus [15] позволяет разработчику создавать приложение в виде группы процессов таким образом, чтобы любое сообщение, отправленное одним процессом, гарантированно принималось всем или никаким другим процессом. Как оказалось, такие гарантии могут значительно упростить разработку распределенных приложений и обычно реализуются как часть промежуточного программного обеспечения.

1.2 Типы распределенных систем

Важным классом распределенных систем является тот, который используется для высокопроизводительных вычислительных задач. Грубо говоря, можно провести различие между двумя подгруппами. В кластерных вычислениях базовое оборудование состоит из набора похожих рабочих

станций или компьютеров, тесно связанных посредством высокоскоростной локальной сети. Кроме того, на каждом узле работает одна и та же операционная система.

Ситуация становится совсем другой в случае грид-вычислений. Эта подгруппа состоит из распределенных систем, которые часто создаются как объединение компьютерных систем, где каждая система может находиться в другом административном домене и может сильно отличаться, если речь идет об аппаратном, программном обеспечении и используемой сетевой технологии.

С точки зрения грид-вычислений, следующий логический шаг - просто передать на аутсорсинг всю инфраструктуру, которая необходима для приложений с интенсивными вычислениями. По сути, в этом и заключается суть облачных вычислений: предоставление средств для динамического построения инфраструктуры и составления того, что необходимо из доступных сервисов. В отличие от грид-вычислений, которые тесно связаны с высокопроизводительными вычислениями, облачные вычисления - это гораздо больше, чем просто предоставление большого количества ресурсов.

Кластерные вычислительные системы обрели популярность, когда соотношение цены и производительности персональных компьютеров и рабочих станций улучшилось. В определенный момент стало финансово и технически привлекательным создать суперкомпьютер с использованием готовой технологии, просто подключив набор относительно простых компьютеров в высокоскоростной сети. Практически во всех случаях кластерные вычисления используются для параллельного программирования, в котором одна (требующая большого объема вычислений) программа выполняется параллельно на нескольких машинах.

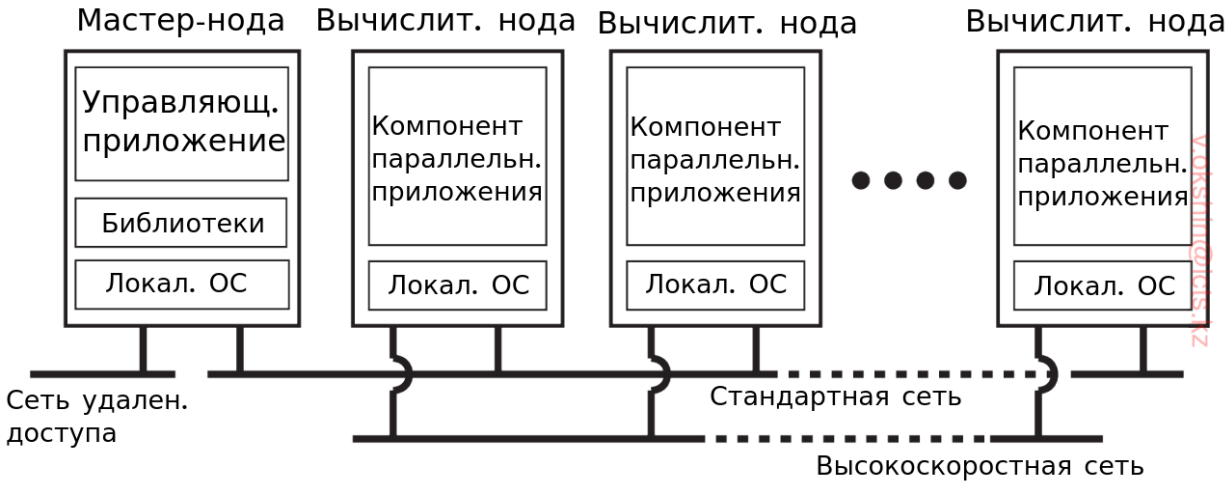

Рисунок 1.2 – Пример кластерной вычислительной системы

Пример кластерного вычисления образован кластерами Beowulf на основе Linux, общая конфигурация которых показана на рисунке 1.2. Каждый кластер состоит из набора вычислительных узлов, которые контролируются и

доступны с помощью одного главного узла. Мастер обычно обрабатывает распределение узлов для конкретной параллельной программы, поддерживает пакетную очередь отправленных заданий и предоставляет интерфейс для пользователей системы. Таким образом, мастер фактически запускает промежуточное программное обеспечение, необходимое для выполнения программ и управления кластером, в то время как вычислительные узлы оснащены стандартной операционной системой, дополненной типичными функциями промежуточного программного обеспечения для связи, хранения, отказоустойчивости и так далее. Таким образом, кроме главного узла, вычислительные узлы очень идентичны.

Еще более симметричный подход используется в системе MOSIX [17]. MOSIX пытается предоставить односистемный образ кластера, что означает, что для процесса компьютер кластера обеспечивает максимальную прозрачность распределения, представившись одним компьютером. Как уже упоминалось выше, предоставление такого изображения при любых обстоятельствах крайне затруднительно. В случае MOSIX высокая степень прозрачности обеспечивается за счет того, что процессы могут динамически мигрировать между узлами, составляющими кластер. Процесс миграции позволяет пользователю запускать приложение на любом узле (называемом домашним узлом), после чего он может прозрачно перемещаться на другие узлы, например, для эффективного использования ресурсов.

Однако несколько современных кластерных компьютеров отошли от этих симметричных архитектур к более гибридным решениям, в которых промежуточное программное обеспечение функционально распределено по разным узлам, как объясняется в [18]. Преимущество такого разделения очевидно: наличие вычислительных узлов с выделенными, облегченными операционными системами, скорее всего, обеспечит оптимальную производительность для приложений, интенсивно использующих вычислительные ресурсы. Аналогично, функциональность хранилища, скорее всего, может оптимально обрабатываться другими специально настроенными узлами, такими как файловый сервер и сервер каталогов. То же самое относится и к другим специализированным сервисам промежуточного программного обеспечения, включая управление заданиями, сервисы баз данных и, возможно, общий доступ в Интернет к внешним сервисам.

Характерной чертой традиционных кластерных вычислений является их однородность. В большинстве случаев компьютеры в кластере в основном одинаковы, имеют одинаковую операционную систему и все подключены через одну сеть. Однако, как говорилось выше, наблюдается тенденция к созданию более гибридных архитектур, в которых узлы специально настроены для определенных задач. Это разнообразие еще более распространено в сетевых вычислительных системах: не делается никаких предположений относительно сходства аппаратного обеспечения, операционных систем, сетей, административных доменов, политик безопасности и т. д.

Ключевой проблемой в системе грид-компьютинга является то, что ресурсы разных организаций объединяются, чтобы обеспечить совместную работу группы людей из разных учреждений, фактически формируя «федерацию» систем. Такое совместное функционирование осуществляется в форме виртуальной организации. Процессы, принадлежащие одной и той же виртуальной организации, имеют права доступа к ресурсам, предоставленным этой организации. Как правило, ресурсы состоят из вычислительных серверов, хранилищ и баз данных.

Учитывая его природу, большая часть программного обеспечения для реализации грид-вычислений развивается вокруг предоставления доступа к ресурсам из разных административных доменов и только тем пользователям и приложениям, которые принадлежат конкретной виртуальной организации. По этой причине основное внимание часто уделяется архитектурным вопросам. Архитектура, первоначально предложенная в [19] показана на рисунке 1.3, по-прежнему лежит в основе многих вычислительных систем.

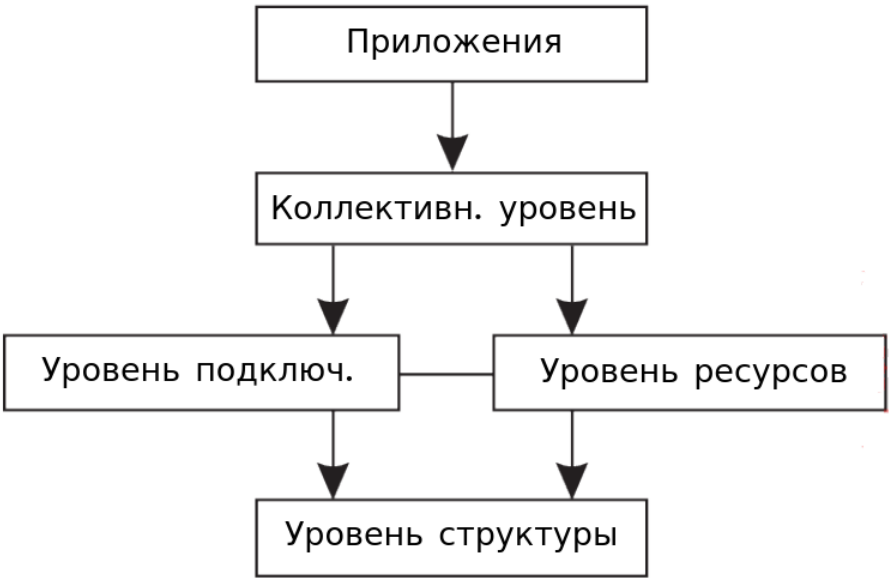

Рисунок 1.3 – Многоуровневая архитектура для сетевых вычислительных систем

Архитектура состоит из четырех слоев. Самый нижний уровень структуры предоставляет интерфейсы для локальных ресурсов на определенном сайте. Стоит обратить внимание, что эти интерфейсы предназначены для совместного использования ресурсов в виртуальной организации. Как правило, они предоставляют функции для запроса состояния и возможностей ресурса, а также функции для фактического управления ресурсами (например, блокировки ресурсов).

Уровень подключения состоит из протоколов связи для поддержки сетевых транзакций, которые охватывают использование нескольких ресурсов. Например, протоколы необходимы для передачи данных между ресурсами или просто для доступа к ресурсу из удаленного местоположения.

Кроме того, уровень подключения будет содержать протоколы безопасности для аутентификации пользователей и ресурсов. Необходимо отметить, что во многих случаях пользователи не проходят аутентификацию; вместо этого программы, действующие от имени пользователей, проходят проверку подлинности. В этом смысле делегирование прав от пользователя программам является важной функцией, которую необходимо поддерживать на уровне связности.

Уровень ресурсов отвечает за управление одним ресурсом. Он использует функции, предоставляемые уровнем связности, и напрямую вызывает интерфейсы, предоставляемые уровнем структуры. Например, этот уровень будет предлагать функции для получения информации о конфигурации конкретного ресурса или, в общем, для выполнения определенных операций, таких как создание процесса или чтение данных. Таким образом, уровень ресурсов, как считается, отвечает за управление доступом и, следовательно, будет полагаться на аутентификацию, выполняемую как часть уровня связности.

Следующий уровень в иерархии - это коллективный уровень. Он имеет дело с обработкой доступа к нескольким ресурсам и обычно состоит из сервисов для обнаружения ресурсов, распределения и планирования задач по нескольким ресурсам, репликации данных и так далее. В отличие от уровня связности и ресурсов, каждый из которых состоит из сравнительно небольшого стандартного набора протоколов, коллективный уровень может состоять из множества различных протоколов, отражающих широкий спектр услуг, которые он может предложить виртуальной организации.

Наконец, прикладной уровень состоит из приложений, которые работают в виртуальной организации и которые используют сетевую вычислительную среду.

Обычно уровни коллективности, подключения и ресурсов формируют ядро того, что можно назвать уровнем промежуточного программного обеспечения. Эти уровни совместно обеспечивают доступ к ресурсам, которые потенциально могут быть распределены по нескольким сайтам, и управление ими.

Важным наблюдением с точки зрения промежуточного программного обеспечения является то, что в грид-вычислениях понятие сайта (или административной единицы) является распространенным. Эта распространенность подчеркивается постепенным переходом к сервис-ориентированной архитектуре, в которой сайты предлагают доступ к различным уровням через набор веб-сервисов [20]. К настоящему времени это привело к определению альтернативной архитектуры, известной как Open Grid Services Architecture (OGSA) [21]. OGSA основана на оригинальных идеях, сформулированных [19], однако процесс стандартизации делает его сложным, если не сказать больше. Реализации OGSA обычно соответствуют стандартам web-сервисов.

Пока исследователи размышляли о том, как организовать грид-вычисления, которые были бы легко доступны, организации, отвечающие за

эксплуатацию центров обработки данных, столкнулись с проблемой открытия своих ресурсов для клиентов. В конечном итоге это привело к концепции вспомогательных вычислений, с помощью которых клиент может загружать задачи в центр обработки данных и использовать его ресурсы для вычислений. Эти служебные вычисления легли в основу того, что сейчас называется облачными вычислениями.

Следуя [22], облачные вычисления характеризуются легко используемым и доступным пулом виртуализированных ресурсов. То, как и какие используются ресурсы, может быть настроено динамически, обеспечивая основу для масштабируемости: если требуется выполнить больше работы, клиент может просто получить больше ресурсов. Связь с сервисными вычислениями формируется тем фактом, что облачные вычисления обычно основаны на модели с оплатой за использование, в которой гарантии предоставляются посредством соглашений об уровне обслуживания (SLA).

Рисунок 1.4 – Структура облака

На практике облака состоят из четырех уровней [23], как показано на рисунке 1.4.

Аппаратное обеспечение. Самый нижний уровень формируется средствами управления необходимым аппаратным обеспечением: процессорами, маршрутизаторами, а также системами питания и охлаждения. Обычно он применяется в центрах обработки данных и содержит ресурсы, которые клиенты обычно никогда не видят напрямую.

Инфраструктура. Это важный уровень, формирующий основу для большинства платформ облачных вычислений. Он развертывает методы виртуализации, чтобы предоставить клиентам инфраструктуру, состоящую из виртуальной памяти и вычислительных ресурсов.

Платформа. Можно утверждать, что уровень платформы предоставляет заказчику облачных вычислений то, что операционная система предоставляет

разработчикам приложений, а именно средства для простой разработки и развертывания приложений, которые должны работать в облаке. На практике разработчику приложения предлагается специфичный для поставщика API, который включает вызовы для загрузки и выполнения программы в облаке этого поставщика. В некотором смысле это сопоставимо семейству системных вызовов семейства Unix exec, которые принимают исполняемый файл в качестве параметра и передают его в операционную систему для выполнения.

Как и операционные системы, уровень платформы предоставляет высокоуровневые абстракции для хранения и тому подобное. Например, система хранения Amazon S3 [24] предлагается разработчику приложения в форме API, позволяющего (локально созданным) файлам организовывать и хранить их в контейнерах. Контейнер чем-то сравним с каталогом. Сохраняя файл в контейнере, этот файл автоматически загружается в облако Amazon.

Приложение. Актуальные приложения выполняются на этом уровне и предлагаются пользователям для дальнейшей настройки. Хорошо известные примеры включают в себя офисные пакеты (текстовые процессоры, приложения для работы с электронными таблицами, приложения для презентаций и т. д.). Важно понимать, что эти приложения выполняются в облаке поставщика. Как и прежде, их можно сравнить с традиционным набором приложений, которые поставляются при установке операционной системы.

Поставщики облачных вычислений предлагают эти уровни своим клиентам через различные интерфейсы (включая инструменты командной строки, программные интерфейсы и веб-интерфейсы), что приводит к трем различным типам услуг:

– инфраструктура как услуга (IaaS) – охватывается уровень оборудования и инфраструктуры;

– платформа как услуга (PaaS) – охватывается уровень платформы;

– программное обеспечение как услуга (SaaS) – охватывается уровень приложения.

На данный момент использовать облака относительно легко. Как следствие, облачные вычисления как средство аутсорсинга локальных вычислительных инфраструктур стали серьезным выбором для многих предприятий. Тем не менее, по-прежнему существует ряд серьезных препятствий, в том числе проблемы с блокировкой провайдера, вопросы безопасности и конфиденциальности, а также зависимость от доступности услуг [25]. Кроме того, поскольку детали того, как на самом деле выполняются конкретные облачные вычисления, как правило, скрыты и даже, возможно, неизвестны или непредсказуемы, заранее выполнить требования к производительности может оказаться невозможным. Кроме того, [26] показали, что разные провайдеры могут легко показать очень разные профили производительности. Облачные вычисления больше не вызывают былого ажиотажа и, безусловно, являются серьезной альтернативой поддержанию огромных локальных инфраструктур, однако пространство для их улучшения все еще сохраняется.

Другой важный класс распределенных систем встречается в организациях, которые столкнулись с множеством сетевых приложений, но для которых совместимость оказалась болезненным опытом. Многие из существующих промежуточных решений являются результатом работы с инфраструктурой, в которой было проще интегрировать приложения в корпоративную информационную систему [16].

Можно выделить несколько уровней, на которых может происходить интеграция. Во многих случаях сетевое приложение просто состоит из сервера, на котором выполняется это приложение (часто включающее базу данных) и делающее его доступным для удаленных программ, называемых клиентами. Такие клиенты отправляют запрос на сервер для выполнения определенной операции, после чего ответ отправляется обратно. Интеграция на самом низком уровне позволяет клиентам объединять несколько запросов, возможно, для разных серверов, в один больший запрос и выполнять его как распределенную транзакцию. Основная идея заключается в том, что выполняются или все запросы, или не один вообще.

Поскольку приложения стали более сложными и постепенно были разделены на независимые компоненты (особенно отличающие компоненты базы данных от компонентов обработки), стало ясно, что интеграция также должна происходить, позволяя приложениям взаимодействовать друг с другом напрямую. Это привело к созданию огромной отрасли, которая концентрируется на интеграции корпоративных приложений (EAI).

1.3 Стили архитектур распределенных систем

Исследования в области архитектуры программного обеспечения значительно продвинулись, и в настоящее время общепризнанно, что разработка или внедрение архитектуры имеет решающее значение для успешной разработки больших программных систем.

Очень важно отметить понятие архитектурного стиля. Такой стиль формулируется с точки зрения компонентов, способа, которым компоненты связаны друг с другом, обмена данными между компонентами и, наконец, как эти элементы совместно настраиваются в систему. Компонент – это модульная единица с четко определенными необходимыми и предоставленными интерфейсами, которые могут быть заменены в его среде [27]. Важно, чтобы компонент был заменен, и, в частности, пока система продолжает работать. Это связано с тем, что во многих случаях невозможно отключить систему для технического обслуживания. В лучшем случае, только его части могут быть временно выведены из строя. Замена компонента может быть выполнена, только если его интерфейсы остаются нетронутыми.

Несколько сложнее понять концепцию соединителя, который обычно описывается как механизм, обеспечивающий координацию или взаимодействие между компонентами [28, 29]. Например, соединитель может быть образован средствами (удаленных) вызовов процедур, передачи сообщений или потоковой передачи данных. Другими словами, соединитель обеспечивает поток управления и данных между компонентами. Используя

компоненты и соединители, мы можем прийти к различным конфигурациям, которые, в свою очередь, были классифицированы по архитектурным стилям. К настоящему времени определено несколько стилей, из которых наиболее важными для распределенных систем являются:

– многоуровневая архитектура;
– объектно-ориентированная архитектура;
– ресурсно-центрированная архитектура;
– событийная архитектура.

Основная идея многоуровневого стиля проста: компоненты располагаются на нескольких уровнях, где компонент на уровне L_j может выполнить обратный вызов к компоненту на нижнем уровне L_i ($i<j$) и, как правило, ожидает ответа. Только в исключительных случаях возможен переход на компонент более высокого уровня. Три распространенных случая показаны на рисунке 1.5.

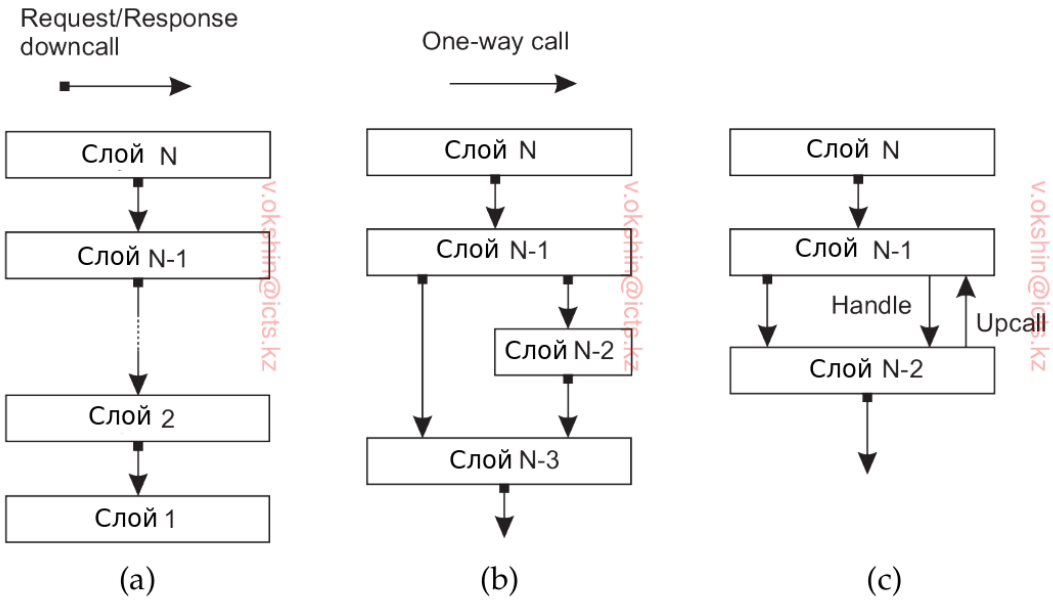

Рисунок 1.5 – а – Чистая многоуровневая организация, б – смешанная многоуровневая организация, в – многоуровневая организация с вызовами

На рисунке 1.5.а показана стандартная организация, в которой выполняются только нисходящие вызовы до следующего нижнего уровня. Эта организация обычно развертывается в случае сетевого взаимодействия.

Во многих ситуациях может использоваться струткту́ра, показанная на рисунке 1.5.б. Рассмотрим, например, приложение А, которое использует библиотеку L_{OS} для взаимодействия с операционной системой. В то же время приложение использует специализированную математическую библиотеку L_{math}, которая была также реализована с использованием L_{OS}. В этом случае, ссылаясь на рисунок 1.5.б, А реализуется на уровне N-1, L_{math} на уровне N-2 и L_{OS}, которая является общей для них обоих, на уровне N-3.

Наконец, особая ситуация показана на рисунке 1.5.в. В некоторых случаях удобно, чтобы нижний уровень выполнял вызов до следующего более

высокого уровня. Типичный пример – когда операционная система сигнализирует о возникновении события, для чего она вызывает пользовательскую операцию, для которой приложение ранее передавало ссылку (обычно называемую дескриптором).

В объектно-ориентированных архитектурах используется гораздо более свободная организация, показанная на рисунке 1.6. По сути, каждый объект соответствует понятию компонент, и эти компоненты связаны через механизм вызова процедуры. В случае распределенных систем вызов процедуры также может происходить по сети, то есть вызывающий объект не обязательно должен выполняться на той же машине, что и вызываемый объект.

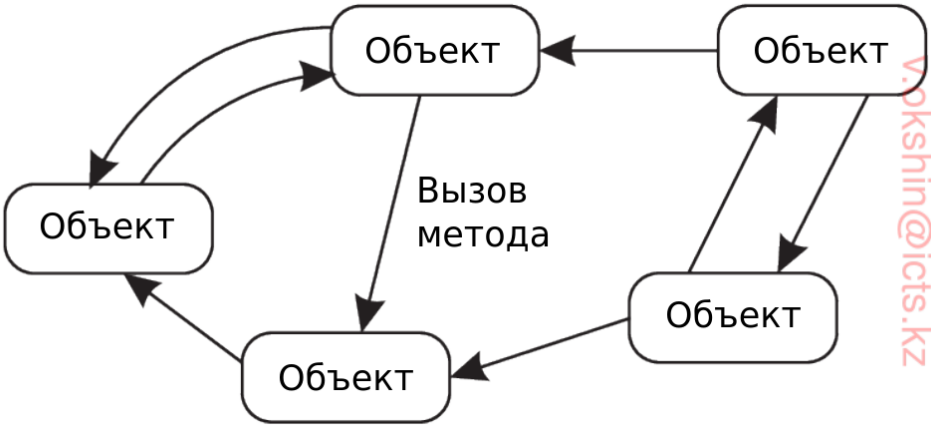

Рисунок 1.6 – Объектный архитектурный стиль

Объектно-ориентированные архитектуры привлекательны тем, что обеспечивают естественный способ инкапсуляции данных (называемых состоянием объекта) и операций, которые можно выполнять над этими данными (которые называются методами объекта) в единый объект. Интерфейс, предлагаемый объектом, скрывает детали реализации, по сути, это означает, что, в принципе, можно считать объект полностью независимым от его среды. Как и в случае с компонентами, это также означает, что если интерфейс четко определен и оставлен без изменений, объект должен быть заменен на объект, имеющий точно такой же интерфейс.

Такое разделение между интерфейсами и объектами, реализующими эти интерфейсы, позволяет размещать интерфейс на одной машине, в то время как сам объект находится на другой машине. Эта организация, показанная на рисунке 1.7, обычно называется распределенным объектом.

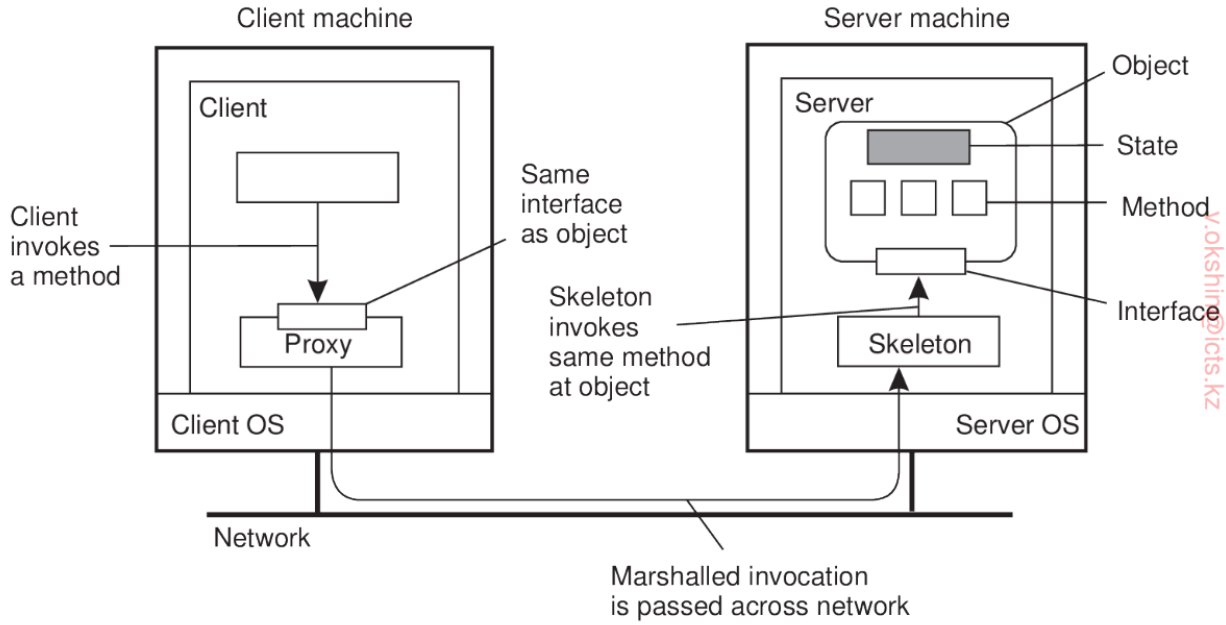

Рисунок 1.7 – Общая организация удаленного объекта с прокси на стороне клиента

Характерной, но несколько противоречивой особенностью большинства распределенных объектов является то, что их состояние не распределено: оно находится на одной машине. Только интерфейсы, реализованные объектом, становятся доступными на других машинах. Такие объекты также называются удаленными объектами. В общем распределенном объекте само состояние может быть физически распределено по нескольким машинам, но это распределение в то же время скрыто от клиентов за интерфейсами объекта.

По мере того, как все большее число сервисов становится доступным через Интернет, а развитие распределенных систем с помощью композиции сервисов становится все более необходимым, исследователи переосмысливают архитектуру распределенных систем функционирующих на основе Интернета. Одна из проблем с составом сервиса заключается в том, что соединение различных компонентов может легко превратиться в «интеграционный кошмар».

В качестве альтернативы можно также рассматривать распределенную систему как огромную коллекцию ресурсов, которые индивидуально управляются компонентами. Ресурсы могут быть добавлены или удалены приложениями, а также могут быть извлечены или изменены. Этот подход в настоящее время получил широкое распространение в Интернете и известен как передача состояния представительства (REST) [30]. Существует четыре ключевых характеристики того, что известно как архитектуры RESTful [31]:

– ресурсы идентифицируются по единой схеме именования;

– все сервисы предлагают одинаковый интерфейс, состоящий максимум из четырех операций;

– сообщения, отправляемые в службу или из нее, полностью описываются;

– после выполнения операции в службе этот компонент забывает все о вызывающем.

Поскольку системы продолжают расти, становится важным иметь архитектуру, в которой зависимости между процессами становятся настолько свободными, насколько это возможно. Большой класс распределенных систем принял архитектуру, в которой существует сильное разделение между обработкой и координацией. Идея состоит в том, чтобы рассматривать систему как совокупность автономно работающих процессов. В этой модели координация включает в себя взаимодействие между процессами. Она образует «клей», который связывает действия, выполняемые процессами, в единое целое [32].

1.4 Требования, предъявляемые к распределенным системами

Для полноценного функционирования распределенная система должна соответствовать ряду требований. К этим требованиям относятся: безопасность, открытость, прозрачность, надежность [38]. Рассмотрим их подробнее.

Прозрачность распределенной системы. Смысл прозрачности заключен в том, что система для пользователя должна представляться единым и целым объектом. Пользователь не должен знать внутреннюю структуру распределенной системы и количество компонентов в ней. Иными словами, протокол взаимодействия пользователя и распределенной системы не должен меняться в зависимости от состояния каждой отдельной ее составляющей. Тут стоит рассмотреть некоторые частные примеры прозрачности:

– прозрачность расположения узлов и данных. В распределенных системах прозрачность расположения заключается в том, что пользователю неизвестно где расположены необходимые ему ресурсы. Файлы могут перемещаться на различные узлы распределенной системы, например, если на одном узле распределенной системы произошел сбой, и данные были восстановлены на другом узле распределенной системы, то в таком случае, эти изменения не должны никоим образом отразиться на взаимодействии пользователя и системы. Например, в распределенных информационных файловых системах, пользователь должен видеть лишь единое файловое пространство, при том, что данные могут располагаться физически на разных серверах;

– принцип прозрачности доступа занимает одну из важнейших ролей в распределенных системах. Его смысл заключается в том, что различия доступа и предоставления данных должны быть скрыты;

– прозрачность параллелизма доступа заключается в том, что в системе одновременно может присутствовать большое количество пользователей. Эти пользователи должны иметь параллельный доступ к общим данным. При этом тот факт, что данные в этот момент используются кем-то еще, должен быть скрыт от пользователей;

– прозрачность резервирования. Резервирование используется для обеспечения сохранности информации, т. е. создаются несколько копий

файлов и размещаются на нескольких файловых серверах. Однако, при работе с данными пользователь не должен принимать участие в процессе резервирования.

Как службы, так и приложения предоставляют ресурсы, которые могут совместно использоваться клиентами в распределенной системе. Поэтому существует вероятность того, что несколько клиентов попытаются получить доступ к общему ресурсу одновременно. Объект, представляющий общий ресурс в распределенной системе, должен отвечать за обеспечение его правильной работы в параллельной среде. Это относится не только к серверам, но и к объектам в приложениях. Поэтому любой программист, принимающий реализацию объекта, который не был предназначен для использования в распределенной системе, должен сделать все необходимое, чтобы сделать его безопасным в параллельной среде.

Открытость распределенной системы. Если ранние распределенные системы были призваны решать конкретные задачи и при этом зачастую создавались лишь в пределах одной организации, то современные распределенные системы отличаются тем, что могут быть использованы для решения большого спектра задач и создаются более открытыми. Благодаря развитию систем передач данных, аппаратных средств и информационных технологий в целом, снимается географическое ограничение при создании распределенной системы. Под открытостью распределенных систем понимается возможность взаимодействия с другими открытыми системами. То есть всем компоненты распределенной системы должны использовать стандартные общедоступные технологии. Такой подход позволит поддерживать взаимодействие распределенной системы с другими системами, а также будет поддерживать принцип прозрачности доступа. Открытость системы может быть достигнута с помощью: языков программирования, аппаратных платформ, программного обеспечения.

Обобщая вышесказанное, можно сказать что, открытость компьютерной системы – это характеристика, которая определяет, может ли система быть расширена и повторно реализована различными способами. Открытость распределенных систем определяется прежде всего тем, в какой степени новые службы совместного использования ресурсов могут быть добавлены и доступны для использования различными клиентскими программами.

Многие информационные ресурсы, предоставляемые и обслуживаемые в распределенных системах, имеют высокую внутреннюю ценность для своих пользователей. Поэтому их безопасность имеет большое значение. Безопасность информационных ресурсов состоит из трех компонентов: конфиденциальности, целостности и доступности. Применение политики безопасности к взаимозависимой системе – большая проблема в задаче проектирования распределенной системы. Поскольку распределенные системы имеют дело с конфиденциальными данными и информацией, система должна обеспечивать достаточный уровень безопасности и конфиденциальности. Защита активов распределенной системы, включая базовые ресурсы, хранилище, коммуникацию и ввод-вывод пользовательского

интерфейса, а также составляющие более высокого уровня этих ресурсов, такие как процессы, файлы, сообщения, окна отображения и более сложные объекты, являются важными задачами в создании распределенной системы. Безопасность распределенной системы является, в общем случае, совокупностью 3 факторов:

– обеспечение конфиденциальности данных и ресурсов;

– обеспечение конфиденциальности доступа к ресурсам для множества пользователей;

– обеспечение целостности ресурсов и данных.

Необходимость создания распределенных систем, которые обеспечивают необходимую безопасность данных и всей структуры распределенной системы, возникает повсеместно. Многие вопросы безопасности могут быть решены на уровне отдельных узлов распределенной системы, например, путем установки firewall-ов и антивирусного программного обеспечения на отдельные узлы системы, введением политики аутентификации пользователей и другими методами. Но в силу особенности архитектуры большинства распределенных систем, данный подход не всегда является эффективным. Программное обеспечение не всегда может обеспечить необходимую конфиденциальность данных в распределенной системе. Например, программное обеспечение не всегда может полноценную защиту от MIMT и DDOS атак на распределенную сеть. Зачастую методы защиты от подобных атак не всегда являются приемлемыми для узлов вычислительной сети. Важным показателем, при организации защиты распределенной системы, является уровень доступности системы. Уровень доступности распределенной системы определяется не только доступностью ресурса в момент времени t, но и принципами организации защиты распределенной системы, так как большинство программных средств, обеспечивающие защиту от атак, направлены на отказ в обслуживании. Большое внимание данному вопросу уделяется многими производителями антивирусного программного обеспечения.

Качество обслуживания: как определить приемлемый уровень качества обслуживания, предоставляемого пользователям. Качество обслуживания в значительной степени зависит от ресурсов, которые должны быть выделены процессам в системе, распределения ресурсов, аппаратного обеспечения, адаптивности системы, сети и т.д. Хорошая производительность, доступность и надежность необходимы для обеспечения хорошего качества обслуживания. После того как пользователи получают необходимую им функциональность, например файловую службу в распределенной системе, можно перейти к вопросу о качестве предоставляемой услуги. Основными не функциональными свойствами систем, влияющими на качество обслуживания клиентов и пользователей, являются надежность, безопасность и производительность. Адаптивность к изменяющимся конфигурациям систем и доступность ресурсов являются еще одним важным аспектом качества обслуживания.

Идентификация ресурсов: ресурсы распределяются между различными компьютерами, и для точной привязки ресурсов должна быть разработана правильная схема именования.

Коммуникации: распределенные системы стали более эффективными с появлением Интернета, но существуют определенные требования к производительности, надежности и т.д. Необходимо использовать эффективные подходы к коммуникации.

Архитектура программного обеспечения отражает функциональность приложения, распределенную между логическими компонентами и процессорами. Выбор правильной архитектуры для приложения необходим для повышения качества обслуживания.

Обработка отказов компонентов системы. Компьютерные системы иногда могут выходить из строя. При возникновении неисправностей в аппаратном или программном обеспечении программы могут выдавать неверные результаты или останавливаться до завершения запланированных вычислений. Сбои в распределенной системе носят частичный характер – то есть некоторые компоненты выходят из строя, а другие продолжают функционировать. Поэтому обработка отказов особенно сложна.

Одной из первоначальных целей построения распределенных систем было сделать их более надежными. Идея заключается в том, что если машина выходит из строя, то какая-то другая машина берет на себя ее работу. Высоконадежная система должна быть очень доступной, но этого недостаточно. Данные, доверенные системе, не должны быть потеряны или искажены каким-либо образом, и если файлы хранятся избыточно на нескольких серверах, все копии должны быть согласованы. Как правило, чем больше копий хранится, тем лучше доступность, но тем больше вероятность того, что они будут несогласованными, особенно если обновления происходят часто.

В связи с появлением новых методов и алгоритмов, требовательных к вычислительным ресурсам и, самое главное, к ресурсам времени, необходимость в доступности распределенных систем в момент времени t становится крайне актуальным. Основным показателем, определяющим надежность всей распределенной системы, является отказоустойчивость. Отказоустойчивость это важнейшее свойство вычислительной системы, которое заключается в возможности продолжения действий, заданных программой, после возникновения неисправностей.

Несмотря на все достоинства распределенных систем по сравнению с традиционными централизованными системами (распределенные системы обеспечивают значительно меньшую стоимость развертывания и простоту реализации), распределенные системы имеют и ряд существенных недостатков. Основными проблемами распределенных систем по сравнению с традиционными системами являются:

– проблемы администрирования системы;

– проблемы ограниченности масштабируемости распределенной системы;

– проблемы переносимости программного обеспечения.

Проблемы администрирования системы включают проблемы : проблемы балансирования нагрузки на узлы системы; проблемы восстановления данных в случае возникновения ошибок. Фрагментация ресурсов в распределенных системах предписывает необходимость создания гибких настраиваемых средств администрирования. Так как в глобально распределенных системах администрирование должно происходить в автоматическом режиме, то в связи с этим возникают следующие основные проблемы администрирования распределенных систем:

– балансировка нагрузки на узлы системы;

– восстановление данных в случае возникновения ошибки;

– сбор статистики с узлов системы;

– обновление программного обеспечения на узлах системы в автоматическом режиме.

Стоит понимать, что данный список является обобщенным, т.е. для каждой конкретной распределенной системы возможно возникновение проблем, которые здесь не перечислены. Но данные проблемы являются наиболее часто встречающимися на практике и заслуживают особого внимания при проектировании распределенной системы. Последние две проблемы хорошо изучены, и на рынке программного обеспечения распределенных систем существует множество разработанных программных средств, которые обеспечивают как сбор статистики, так и обновление программного обеспечения. Научный же интерес представляют методы балансирования нагрузки на узлы системы и методы восстановления данных в случае возникновения ошибок. В силу специфики распределенных систем, а также гетерогенности оборудования и архитектуры распределенных систем, не существует единого метода проектирования, который обеспечивал бы решение этих проблем.

Управление ресурсами: в распределенных системах объекты, состоящие из ресурсов, расположены в разных местах. Маршрутизация – это проблема на сетевом уровне распределенной системы и на уровне приложений. Управление ресурсами в распределенной системе будет взаимодействовать с ее гетерогенной природой.

Гетерогенность. Распределенные системы позволяют пользователям получать доступ к службам и запускать приложения на разнородном наборе компьютеров и сетей. Распределенные системы могут состоять из множества различных видов сетей, их различия маскируются тем фактом, что все компьютеры, подключенные к ним, используют интернет-протоколы для связи друг с другом. Например. компьютер, подключенный к сети Ethernet, имеет реализацию интернет-протоколов через сеть Ethernet, в то время как компьютер в сети другого типа будет нуждаться в реализации интернет-протоколов для этой сети.

1.5 Моделирование систем и угроз

Термин «надежность» широко используется в самых разных контекстах и зачастую подразумевает совершенно разные значения. В контексте распределенных систем, надежность означает способность распределенной системы обеспечить корректную работу сервисов для пользователей, несмотря на такие угрозы как: ошибки в программном обеспечении, поломки оборудования и злонамеренные атаки.

Для того чтобы понять, насколько надежна распределенная система, необходимо четко смоделировать как систему, так и угрозы для нее. Также необходимо определить общие свойства надежных распределенных систем и метрики для оценки их надежности.

Каждая система предназначена для предоставления набора сервисов для пользователей. Каждый сервис обладает неким интерфейсом посредством которого пользователь может взаимодействовать с системой. То, что система должна делать для каждой службы, определяется как набор функций в соответствии с функциональной спецификацией системы. Статус системы определен ее состоянием. Охарактеризовать состояние системы на практике оказывается достаточно сложным. Система может состоять из нескольких процессов, охватывающих один или несколько узлов, в свою очередь, каждый процесс может состоять из одного или нескольких потоков. Поэтому состояние системы формируется из общего состояния процессов и потоков в системе. Состояние процесса обычно исходит из значений его регистров, стека, файловых дескрипторов и состояния ядра. Часть состояния может быть видна пользователям системы посредством данных содержащихся в ответах на пользовательский запросы. Такое состояние называется внешним состоянием и обычно является абстрактным состоянием, определенным в функциональной спецификации системы. Другая часть состояния, которая не видна пользователям, называется внутренним состоянием. Систему можно восстановить до того места, где она была до сбоя, если ее состояние было зафиксировано и не потеряно из-за сбоя (например, если состояние сериализовано и записано в стабильное хранилище).

С точки зрения структуры, система состоит из одного или нескольких компонентов (таких как узлы или процессы), и система всегда имеет границу, которая отделяет систему от ее среды. Здесь среда относится ко всем другим системам, с которыми взаимодействует текущая система. Обратите внимание, что то, что здесь называется системой, всегда относительно по отношению к текущему контексту. Компонент в (большей) системе сам по себе является системой, когда мы хотим изучить его поведение, и он, в свою очередь, может иметь свои собственные внутренние структуры.

Предоставляет ли система услуги правильным образом или нет, оценивается по тому, выполняет ли система функции, определенные в ее функциональной спецификации. Когда система не функционирует в соответствии с ее функциональной спецификацией, мы говорим, что произошел сбой службы (или просто сбой). Отказ системы вызван частью неправильных значений в ее состоянии, то есть ошибками в ее состоянии. Мы предполагаем, что ошибки вызваны некоторыми неисправностями [6].

Следовательно, угрозы надежности системы моделируются как различные сбои.

Сбой может не всегда проявляться и вызывать ошибку. В частности, программный дефект (часто называемый программной ошибкой) может не быть обнаружен до тех пор, пока код, содержащий дефект, не будет выполнен при выполнении определенного условия. Например, если совместно используемая переменная не защищена блокировкой в многопоточном приложении, сбой (часто называемый условием гонки) не проявляется, если только два или более потоков не пытаются обновить совместно используемую переменную одновременно. В качестве другого примера, если при доступе к массиву отсутствует проверка границ, ошибка не будет отображаться, пока процесс не попытается получить доступ к массиву с индексом вне границ.

Когда ошибка не проявляется, мы говорим, что ошибка неактивна. Если же определенное условие выполняется, неисправность активируется.

При появлении сбоя изначально будет вызываться ошибка в компоненте, который охватывает дефектную область (в программном коде). Когда компонент взаимодействует с другими компонентами системы, ошибка распространяется на другие компоненты. Когда ошибки распространяются на интерфейс системы и услуги, предоставляемые пользователю, отклоняются от спецификации, происходит сбой службы. Из-за рекурсивного характера общего состава системы, отказ одной системы может вызвать сбой в более крупной системе, когда первая составляет компонент второй, как показано на рисунке 1.8. Такая связь между сбоем, ошибкой и отказом упоминается как «цепь угроз» в [2].

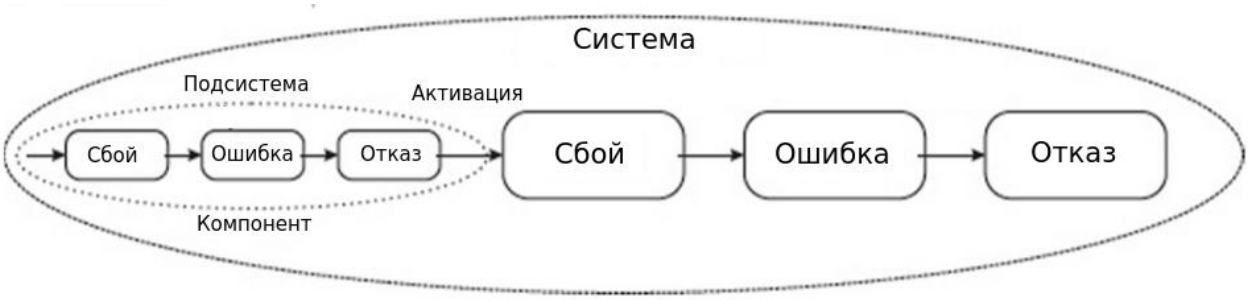

Рисунок 1.8 – Пример цепи угроз с двумя рекурсивными уровнями

Конечно, не все сбои могут быть проанализированы с помощью вышеуказанной цепочки угроз. Например, отключение питания всей системы немедленно приведет к отказу системы.

Неисправности могут быть классифицированы на основе различных критериев: источник неисправности,

Основываясь на источнике неисправностей, неисправности можно классифицировать как:

– аппаратные сбои, если они вызваны отказом аппаратных компонентов, таких как перебои в подаче электроэнергии, отказы жесткого диска, неисправные микросхемы памяти и т. д.;

– сбои программного обеспечения, если они вызваны ошибками в программном обеспечении, такими как состояние гонки и проверка без ограничений для массивов и т. д.;

– ошибки оператора, если ошибки вызваны оператором системы, например, неправильная конфигурация, неправильные процедуры обновления и т. д.

На основании присутствия умышленности, неисправности можно классифицировать как:

– не злонамеренные ошибки, если ошибки не вызваны человеком со злым умыслом. Например, естественные сбои оборудования и некоторые остающиеся программные ошибки не являются злонамеренными;

– злонамеренные ошибки, если ошибки вызваны лицом с намерением нанести вред системе, например, чтобы помешать работе пользователей или поставить под угрозу целостность сервиса. Злонамеренные ошибки часто упоминаются как «проблема византийских генералов» [5].

По продолжительности сбоев, сбои можно классифицировать как:

– мгновенные сбои – такие сбои активируется на мгновение и снова становятся бездействующими. Например, состояние гонки часто может отображаться как временная ошибка, потому что, если потоки прекращают одновременный доступ к общей переменной, то ошибка исчезает.

– перманентные неисправности – однажды активированная неисправность, остающаяся активированной до тех пор, пока неисправный компонент не будет отремонтирован или не будет устранен источник неисправности. Например, отключение питания считается постоянной неисправностью, поскольку, если питание не будет восстановлено, компьютерная система останется отключенной.

На основании того, как отказ одного компонента влияет на другие компоненты в системе, отказы можно классифицировать как:

– с повреждением данных – происходит если значения, переданные другим компонентам, неверны из-за сбоев. Неисправный компонент может передавать одинаково неправильные значения другим компонентам или возвращать разные значения для различных компонентов, с которыми он взаимодействует;

– временной сбой – если неисправный компонент либо возвращает ответ слишком рано, либо слишком поздно, после получения запроса от другого компонента. Крайний случай – это когда неисправный компонент вообще перестает отвечать (то есть, для возврата ответа требуется бесконечное количество времени), например, когда компонент отказывает или зависает из-за бесконечного цикла или тупика.

Основывая на том, является ли неисправность воспроизводимой или детерминированной, неисправности (в первую очередь неисправности программного обеспечения) можно классифицировать как:

– воспроизводимые / детерминированные ошибки. Ошибка происходит детерминировано и может быть легко воспроизведена. Доступ к нулевому указателю является примером детерминированной ошибки, которая часто

приводит к поломке системы. Этот тип неисправностей может быть легко идентифицирован и устранен.

– недетерминированные ошибки. Ошибка, по-видимому, бывает недетерминированной и трудно воспроизводимой. Например, если ошибка вызвана определенным чередованием нескольких потоков, когда они получают доступ к некоторой общей переменной, то такую ошибку будет трудно воспроизвести. Этот тип ошибок программного обеспечения также называют «Гейзенбагс», чтобы подчеркнуть их неопределенность.

Учитывая количество ошибок в системе, их можно классифицировать на основе их взаимосвязи:

– независимые неисправности, если нет причинно-следственной связи между неисправностями, например, неисправностью A и неисправностью B, B не вызвано A, а A не вызвано B.

– коррелированные неисправности, если неисправности причинно связаны, например, с учетом неисправности A и неисправности B, либо B вызвано A, либо A вызвано B. Если несколько компонентов выходят из строя по общей причине, отказы называются отказом общего режима.

При выходе системы из строя, желательно избегать катастрофических последствий, таких как гибель людей. Последствия отказа системы могут быть смягчены путем включения в систему механизма обеспечения надежности, такого что при сбое система переставала отвечать на запросы (такие системы называются системами аварийного останова), если это невозможно, система должна возвращать неправильные значения вместо несовместимых значений для всех компонентов, с которыми она может взаимодействовать. Если сбой системы не причиняет большого вреда ни человеческой жизни, ни окружающей среде, такая система называется отказоустойчивой. Обычно отказоустойчивая система обладает набором безопасных состояний. Когда отказоустойчивая система больше не может работать в соответствии с ее спецификацией из-за неисправностей, она может перейти в одно из предопределенных безопасных состояний при сбое. Например, компьютерная система, которая используется для управления атомной электростанцией, должна быть отказоустойчивой системой.

Возможно, это может показаться парадоксальным, но часто желательно, чтобы система немедленно прекратила свою работу, когда она находится в состоянии ошибки или сталкивается с неожиданным состоянием. Практика разработки программного обеспечения, обеспечивающая такое поведение, называется быстрый сбой [8]. Преимущества практики отказоустойчивости состоят в том, что она позволяет заблаговременно обнаруживать ошибки программного обеспечения и диагностировать ошибки. Когда ошибка распространяется на многие другие компоненты, гораздо труднее определить источник проблемы.

1.6 Средства для достижения надежности

Существует три основных подхода к повышению надежности распределенных систем:

– предотвращение сбоев: создание и использование высококачественных программных компонентов и оборудования, которые менее подвержены сбоям;

– обнаружение и диагностика неисправностей: несмотря на то, что обнаружение неисправностей тривиально, компоненты в практической системе могут выходить из строя различными способами, отличными от неисправности, и, если не обнаружены, целостность системы не может быть гарантирована;

– отказоустойчивость: система способна восстанавливаться после различных отказов без прерывания обслуживания, если система использует достаточную избыточность, чтобы система могла маскировать сбои части своих компонентов, или с минимальным прерыванием обслуживания, если система использует менее дорогостоящие средства обеспечения надежности, такие как логирование и контрольные точки.

Для программных компонентов предотвращение сбоев направлено на обеспечение правильной спецификации проекта и правильной реализации до выпуска распределенной системы. Эта цель может быть достигнута путем использования стандартных методов разработки программного обеспечения, например:

– более строгий дизайн программного обеспечения с использованием таких методов, как формальные методы. Формальные методы требуют использования формального языка для облегчения проверки спецификации;

– более тщательное тестирование программного обеспечения для выявления и устранения ошибок в программном обеспечении из-за остаточных недостатков проекта и внедренных во время внедрения;

– для некоторых приложений может быть нецелесообразно использовать формальные методы, и в этом случае целесообразно разрабатывать для тестируемости [2], например, широко используя модульное тестирование, которое доступно во многих современных языках программирования, таких как Java и C#.

Обнаружение неисправностей является важным шагом в обеспечении надежности системы. Выявление сбоев относительно тривиально, например, можно периодически проверять каждый компонент, чтобы проверить его работоспособность. Если ответ не получен после нескольких последовательных проверок, то компонент может быть объявлен как аварийный. Однако компоненты в системе могут выходить из строя по-разному, и они могут совершенно различным образом реагировать на каждую проверку после сбоя. Обнаружить такие неисправности является нетривиальной задачей, особенно в большой распределенной системе. Диагностика необходима для определения того, что неисправность действительно произошла, и для локализации источника неисправности (то есть, точно определить неисправный компонент). Для этого моделируется распределенная система, и часто используются сложные статистические инструменты [4].

Большой прогресс был достигнут в разработке современных языков программирования для включения некоторых форм обнаружения и обработки программных ошибок, таких как неожиданный ввод или состояние. Наиболее ярким примером является обработка исключений. Блок кода может быть заключен в конструкцию try-catch. Если во время выполнения кода возникает ошибка, блок catch будет выполнен автоматически. Исключения также могут распространяться вверх по вызывающей цепочке. Если возникает исключение, и оно не обрабатывается никаким кодом, предоставленным разработчиком, среда выполнения языка обычно завершает процесс.

Метод блока восстановления, разработанный для обеспечения отказоустойчивости программного обеспечения [7], может рассматриваться как расширение механизма обработки исключений в языке программирования. Важным этапом в блоках восстановления является приемочное тестирование, которое является формой обнаружения неисправностей. Разработчик должен предоставить приемочный тест для каждого модуля системы. Когда приемочный тест не пройден, обнаруживается программный сбой. Затем выполняется альтернативный блок кода, после чего приемочный тест оценивается снова. Может быть предусмотрено несколько альтернативных блоков кода для повышения надежности системы.

Как только неисправность обнаружена и локализована, она должна быть изолирована и удалена из системы. Впоследствии неисправный компонент либо ремонтируется, либо заменяется. Отремонтированный или замененный компонент может быть повторно принят в систему. Чтобы выполнить эти шаги, систему часто необходимо перенастроить. В распределенной системе часто необходимо иметь представление о членстве, то есть каждый компонент осведомлен о списке компонентов, которые считаются частью системы, и их ролях. Когда неисправный компонент удаляется из системы, выполняется реконфигурация, и формируется новое членство, исключая неисправный компонент. Когда компонент отремонтирован или заменен, и повторно принят в систему, он снова становится частью членства.

Особый случай устранения неисправностей – это исправления программного обеспечения и обновления. Ошибки и уязвимости в программном обеспечении могут быть устранены путем обновления программного обеспечения при исправлении исходной системы. Практически все современные операционные системы и пакеты программного обеспечения включают возможность обновления программного обеспечения.

Одного только надежного программного обеспечения обычно недостаточно для обеспечения высокой надежности из-за возможности аппаратных сбоев. Если в распределенной системе нет контрольных точек, простой перезапуск системы после сбоя не приведет к автоматическому восстановлению ее состояния до того состояния, которое было до сбоя. Следовательно, необходимы методы отказоустойчивости для повышения надежности распределенных систем до уровня выше.

Существуют разные методы отказоустойчивости, которые могут использоваться для удовлетворения различных требований к надежности. Для приложений, которым требуется высокая доступность, но не обязательно высокая надежность может быть достаточно журналирования и создания контрольных точек, которые требуют минимальных затрат времени выполнения и используют минимальные дополнительные ресурсы. Более требовательные приложения могут использовать вычислительные методы ориентированные на восстановление. Оба типа отказоустойчивых методов основаны на восстановлении с откатом. После перезапуска отказавшей системы самое последнее правильное состояние (называемое контрольной точкой) системы находится в журнале, и система восстанавливается в это правильное состояние.

Создание точек восстановления и журналирование являются наиболее важными методами для достижения надежности в распределенных системах [7]. Эти методы позволяют обеспечить отказоустойчивость, которая относительно проста в реализации и требует минимальное количество времени на выполнение. Несмотря на то, что некоторая информация может быть потеряна (при использовании только контрольных точек) при возникновении сбоя и во время восстановления после сбоя, как правило, больше, чем при более сложных подходах отказоустойчивости, этого может быть достаточно для многих приложений. Кроме того, это применяется на всех уровнях механизма обеспечения надежности.

Контрольные точки распределенной системы представляют собой снимки состояния системы [7]. Если контрольная точка доступна после сбоя системы, ее можно использовать для восстановления системы до состояния, когда контрольная точка была взята. Контрольная точка – это периодическое действие, при котором копируется состояние системы и сохраняется в стабильном хранилище, которое не подвержено возможным ошибкам.

Чтобы восстановить систему до того момента, когда она вышла из строя, в дополнение к периодическим контрольным точкам также должна собираться и другая информация о состоянии системы. Обычно все поступающие запросы в систему регистрируются. Другие недетерминированные события могут также регистрироваться, чтобы гарантировать правильное восстановление.

Контрольные точки и ведение журнала предоставляют форму восстановления с откатом [7], поскольку они могут восстановить систему до состояния до сбоя. Напротив, существуют другие подходы, которые выполняют восстановление с повтором транзакций, то есть сбойный процесс может быть восстановлен до текущего состояния путем включения избыточности процесса в систему. Однако протоколы восстановления с повтором транзакций обычно значительно увеличивают время выполнения и требуют больше физических ресурсов.

Пример сценария восстановления отката показан на рисунке 1.9. В случае сбоя системы требуется некоторое время для обнаружения сбоя. Затем система перезагружается, и самая последняя контрольная точка в журнале

используется для восстановления системы до этой контрольной точки. Если есть зарегистрированные запросы, эти запросы повторно выполняются системой, после чего восстановление завершается. Затем система возобновляет обработку новых запросов.

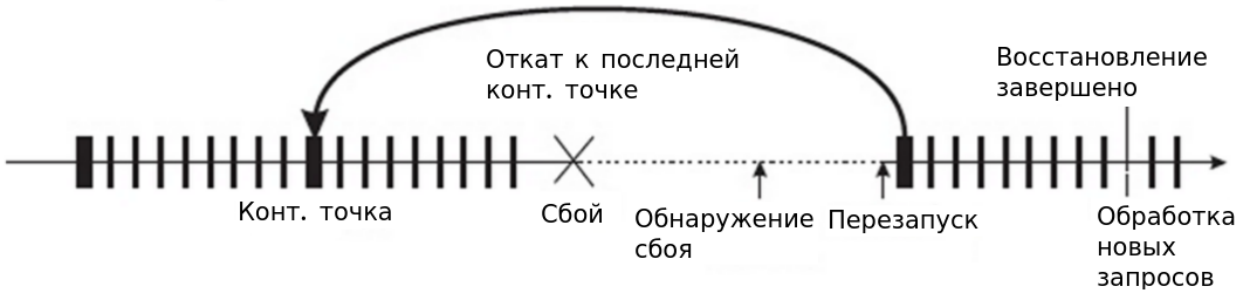

Рисунок 1.9 – Восстановление состояния происходит, используя контрольные точки и, как правило, логируя запросы полученные в период от появления сбоя до его обнаружения

Для распределенной системы, которая требует высокой надежности, то есть непрерывного правильного обслуживания, должны использоваться избыточные экземпляры системы, чтобы система могла продолжать работать правильно, даже если часть избыточных копий (называемых репликами) выходит из строя. Использование избыточных экземпляров также позволяет допускать злонамеренные сбои при условии, что реплики отказывают независимо. Когда неисправная реплика восстанавливается, ее можно включить обратно в систему, приведя ее состояние в соответствие с текущим состоянием других реплик. Эта стратегия восстановления называется восстановлением с повтором транзакций.

Пример сценария восстановления с повтором транзакций показан на рисунке 1.10. При обнаружении сбоя реплики реплика перезапускается (возможно, после ремонта). Для повторной отправки перезапущенной реплики в систему нефункционирующая реплика берет контрольную точку состояния и переносит эту точку к себе. Перезапущенная реплика может откатить свое состояние, используя полученную контрольную точку, которая представляет последнее состояние системы.

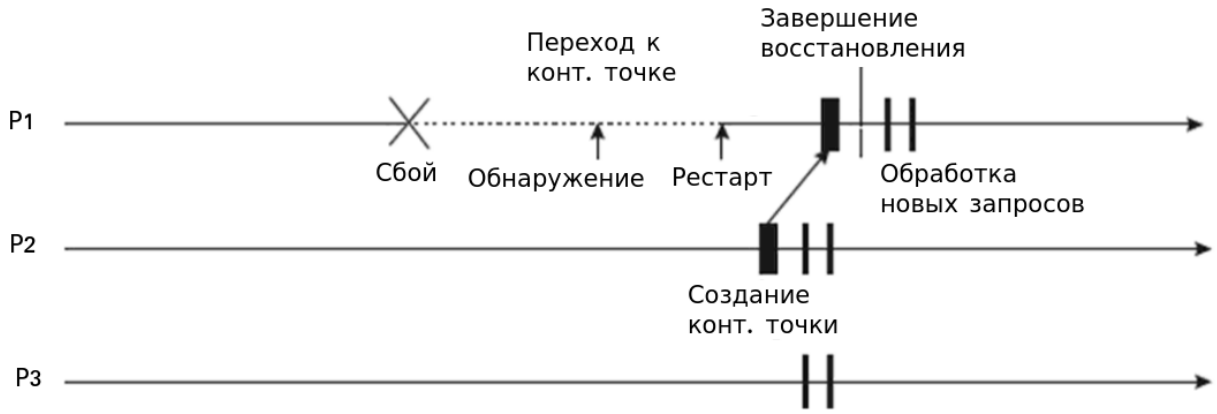

Рисунок 1.10 – При наличии избыточных экземпляров в системе сбой реплики может быть замаскирован, и система продолжит предоставлять услуги своим пользователям

Чтобы избежать сбоев в общего режима (т. е. коррелированных сбоев), будет полезно, если каждая реплика может выполнять разные версии системного кода. Эта стратегия называется n-версией программирования [9]. Преобразование программы также может быть использовано для получения разнообразных реплик с более низкой стоимостью разработки программного обеспечения [10]. Специальная форма n-версии программирования появляется в методе восстановления блока для обеспечения отказоустойчивости программного обеспечения [7]. Вместо использования разных версий программного обеспечения в разных репликах каждый модуль системы оснащен основной версией и одной или несколькими альтернативными версиями. В конце выполнения основной версии проводится приемочный тест. Если тестирование не пройдено, выполняется первая альтернативная версия, и приемочный тест оценивается снова. Это продолжается до тех пор, пока не будут исчерпаны все альтернативные версии, и в этом случае модуль возвращает ошибку.

2 Тематическое моделирование

2.1 Метод тематического моделирования

Тематическое моделирование – одно из современных направлений статистической обработки естественного языка (natural language processing, NLP). Тематическая модель коллекции текстовых документов определяет, к каким темам относится каждый документ, и какие слова образуют каждую тему.

Вероятностная тематическая модель описывает каждую тему дискретным распределением вероятностей слов, а каждый документ – дискретным распределением вероятностей тем. Построенная модель способна преобразовывать любой текст в вектор вероятностей тем. Похожую задачу решают модели векторных представлений слов, предложений и документов, однако в них координаты не имеют интерпретации.

Тематическое моделирование похоже на кластеризацию документов. Отличие в том, что при кластеризации документ целиком относится только к одному кластеру, тогда как тематическая модель осуществляет мягкую кластеризацию, разделяя документ между несколькими кластерами-темами.

Многие приложения текстовой аналитики используют тематические векторные представления текста: выявление трендов в новостных потоках, патентных базах, архивах научных публикаций, многоязычный информационный поиск, поиск тематических сообществ в социальных сетях, классификация и категоризация документов, тематическая сегментация текстов, тегирование веб-страниц, обнаружение текстового спама. Существую и не-текстовые приложения тематического моделирования в анализе изображений и видеопотоков, в рекомендательных системах, в популяционной генетике, в биоинформатике для анализа нуклеотидных и аминокислотных последовательностей.

Построение тематической модели по коллекции документов является некорректно поставленной оптимизационной задачей, которая может иметь бесконечное множество решений. Согласно теории регуляризации А.Н. Тихонова [11], решение такой задачи возможно доопределить и сделать устойчивым. Для этого к оптимизационному критерию добавляется *регуляризатор* – дополнительный критерий, учитывающий специфические особенности прикладной задачи или знания предметной области. В сложных приложения дополнительных критериев может быть несколько.

Аддитивная регуляризация тематических моделей (ARTM) – это многокритериальный подход, в котором модель оптимизируется по взвешенной сумме критериев [3]. ARTM позволяет строить модели с требуемыми свойствами, перенося регуляризаторы из одних моделей в другие или объединяя регуляризаторы от различных моделей. Для обучения любых моделей и их комбинаций используется один и тот же алгоритм, к которому регуляризаторы подключаются как модули [33, 35, 36]. Модульность реализована в библиотеке тематического моделирование с открытым исходным кодом BigARTM [33, 35]. ARTM не является еще одной моделью или методом – это общий подход к построению и комбинированию тематических моделей.

Доминирующим подходом к тематическому моделированию в настоящее время является байесовское обучение. В отличие от ARTM, в нём нет естественного разделения моделей на универсальный алгоритм и отторгаемые от него модули-регуляризаторы. Для каждой модели приходится заново проводить математический вывод в программную реализацию.

Пусть D – множество (коллекция) текстовых документов, W – множество (словарь) всех употребляемых в них термов. Термами могут быть слова, нормальные формами слов, словосочетания или термины, в зависимости от того, какие виды предварительной обработки текстов были выполнены. Каждый документ $d \in D$ представляет собой последовательность n_d термов $\omega_1, \ldots, \omega_n$ из словаря W.

Гипотеза о существовании тем. Каждое вхождение терма w в документ d связано с некоторой темой t из заданного конечного множества T. Коллекция документов представляет собой последовательность троек $\Omega_n = \{(w_i, d_i, t_i) \mid i = 1, \ldots, n\}$. Термы w_i и документы d_i являются наблюдаемыми переменными, темы t_i не известны и являются латентными (скрытыми) переменными.

Гипотеза «мешка слов». Порядок термов в документах не важен для выявления тематики, то есть тематику документа можно узнать даже после произвольной перестановки термов, хотя для человека такой текст потеряет смысл. Это предположение называют гипотезой «мешка слов» (bag of words). Порядок документов в коллекции также не имеет значения – это предположение называют гипотезой «мешка документов». Гипотеза «мешка слов» позволяет перейти к компактному представлению документа как мультимножества – подмножества термов $w \in W$, в котором каждый терм $w \in d$ повторен n_{dw} раз.

Гипотеза о вероятностном по порождении данных. Множество $\Omega = D \times W \times T$ является конечным вероятностным пространством с неизвестной функцией вероятности $p(d, w, t)$. Коллекция документов является выборкой троек (w_i, d_i, t_i), порождаемых случайно и независимо друг от друга из распределения $p(d, w, t)$. Это предположение является вероятностным уточнением гипотезы «мешка слов».

Благодаря предположению о независимости, реализовавшуюся выборку Ω_n элементов из Ω можно рассматривать как новое вероятностное пространство с n равновероятными элементарными исходами. В пространстве n легко находить вероятности различных событий, причем они совпадают с частотными оценками вероятностей тех же событий в пространстве . В частности, в пространстве n выражение (1) равно вероятности события «терм w документа d связан с темой t», а в пространстве оно равно выборочной частотной оценке вероятности того же события.

$$p(d, w, t) = \frac{1}{n}\sum[d_i = d][w_i = w][t_i = t] \qquad (1)$$

Договоримся в дальнейшем записывать все вероятности в пространстве Ω, если не оговорено иного. Многие выкладки будут справедливы в обоих пространствах. Пространство Ω_n имеет формальное ограничение – оно строится по фиксированной коллекции. Если в коллекцию добавляются новые документы, то пространство Ω_n изменяется, тогда как пространство Ω остается неизменным.

Гипотеза условной независимости. Появление термов в документе d по теме t зависит от темы, но не зависит от документа d , и описывается общим для всех документов распределением $p(w \mid t)$:

$$p(w \mid d, t) = p(w \mid t) \qquad (2)$$

Вероятностная тематическая модель порождения текста. Согласно формуле полной вероятности и гипотезе условной независимости, распределение термов в документе $p(w \mid d)$ описывается вероятностной смесью распределений термов в темах $\varphi_{wt} = p(w \mid t)$ с весами $\theta_{td} = p(t \mid d)$:

$$p(w \mid d) = \sum_{t \in T} p(w \mid t, d)\, p(t \mid d) = \sum_{t \in T} \varphi_{wt} \theta_{td} \qquad (3)$$

Вероятностная модель (3) описывает процесс порождения коллекции по известным распределениям $p(w \mid t)$ и $p(t \mid d)$. Этот процесс показан на рисунке 2.1.

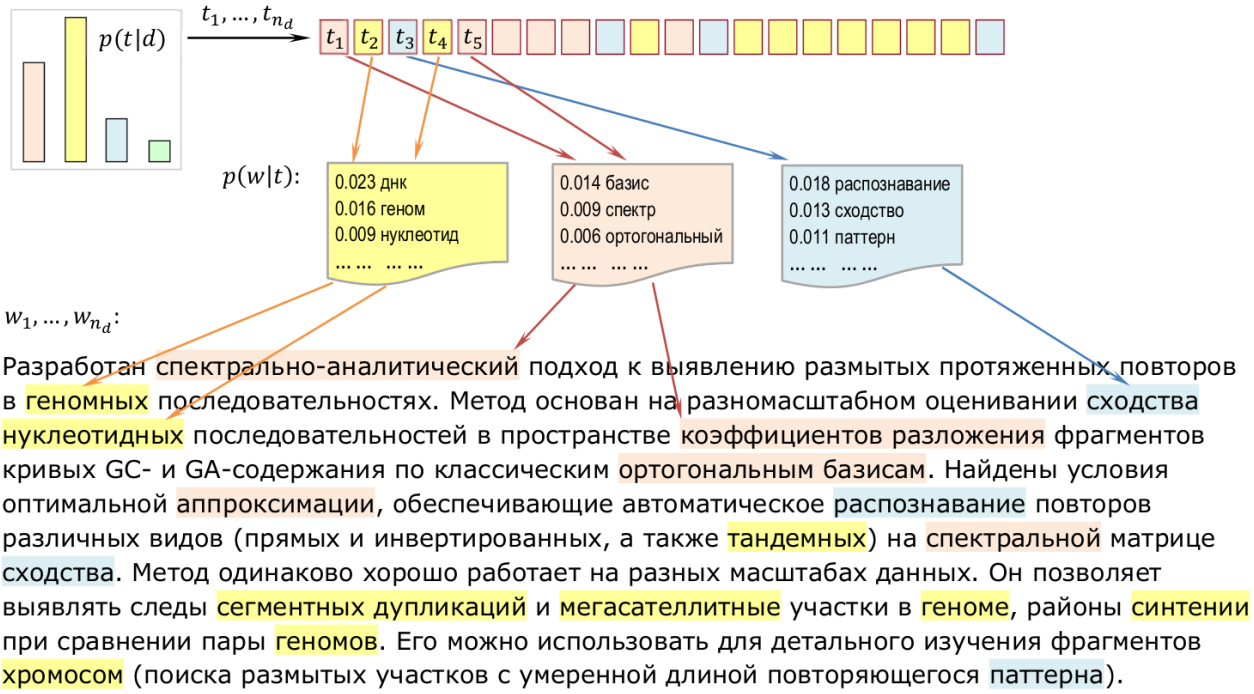

Рисунок 2.1 – Процесс порождения текстовой коллекции вероятностной тематической моделью

Задача тематического моделирования – это обратная задача: по заданной коллекции D требуется найти параметры φ_{wt} и θ_{td}, при которых тематическая модель (2) хорошо приближает частотные оценки условных вероятностей $p\,(w|d) = n_{dw} / n_d$.

Распределение вида $p(t|x)$ будем называть тематикой объекта x . Можно говорить о тематике документа $p(t|d)$, терма $p(t|w)$, терма в документе $p(t \mid d,w)$.

Целью тематического моделирования является выявление тематической кластерной структуры текстовой коллекции, определение тематики документов и связанных с ними объектов, описание семантики каждой темы t словами естественного языка с помощью распределений $p(w|t)$.

Низкоранговое матричное разложение. Равенство (2) можно переписать в матричном виде. В левой части равенства находится известная матрица частот термов в документах $F = (p(w \mid d))_{W \times D}$. Правая часть представляет собой произведение двух неизвестных матриц – матрицы термов тем $\Phi = (\varphi_{wt})_{W \times D}$ и матрицы тем документов $\Theta = (\theta_{td})_{T \times D}$. Обычно число тем $|T|$ много меньше $|D|$ и $|W|$, поэтому задача тематического моделирования сводится к поиску приближенного матричного разложения $F \approx \Phi\Theta$, ранг которого не превышает $|T|$.

Все три матрицы F, Φ, Θ являются стохастическими, то есть имеют неотрицательные нормированные столбцы f_d, φ_t, θ_d, представляющие дискретные распределения. Произведение называется стохастическим матричным разложением.

Частотные оценки условных вероятностей. В пространстве Ω_n вероятности, выражающиеся через переменные d и w , совпадают с частотами соответствующих наблюдаемых событий:

$$p(d,w) = \frac{n_{dw}}{n}, \quad p(d) = \frac{n_d}{n}, \quad p(w) = \frac{n_w}{n}, \quad p(w \mid d) = \frac{n_{dw}}{n_d}; \tag{4}$$

n_{dw} – число вхождений терма w в документ d;

$n_d = \Sigma_w n_{dw}$ – длина документа d в термах;

$n_w = \Sigma_d n_{dw}$ – число вхождений терма w во все документы коллекции;

$n = \Sigma_d \Sigma_w n_{dw}$ – длина коллекции в термах.

Вероятности, связанные со скрытой переменной t, тоже определяются как частоты:

$$p(t) = \frac{n_t}{n}, \; p(w \mid t) = \frac{n_{wt}}{n_t}, \; p(t \mid d) = \frac{n_{td}}{n_d}, \; p(t \mid d,w) = \frac{n_{tdw}}{n_{dw}} \tag{5}$$

n_{tdw} – число троек, в которых терм w документа d связан с темой t;

$n_{td} = \Sigma_w n_{tdw}$ – число троек, в которых терм документа d связан с темой t;

$n_{wt} = \Sigma_d n_{tdw}$ – число троек, в которых терм w связан с темой t;

$n_t = \Sigma_d \Sigma_w n_{tdw}$ – число троек, связанных с темой t.

В отличие от (4), эти частотные оценки не могут быть вычислены непосредственно по исходным данным, так как темы t_i неизвестны.

Согласно закону больших чисел, при $n \to \infty$ частотные оценки, определяемые формулами (4)–(5), стремятся к соответствующим вероятностям в пространстве Ω.

ЕМ-алгоритм. Нужно отметить, что все оценки (5) выражаются через $n_{tdw} = n_{dw} p(t|d,w)$. Зная условные распределения $p(t|d,w)$, можно оценить искомые параметры тематической модели $\varphi_{wt}=p(w|t)$ и $\theta_{td}=p(t|d)$. И, наоборот, зная параметры модели, можно выразить условные вероятности $p(t|d,w)$ по формуле Байеса:

$$p(t|d,w) = \frac{p(t,w|d)}{p(w|d)} = \frac{p(w|t)p(t|d)}{p(w|d)} = \frac{\phi_{wt}\theta_{td}}{\sum_s \phi_{ws}\theta_{sd}} \tag{6}$$

Таким образом, получаем систем нелинейных уравнений относительно параметров модели φ_{wt}, θ_{td} и вспомогательных переменных p_{tdw}, n_{wt}, n_{td}:

$$p_{tdw} = \frac{\varphi_{wt}\theta_{td}}{\sum_s \varphi_{ws}\theta_{sd}} \tag{7}$$

$$\varphi_{wt} = \frac{n_{wt}}{\sum_s \varphi_{wt}\theta_{sd}}; \qquad n_{wt} = \sum_{d \in D} n_{dw} p_{tdw} \tag{8}$$

$$\theta_{td} = \frac{n_{td}}{\sum_t n_{td}}; \qquad n_{td} = \sum_{w \in d} n_{dw} p_{tdw} \tag{8}$$

Для её решения удобно применять метод простых итераций: сначала выбираются начальные приближения параметров φ_{wt} и θ_{td}, по ним вычисляются вспомогательные переменные p_{tdw}, которые позволяют найти следующее приближение параметров φ_{wt} и θ_{td}. Вычисления по формулам (7)–(9) продолжаются в цикле до сходимости.

Этот итерационный процесс является частным случаем EM-алгоритма, предназначенного для построения вероятностных моделей со скрытыми переменными [37]. Вычисление условных распределений скрытых переменных (7) называется E-шагом (expectation), вычисление параметров модели (7)–(9) – M-шагом (maximization).

Далее выводится EM-алгоритм из общей оптимизационной постановки задачи. Такой подход к нему элементарным путем, который дает простое интуитивное понимание сути EM-алгоритма, но оставляет без ответов важные вопросы: сходится ли алгоритм к решению системы уравнений, единственно ли это решение, и почему эта система описывает тематическую модель, приближающую $p(w|d)$.

Рациональный EM-алгоритм. Вычисление переменных n_{wt}, n_{td}, n_t на M-шаге требует однократного прохода коллекции в цикле по всем термам $w \in d$ всех документов $d \in D$. Внутри этого цикла значение p_{tdw} вычисляется только один раз. Поэтому E-шаг можно встроить внутрь M-шага без дополнительных вычислительных затрат и без хранения трехмерной матрицы p_{tdw}. Этот вариант реализации EM-алгоритма будет называться рациональным.

3 Применение тематического моделирования для обнаружения проблем

3.1 Объекты мониторинга и сбор данных

Распределенная система, на примере которой рассматривается применение алгоритмов автоматизированного обнаружения и устранения проблем, представляет собой распределенную систему мониторинга сети национального оператора телерадиовещания. Объектами мониторинга являются оборудование радио-телевизионных станций, расположенных в населенных пунктах республики. Блоки оборудования РТС условно разделяются на две группы:

– оборудование, обеспечивающие передачу и прием информационных сигналов;

– вспомогательное и обеспечивающие оборудование.

В состав каждой РТС входит типовой набор устройств: коммутатор резервирования спутниковых трактов приема, приемник-ремультиплексор, модулятор DVB-T2, блок анализа качества сигнала сигнала эфирного вещания, контроллер системы электроснабжения, контроллер жизнеобеспечения, терминал VSAT, маршрутизатор Ethernet, контроллер системы кондиционирования и вентиляции, климатические датчики, датчики

охранной системы, контроллер системы охранно-пожарной сигнализации, контроллер системы видеонаблюдения.

Мониторинг состояния данных устройств на каждой станции осуществляется посредством блока контроля и управления. Блок контроля и управления представляет собой программно-аппаратный комплекс взаимодействующий с оборудованием станции с помощью таких протоколов как SNMP, Modbus, ICMP и т. д. В задачи БКУ входит:

– опрос оборудования;

– хранение и обработка полученных от оборудования данных;

– отправка данных имеющих значительную дельту по отношению к раннее отправленным данным;

– отображение исторических данных, событий, проблем.

Также к задачам БКУ стоит отнести соблюдение порядка очередности отправки данных. То есть, в случае отсутствия связи с центром управления сетью, БКУ должен отделить и накапливать данные предназначенные для отправки в дальнейшем. При возобновлении связи БКУ должен начать отправку данных в центр управления сетями. Временной период, за который на БКУ должны храниться данные – 6 месяцев.

Как говорилось выше, данные, собранные на БКУ, отправляются в центр управления сетью. В ЦУС-е агрегируются данные мониторинга с радио-телевизионных станций всей Республики. Эти данные должны храниться на ЦУС-е уже в течении трех лет. Содержимое базы данных ЦУС-а является источником данных для анализа методом тематического моделирования.

3.2 Сбор и подготовка данных для анализа

Объектом анализа являются события в системе мониторинга. Каждое событие это одиночное возникновение того, что заслуживает внимания. События возникают в результате изменения состояния триггеров. Событие в системе мониторинга Zabbix обладает рядом свойств: имя, источник, подтверждение наличия, время создания, значение события, критичность, время восстановления и т. д.

Источниками событий являются изменения параметров наблюдаемых устройств. Если полученное значение аварийное, то срабатывает триггер – появляется аварийное событие. События хранятся в базе данных. Интерфейс системы мониторинга позволяет просматривать события за определенный промежуток времени и от определенной группы устройств. Также посредством как основного интерфейса так и API возможно выгрузить события в виде таблицы в формате CSV.

"Важность"	"Время"	"Время восстан"	"Состояние"	"Узел сети"	"Проблема"	"Длительность"
"Высокая"	"15.01.2020 16:10:00"	""	"ПРОБЛЕМА"	Щит ввода электропитания 1 (esw1-f.alm0165s.ktr.kz)"	"device_availability_high"	"1м 18д 23ч"
"Высокая"	"15.01.2020 11:38:50"	""	"ПРОБЛЕМА"	Сервер управления и контроля энергоснабжением (srv-pw.alm0165s.ktr.kz)"	"device_availability_high"	"1м 19д 4ч"
"Чрезвычайная"	"09.12.2019 14:08:21"	""	"ПРОБЛЕМА"	Сайт-контроллер (scu.alm0165s.ktr.kz)"	"matrix_disaster"	"2м 26д 1ч"
"Предупреждение"	"05.12.2019 14:24:48"	""	"ПРОБЛЕМА"	Спутниковый приёмник 2 (rcv2.alm0165s.ktr.kz)"	"link_margin_signal_4_warning"	"3м 1ч"
"Предупреждение"	"05.12.2019 14:03:36"	""	"ПРОБЛЕМА"	Спутниковый приёмник 1 (rcv1.alm0165s.ktr.kz)"	"link_margin_signal_4_warning"	"3м 1ч"
"Высокая"	"25.11.2019 07:01:23"	""	"ПРОБЛЕМА"	Management-коммутатор 1 (sw1-mgmt.alm0165s.ktr.kz)"	"port13_high"	"3м 10д 8ч"
"Высокая"	"21.11.2019 11:11:34"	""	"ПРОБЛЕМА"	IP-телефон служебной технологической связи 1 (vt1.alm0165s.ktr.kz)"	"icmp_check_failed"	"3м 14д 4ч"

Рисунок 3.1 – Несколько событий выгруженных из системы мониторинга

Однако, такой формат необходимо привести формату совместимому с BigARTM. Форматы, поддерживаемые BigARTM:

– Vowpal Wabbit;

– UCI Bag-of-words;

– Batches – бинарный формат специфичный для BigARTM.

Vowpal Wabbit – это одноформатный файл, основанный на следующих принципах:

– каждый документ представлен одной строкой;

– все токены представлены в виде строк (нет необходимости преобразовывать их в целочисленный идентификатор);

– частота токенов по умолчанию равна 1 или может быть дополнительно указаны после двоеточия;

– пространство имен (Batch.class_id) может быть определено через символ pipe.

```
doc1 Alpha Bravo:10 Charlie:5 |author Ola_Nordmann
doc2 Bravo:5 Delta Echo:3 |author Ivan_Ivanov
```

Рисунок 3.2 – Пример файла в формате Vowpal Wabbit

– при размещении токенов в каждом документе в их оригинальном порядке и без указания их частот приведет к модели с последовательным текстом – не «мешок слов».

```
doc1 this text will be processed not as bag of words | Some_Author
```

Рисунок 3.3 — Пример файла в формате Vowpal Wabbit с оригинальной последовательностью токенов

Формат «UCI Bag-of-words» состоит из двух файлов – vocab.*.txt и docword.*.txt. Формат файла docword.*.txt выглядит следующим образом: сначала указываются заголовочные строчки которые описывают дальнейшую последовательность и значение чисел.

```
D
W
NNZ
docID wordID count
docID wordID count
...
docID wordID count
```

Рисунок 3.4 – Пример файла в формате «UCI Bag-of-words»

Здесь мы можем увидеть, что первые три заголовочные строки описывают структуру последующих строк: идентификатор документа, идентификатор токена, количество токенов в документе.

Файл должен быть отсортирован по идентификатору документа. Значения идентификатора документа должны соответствовать единому

```
token1 @default_class
token2 custom_class
token3 @default_class
token4
```

формату. Формат файла vocab.*.txt – это строка, содержащая строки формата wordID=n. Тут необходимо отметить, что слова не должны иметь пробелов или табуляций. В файле vocab.*.txt также можно указать пространство имен (Batch.class_id) для токенов.

Рисунок 3.5 – Пример файла vocab.*.txt с указанием пространства имен (Batch.class_id)

Для разделения токена и его класса необходимо использовать пробел или табуляцию. Токены, для которых не указываются их классы, автоматически присваиваются к классу @default_class.

Формат Batch представляет собой специально созданный для BigARTM двоичный формат файла. Это очень компактный и эффективный формат, базирующийся на таких сущностях как Batch и Item, где:

– batch – это коллекция из нескольких item-ов;

– item – набор пар (идентификатор токена, вес токена).

Batch имеет свой локальный словарь – batch.dictionary. В данном словаре каждому токену сопоставляется его идентификатор.

Наиболее простым и подходящим по всем требованиям форматом файла является формат Vowpal Wabbit. Для начала необходимо выгрузить из системы мониторинга события всех станций за достаточный промежуток времени для создания наиболее репрезентативной картины. Для взаимодействия с системой мониторинга используется ее API. Инструментом для работы с API выбрана библиотека zabbix.api для языка программирования Python (приложение А.1).

Для начала импортируем необходимые библиотеки groups и zabbix.api. Здесь groups представляет собой простой справочный файл в котором перечислены названия, которые, в свою очередь, являются названиями групп узлов сети в системе мониторинга.

```
groupids = [{'groupid': '45', 'groupname': 'site_Коктобе'}, {'groupid': '46', 'groupname': 'site_Астана'},
            {'groupid': '47', 'groupname': 'site_Караганда'}, {'groupid': '48', 'groupname': 'site_Бейнеу'},
            {'groupid': '49', 'groupname': 'site_Боранкуль'}, {'groupid': '50', 'groupname': 'site_Форт-Шевченко'},
            {'groupid': '51', 'groupname': 'site_Курык'}, {'groupid': '52', 'groupname': 'site_Кызылозен'},
            {'groupid': '53', 'groupname': 'site_Онды'}, {'groupid': '54', 'groupname': 'site_Сайотес'},
            {'groupid': '55', 'groupname': 'site_Сенек'}, {'groupid': '56', 'groupname': 'site_Шайыр'},
            {'groupid': '57', 'groupname': 'site_Тущыкудык'} ...]
```

Рисунок 3.6 – Отрывок из файла groups.py с перечислением названий РТС

zabbix.api – python-библиотека для взаимодействия с API системы мониторинга Zabbix. Данная библиотека позволяет программно извлекать и изменять конфигурацию Zabbix, также обеспечивает доступ к данным истории. Библиотека широко используется для:

– создания новых приложений для работы с Zabbix;

– интеграции Zabbix со сторонним программным обеспечением;

– автоматизации рутинных задач.

Zabbix API является API на основе веб и поставляется как часть веб-интерфейса. Он использует протокол JSON-RPC 2.0, из чего вытекает следующее:

– API состоит из набора отдельных методов;

– запросы и ответы между клиентами и API закодированы с использованием формата JSON.

Для инициации взаимодействия с API системы мониторинга создается объект класса ZabbixAPI посредством аутентификации. С помощью метода event.get данного объекта запрашиваются события для каждой группы узлов сети. Так как количество событий может быть значительным – необходимо произвести запрос несколькими итерациями. Полученные события после каждой итерации необходимо должным образом конкатенировать и записать в файл в вышеуказанном формате Vowpal Wabbit.

Рисунок 3.7 – Строка событий для типового ОДРТ из файла в формате Vowpal Wabbit

Здесь можно увидеть структуру строки: в начале находится название текста, далее следует разделитель «|», за ним располагается название модальности и в оставшуюся часть строки занимают токены. Токен представляет собой конкатенированные названия узла сети и имя события. Это необходимо для точного отделения событий друг от друга, так как на совершенно разных узлах сети могут присутствовать одинаковые события. К примеру, событие icmp_check_high, сообщающее о недоступности узла сети через протокол ICMP, может присутствовать как на передатчике, так и на ip-камере. Такие события будут иметь различный смысл, но здесь будут представлены как одно. Через двоеточие указывается количество случаев фиксации события на узле сети, например: «tsm1_TSE800_B_current_active_TS_input_info: 276» говорит о том, что на первом передатчике вход «B» активировался для транспортного потока 276 раз.

3.3 Анализ событий с помощью модели PLSA

Для начала импортируются необходимые модули. Для создания графиков из библиотеки matplotlib импортируется модуль pyplot. matplotlib.pyplot – это набор функций в командном стиле которые делают matplotlib похожим на MATLAB. Каждая функция pyplot вносит некоторые изменения в график: например, создает график, задает область построения на осях, отрисовывает линии на графике, наносит метки и т. д.

Следующим шагом является создание объекта класса artm.BatchVectorizer. artm.BatchVectorizer – это универсальный объект – векторизатор, который отдается на вход всем операциям BigARTM. Векторизатор осуществляет загрузку и преобразование данных во внутренний формат пакетов документов: батчей.

Основные атрибуты объекта:

– collection_name – название текстовой коллекции;

– data_path – адрес документа в файловой системе;

– data_format – тип входной информации. Возможные типы: bow_uci, vowpal_wabbit, bow_n_wd и batches;

– batch_size – количество документов, которые будут храниться в одном batch-файле;

– target_folder – путь к директории, где будут храниться сформированные batch-файлы;

– gather_dictionary – создание словаря по умолчанию в векторизаторе;

– class_ids – модальность или список модальностей для анализа и включения в batch.

Есть два основных способа создания векторизатора. Первый путь: векторизатор создается по матрице «мешка слов» словаря. Словарь определяет соответствие между объектами матрицы и словами коллекции. В таком случает пакеты, полученные в результате данной операции создаются в оперативной памяти. По завершению работы с ними объекты удаляются из оперативной памяти.

Второй способ создания предполагается в случае большого объема данных, который невозможно поместить в объем доступной оперативной памяти. В таком случае чтение данных будет происходить непосредственно с жесткого диска. Итоговые пакеты также будут записываться на диск.

Здесь используется формат исходных данных VowpalWabbit. При таком формате каждая строка такого файла соответствует одному документу. При использовании данного формата документы можно представить как последовательным текстом, так и «мешком слов». Кроме того, документы могут содержать термы различных модальностей.

Следующим шагом является создание PLSA-модели. PLSA (probabilistic latent semantic analysis) – модель вероятностного латентного семантического анализа. PLSA – представляет собой исторически первую вероятностную тематическую модель. Данная модель была предложена Томасом Хофманном в 1999 году [34]. В ARTM она соответствует нулевому регуляризатору, R(Ф,

Θ) = 0. То есть совпадает с системой (6)-(8). В модели PLSA не может быть вырожденных тем или документов, поскольку вырожденность связана с отрицательностью производных регуляризаторов.

Модель PLSA создается как объект класса artm.ARTM. Основные атрибуты объекта:

– num_topics – количество тем в модели, будет переписано в случае указания в конструкторе атрибута topic_names;

– num_processors – определяет как много потоков будут использованы для тренировки модели. В случае если данный параметр не задан, то количество потоков будет определено самой библиотекой;

– topic_names – названия тем в модели;

– class_ids – список идентификаторов классов и их веса для использования в модели. Ключом является идентификатор класса, а значением – вес;

– cache_theta – данный параметр определяет: сохранять или нет матрицу Θ в модели. Этот параметр необходим в случае дальнейшего использования метода get_theta();

– scores – список метрик: объекты классов artm.*Scores;

– regularizers – список регуляризаторов: объекты классов artm.*Regularizer;

– num_document_passes – количество внутренних итераций внутри каждого документа;

– dictionary – словарь, который предназначен для инициализации модели;

– reuse_theta – параметр, который определяет использование или неиспользование матрицы Θ из предыдущих итераций;

– theta_columns_naming – идентификатор или название, определяющее наименование документов в матрице Θ;

– show_progress_bar – логический флаг, указывающий, показывать ли индикатор выполнения в операция fit_offline, fit_online и transform;

– theta_name – название матрицы Θ;

– parent_mode – объект класса ARTM для использования в качестве родительского уровня в иерархии;

– parent_model_weight – вес родительского модуля.

Создаваемый объект имеет следующие атрибуты:

– количество искомых тем равно десяти, т.к. десять – это общепринятое предполагаемое количество тем для начала поиска;

– словарь принимается от объекта векторизатора;

– темам присваиваются названия из диапазона от нуля до девяти, т.к. количество тем равно десяти;

– единственный идентификатор – text, так как в исходных данных присутствует только одна модальность.

Для возможности контроля процесса обучения модели необходимы некоторые метрики, которые будут отображать качество полученного

результата. Для этого выбраны следующие метрики: PerplexityScore, SparsityPhiScore, SparsityThetaScore, TopTokensScore.

PerplexityScore – перплексия. Перплексия – наиболее распространенный внутренний критерий, используемый для оценивания вероятностных моделей языка в компьютерной лингвистике. Для ее корректной работы нужен словарь, содержащий нормированные частоты слов в коллекции. Частоты используются в качестве аппроксимации нулевых значений *p(w|d)* и позволяют корректно оценивать модели на одном словаре, но с разной степенью разреженности.

Перплексия – это мера несоответствия или "удивленности" модели *p(w|d)* термам *w*, которые встречаются в документах *d*. Она определяется через log-правдоподобие каждой модальности *m*:

$$P_m(D;p) = exp\left(\frac{-1}{n_m}\sum_{d\in D}\sum_{w\in W^m} n_{dw}\, lnp(w \mid d)\right) \quad (10)$$

где $n_m = \sum_{d=D}\sum_{w\in W^m} n_{dw}$– длина коллекции по *m*-й модальности. Чем меньше перплексия, тем лучше модель *p* предсказывает появление термов *w* в документах *d*.

Перплексия имеет следующую интерпретацию. Если термы *w* порождаются из равномерного распределения *p(w) = 1/V* на словаре мощности *V*, то перплексия языковой модели *p(w)* на таком тексте сходится к *V* с ростом его длины. Чем сильнее распределение *p(w)* отличается от равномерного, тем меньше перплексия. В случае условных вероятностей *p(w|d)* интерпретация немного другая: если каждый документ генерируется из *V* равновероятных термов (возможно, различных в разных документах), то перплексия сходится к *V*.

Неочевидность численных значений перплексии, ее зависимость не только от качества модели, но и от размерных характеристик коллекции длины документов, мощности словаря, разреженности вероятностного распределения термов – являются основными недостатками перплексии. К примеру, неправильно сравнивать тематические модели по одной й той же коллекции, построенные на разных словарях.

SparsityPhiScore/SparsityThetaScore – разреженности матриц Ф и Θ. Оцениваются долей элементов матрицы, меньших заданного пользователем порога.

TopTokensScrore – топ-слова в темах, список из заданного числа слов с наибольшей вероятностью по каждой теме. Если в параметрах этой метрики указать словарь парной сочетаемости слов, то будет вычислена когерентность $coher_t$ по спискам топ-слов в темах, согласно формуле:

$$l_t = \frac{2}{k(k-1)}\sum_{i=1}^{k-1}\sum_{j=i}^{k} PMI(w_i, w_j) \quad (11)$$

После этого необходимо проинициализировать модель с помощью метода initialize. В виде аргумента данному методу отдается словарь сгенерированный векторизатором.

Следующим шагом является обучение модели. Обучение модели можно производить с помощью функций ARTM.fit_offline или ARTM.fit_online. Методы отличаются друг от друга режим обучения модели. То есть метод fit_online используется для обучения модели в режиме online, а метод fit_offline используется для обучения модели в режиме offline. Режим offline подходит для тех случаев, когда размер входных данных относительно невелик и весь объем данных может поместиться в оперативную память. Offline алгоритм имеет следующие черты: много проходов через коллекцию, один проход через один документ (опционально), только одно обновление матрицы Ф за один проход сбора (в конце прохода). Этот алгоритм стоит использовать при обработке небольшой коллекции.

Режим онлайн подходит для обратного случая. Здесь входные данные поступают в модель порционно. Онлайн алгоритм характеризуют следующие отличительные черты: один проход через коллекцию (опционально), много проходов через один документ, несколько обновлений матрицы Ф за один проход через коллекцию. Этот алгоритм стоит используется, когда предстоит имееть дело с большими коллекциями, а также с коллекциями с быстро меняющимися темами.

Так как объем входных данных для примера имеет небольшой размер – используется метод fit_offline. Параметром данной функиции является num_collection_passes – количество итераций по всей полученной коллекции. Чем больше количество проходов – тем больше времени необходимо модели для обучения. Также, чем больше количество итераций – тем выше показатели обучения модели. Однако, со временем показатель обучения модели достигает некоторого значения и в дальнейшем продолжает расти незначительно. В связи с этим необходимо выбрать такое количество итераций, которое позволило бы обучить модель должным образом и не использовало лишние аппаратные ресурсы.

Для начала значение количество итераций установим равным 20. Время обучения занимает около 5 минут. Для оценки качества обучения рассмотрим раннее установленные метрики. Первая метрика – перплексия. Перплексия – это мера несоответствия или "удивленности" модели $p(w|d)$ термам w, которые встречаются в документах d. Рассмотрим полученный график. Чем меньше перплексия, тем лучше модель p предсказывает появление термов w в документах d.

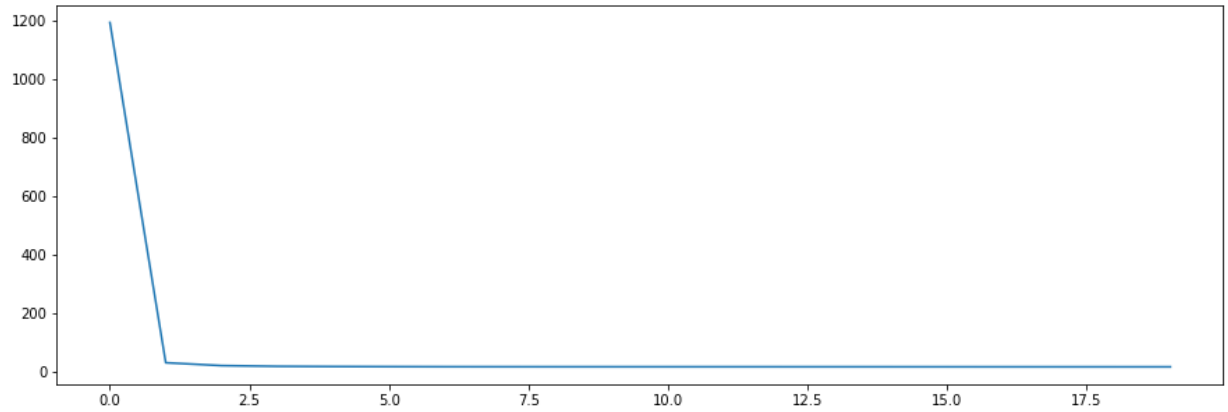

Рисунок 3.8 – График перплексии для модели PLSA

Данный график демонстрирует, что модель довольно быстро сошлась к достаточно маленькому значению.

Следующая метрика на которую необходимо обратить внимание – SparsityPhiScore. Данная метрика демонстрирует разреженность матрицы Ф.

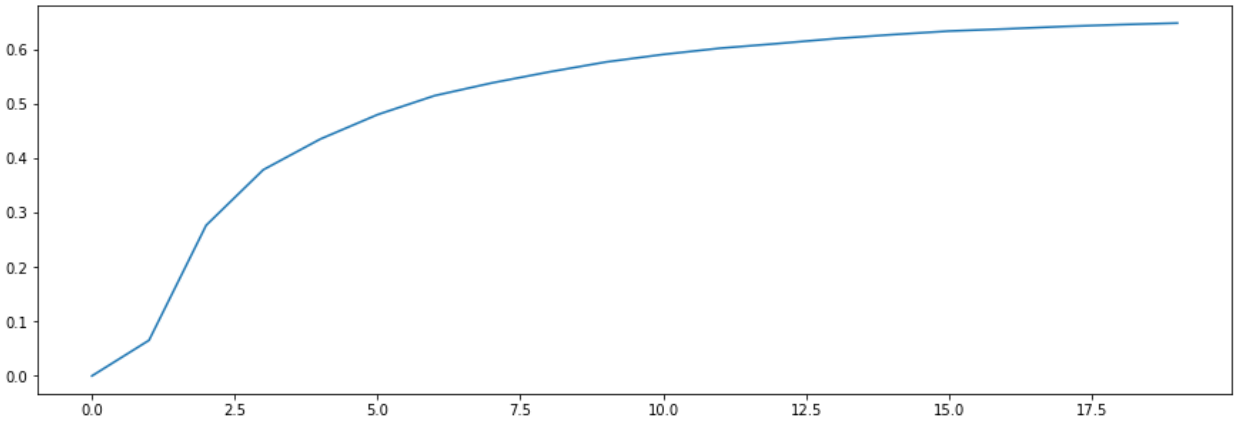

Рисунок 3.9 – График значений метрики SparsityPhiScore

На этом графике достаточно хорошо видно динамику изменения метрики: к двадцатой итерации значение метрики уже почти не меняется. Соответственно, можно утверждать, что двадцать итераций – достаточное количество и в дальнейших экспериментах будет использоваться данное значение.

Последнее значение метрики SparsityPhiScore равно – 0.649. Такое значение нельзя назвать достаточно хорошим. Оно свидельствует о том, что модель еще недостаточно хорошо соотносит термы и тематики.

Рассмотрим же следующую метрику – SparsityThetaScore. SparsityThetaScore – разреженность матрицы Θ. Эта метрика оценивает доли элементов матрицы, меньших заданного пользователем порога.

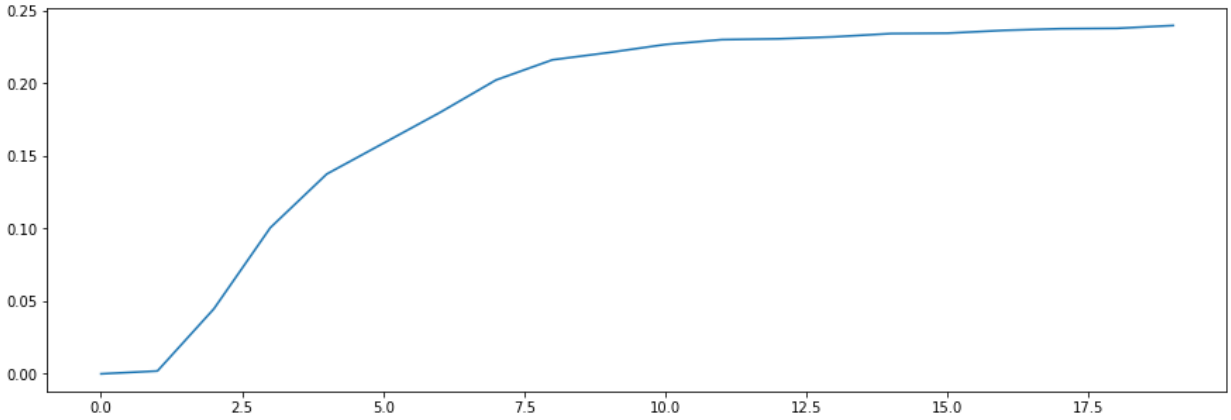

Рисунок 3.10 – График значений метрики SparsityThetaScore

На данном графике также можно отметить, что к двадцатой итерации график метрики выравнивается и почти неменяется. Метрика SparsityThetaScore к двадцатой итерации достигла занчения 0.24. Такое показатель тоже нельзя назвать достаточно высоким. Он, в свою очередь, свидетельствует о том, что модель пока недостаточно хорошо соотносит тексты и темы.

Метрика TopTokensScrore наглядно демонстрирует эффективность работы модели. С ее помощью можно увидеть термы наиболее часто встречающиеся в определенных темах. Если эти термы имеют между собой явную смысловую связь, то можно утверждать, что тема действительно опеределена. При создании модели было определено количество тем равное 10. Полный вывод из 15 встречающихся термов для каждой темы приведен в приложении А.3. Здесь же приведена урезанная метрика из пяти термов на тему:

– sbj0: sm_Spacelink_Status scu_mysqld_test_high scu_mysql_port_unavailable_high scu_snmp_port_unavailable_high scu_max_data_diff_test_high

– sbj1: sm_icmp_check_high srvpw_AC2_State_info srvpw_AC1_AC2_State_high enens1_RFT2_1_EtrPcrAccuracyError_warning scu_local_time_high

– sbj2: enens2_IP2_EtrPcrRepErr_warning sw1-mgmt_port14_high esw1-f_SHV-1_VoltageL3_N_warning esw1-f_SHV-1_VoltageL2_N_warning esw1-f_SHV-1_VoltageL1_N_warning

– sbj3: esw1-f_SHV-1_PhaseAngleL1_L2_warning esw1-f_SHV-1_PhaseAngleL1_L3_warning esw1-f_SHV-1_VoltageL3_N_warning icam3_icmp_check_high esw1-f_SHV-1_VoltageL1_N_warning

– sbj4: mnu-vm_slms4ESFrozenErrTrap_high mnu-vm_slms4TSvcPMTLossErrTrap_high tsm1_txSummary.TxB_warning scu2_matrix_info tsm1_txSummary.TxB_disaster

– sbj5: enens1_RFT2_1_EtrCrcErr_warning dvms-t2_measSigInterfacesDvbt2AGC

Locked_2_high enens2_IP1_EtrCrcErr_warning dvms-t2_measSigInterfacesDvbt2 AGCLocked_1_high enens2_IP2_EtrCrcErr_warning

– sbj6: dvms-s2_PCRrepetition_2_0 dvms-t2_PCRrepetition_1_warning dvms-s2_PCRrepetition_1_0 dvms-s2_Ccerror_2_0 tsm1_txSummary_warning

– sbj7: enens2_IP2_EtrPcrAccuracyError_warning enens2_IP1_EtrTsContErr_high enens2_IP2_EtrTsContErr_high enens1_RFT2_1_EtrCrcErr_warning enens1_RFT2_1_EtrPcrAccuracyError_warning

– sbj8: lsw_DEV-2183_Data_Logger_Amplifier_1_status_disaster enens2_IP1_EtrPcrRepErr_warning enens1_RFT2_1_EtrTsContErr_high esw2-f_SHV-1_SHV-2_PhaseAngleL1_L3_high esw2-f_SHV-1_SHV-2_PhaseAngleL1L2_high

– sbj9: enens1_RFT2_1_EtrTsContErr_high enens1_RFT2_1_EtrTransportErr_warning enens1_RFT2_1_EtrPcrRepErr_warning enens1_RFT2_1_EtrPcrDiscIndError_warning enens1_RFT2_1_EtrTsSyncLoss_disaster

Эти результаты можно интерпретировать следующим образом. Посмотрим на токены из первой тематики. Здесь можно увидеть, что четыре из пяти токенов относятся к блоку контроля и управления, в частности, три токена из четырех сообщают о неполадках в базе данных: scu_mysqld_test_high, scu_mysql_port_unavailable_high, scu_max_data_diff_test_high. Иными словами, из полученных токенов вполне можно предположить, что эта тематика описывает проблемы с СУБД блока контроля и управления. Однако, стоит отметить, что здесь также присутствует токен сообщающий о проблемах со спутниковым каналом: sm_Spacelink_Status. Этот токен имеет слабую смысловую связь с остальными токенами данной темы.

В тематике sbj6 можно заметить, что четыре из пяти токенов свидетельствуют о проблемах на анализаторах транспортных потоков DVMS. В тематике sbj9 также пристутствуют токены проблем на анализаторах транспортного потока. Эти анализаторы разные устройства, которые устанавливаются на разные типы станций, однако, токены обеих тематик говорят о проблемах временной синхронизации MPEG-потоков. Такое разделение тематик с одинаковыми проблемами, но с разными устройствами, можно считать успешным результатом.

Четыре токена из пяти в тематике sbj3 сообщают о проблемах с напряжением и углами фаз на шкафу ввода. Действительно, на практике эти проблемы зачастую сопровождают друг друга.

Остальные тематики выражены не столь сильно. Однако, если принять в расчет то, что данные результаты получены с помощью простейшей базовой модели, то можно констатировать, что результат довольно успешный. Для того чтобы улучшить показатели модели необходимо доопределить задачу, т.е. добавить дополнительный критерий в виде регуляризации.

3.4 Анализ событий с помощью модели ARTM с регуляризатором SmoothSparsePhiRegularizer

Для начала импортируются необходимые модули. Для создания графиков из библиотеки matplotlib импортируется модуль pyplot. matplotlib.pyplot – это набор функций в командном стиле которые делают matplotlib похожим на MATLAB. Каждая функция pyplot вносит некоторые изменения в график: например, создает график, задает область построения на осях, отрисовывает линии на графике, наносит метки и т. д.

Следующим шагом является создание объекта класса artm.BatchVectorizer. artm.BatchVectorizer – это универсальный объект – векторизатор, который отдается на вход всем операциям BigARTM. Векторизатор осуществляет загрузку и преобразование данных во внутренний формат пакетов документов: батчей.

Основные атрибуты объекта:

– collection_name – название текстовой коллекции;

– data_path – адрес документа в файловой системе;

– data_format – тип входной информации. Возможные типы: bow_uci, vowpal_wabbit, bow_n_wd и batches;

– batch_size – количество документов, которые будут храниться в одном batch-файле;

– target_folder – путь к директории, где будут храниться сформированные batch-файлы;

– gather_dictionary – создание словаря по умолчанию в векторизаторе;

– class_ids – модальность или список модальностей для анализа и включения в batch.

Есть два основных способа создания векторизатора. Первый способ: векторизатор создается по матрице «мешка слов» словаря. Словарь определяет соответствие между объектами матрицы и словами коллекции. В таком случает пакеты, полученные в результате данной операции создаются в оперативной памяти. По завершению работы с ними объекты удаляются из оперативной памяти. Второй способ создания предполагается в случае большого объема данных, который невозможно поместить в объем доступной оперативной памяти. В таком случае чтение данных будет происходить непосредственно с жесткого диска. Итоговые пакеты также будут записываться на диск. Однако, так как в прошлом разделе уже были получены готовые батчи, то здесь буду использованы они.

Батчи – двоичный формат BigARTM. Это компактный и эффективный формат, основанный на сериализованных данных в общедоступном интерфейсе BigARTM (батчи и элементы). Батч представляет собой набор из нескольких элементов. Элемент – это набор пар (token_id, token_weight). Нужно обратить внимание на то, что пакет имеет свой локальный словарь, batch.token. Этот словарь, который содержит соответствия идентификаторы токенов фактическим токенам. Чтобы создать батч из текстовых файлов, нужно найти все отдельные слова и отобразить их в последовательные индексы.

Следующим шагом является создание ARTM-модели. Модель ARTM создается как объект класса artm.ARTM. Модель ARTM создается со следующими атрибутами:

– количество искомых тем равно десяти, т.к. десять – это общепринятое предполагаемое количество тем для начала поиска;

– словарь принимается от объекта векторизатора;

– темам присваиваются названия из диапазона от нуля до девяти, т.к. количество тем равно десяти;

– единственный идентификатор – text, так как в исходных данных присутствует только одна модальность.

Перечислим основные методы объекта ARTM-модели:

– clone() – возвращает полную копию объекта artm.ARTM. Данный метод создает полную копию объекта: внутренне состояние, матрицы Φ и Θ, метрики и регуляризаторы;

– dispose() – освобождает всю оперативную память, предназначенную для данной модели;

– dump_artm_model(*data_path*) – выгрузить дамп всей модели в определенную директорию;

– fit_offline(batch_vectorizer=None, num_collection_passes=1, reset_nwt=True) – запуск обучения модели в режиме оффлайн;

– fit_online(batch_vectorizer=None, tau0=1024.0, kappa=0.7, update_every=1, apply_weight=None, decay_weight=None, update_after=None, asynchronous=False) – запуск обучения модели в режиме онлайн;

– get_phi(topic_names=None, class_ids=None, model_name=None) – получить пользовательскую матрицу Φ модели;

– get_theta(topic_names=None) – получить матрицу Θ для обучения набора документов (или кэшированных после преобразования);

– get_score(score_name) – получить оценку результата после применения методово fit_offline, fit_online или transform;

– info – возвращает внутреннюю диагностическую информацию о модели;

– initialize(dictionary=None) – инициализация тематическое модели перед обучением;

– load(filename, model_name='p_wt') – загрузка модели сохраненной на жестком диске при помощи метода ARTM.save();

– remove_theta() – удалить кэшированную матрицу Θ;

– reshape(topic_names=None, dictionary=None) – изменит форму модели;

– reshape_tokens(dictionary) – обновить токены модели;

– reshape_topics(topic_names) – обновить токены модели;

– set_parent_model(parent_model, parent_model_weight=None) –установка родительской модели. Родительская модель – объект класса ARTM для использования в качестве родительского уровня иерархии;

– topic_names – получить или задать список имен тем модели, также позволяет задать новые названия для существующих тем;

– transform(batch_vectorizer=None, theta_matrix_type='dense_theta', predict_class_id=None) – найти матрицу Θ для новых документов. Результирующий объект pandas.DataFrame будет содержать плоскую матрицу Θ, где у каждого элемента есть несколько столбцов – столько же, сколько токенов в этом документе. Эти столбцы будут иметь одинаковый идентификатор элемента. Порядок столбцов с одинаковым идентификатором элемента совпадает с порядком токенов во входных данных (batch.item.token_id);

– transform_sparse(batch_vectorizer, eps=None) – найти матрицу Θ для новых документов в виде разреженной матрицы.

Также как и для модели PLSA в модель ARTM, для возможности контроля процесса обучения модели необходимо ввести некоторые метрики, которые будут отображать качество полученного результата. Для этого выбраны следующие метрики: PerplexityScore, SparsityPhiScore, SparsityThetaScore, TopTokensScore.

Следующим шагом является инициализация модели с помощью метода initialize(). В виде аргумента данному методу отдается словарь сгенерированный векторизатором. Этот метод вызывается в конструкторе ARTM, если задать ему параметр имени словаря. Стоит отметить, что изменение начального поля повлияет на вызов метода initialize(). Также, если у модели начинают появлять признаки деградации, то нет необходимости создавать новою модель. Для исправления ситуации достаточно всего лишь использовать метод initialize(), который заполнит матрицу Φ случайными числами и не изменит ничего другого (ни настройки регуляризаторов/метрик, ни истории из score_tracker).

Для улучшения работы модели в нее необходимо добавить регуляризацию. Регуляризаторы могут воздействовать на матрицы Φ, Θ или (p_{tdw}). Регуляризаторы Φ могут воздействовать на отдельные модальности. Наличие параметра class_id указывает, что регуляризатор работает с одной модальностью, по умолчанию с default_class. Наличие параметра class_ids указывает на то, что регуляризатор работает со списком модальностей, по умолчанию со всеми. Почти все параметры всех регуляризаторов можно менять между итерациями обучения.

Введем в модель регуляризатор SmoothSparsePhiRegularizer. SmoothSparsePhiRegularizer – регуляризатор сглаживания матрицы Φ, реализован по следующей формуле:

$$\varphi_{wt} = \mathrm{norm}(n_{wt} + \tau\beta_w f(\varphi_{wt})) \qquad (12)$$

Если в определении KL-дивергенции заменить логарим ln x на функцию $\lambda(x)$, то $f(x) = x\lambda(x)$. По умолчанию $f(x) = 1$, что и соответствует логариму. В библиотеке можно задавать f(x) как степенную функцию. Вектор (β_ω) загружается из словаря и задаётся значениями поля value каждого слова. Для

каждой темы t может быть задан свой такой вектор. Таким способом можно задавать “белые” или “черные” списки слов для частичного обучения.

В работе данного регуляризатора важную роль играет коэффициент τ. При регуляризации данный коэффициент будет умножаться на частоту токена. То есть, при применении регуляризатора чем большую частоту имеет токен, то тем менее вероятнее он будет входить в отдельные темы.

Коэффициент τ подбирается опытным путем. Для начала зададим коэффициент τ равным -10. Количество проходов по коллекции текстов оставим таким же равным 20.

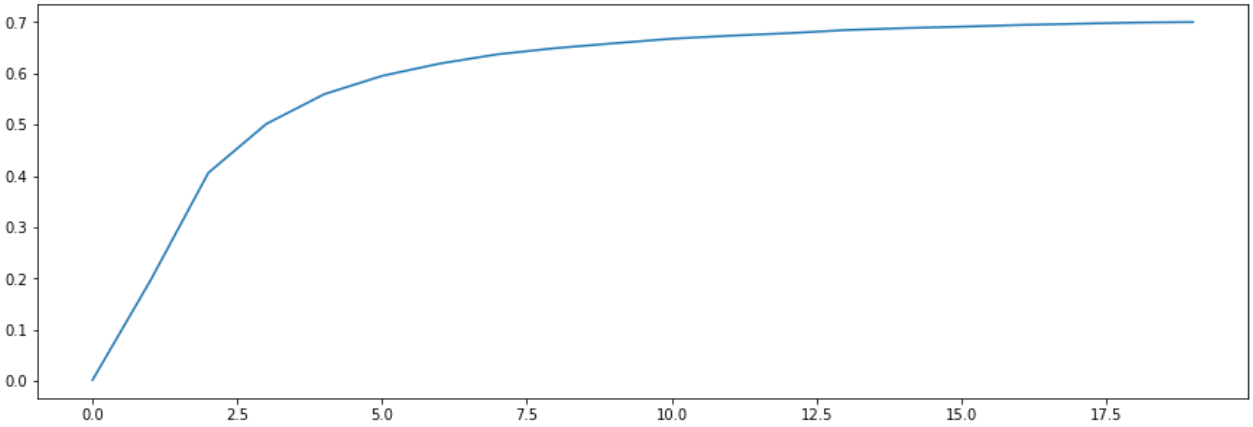

Рисунок 3.11 – График значений метрики SparsityPhiScore с коэффициентом τ=-10

Значение метрики SparsityPhiScore на последнем проходе по коллекции текстов стало 0.699. Последнее значение данной метрики при использовании модели PLSA и таком же количестве проходов по коллекции было равным 0.649.

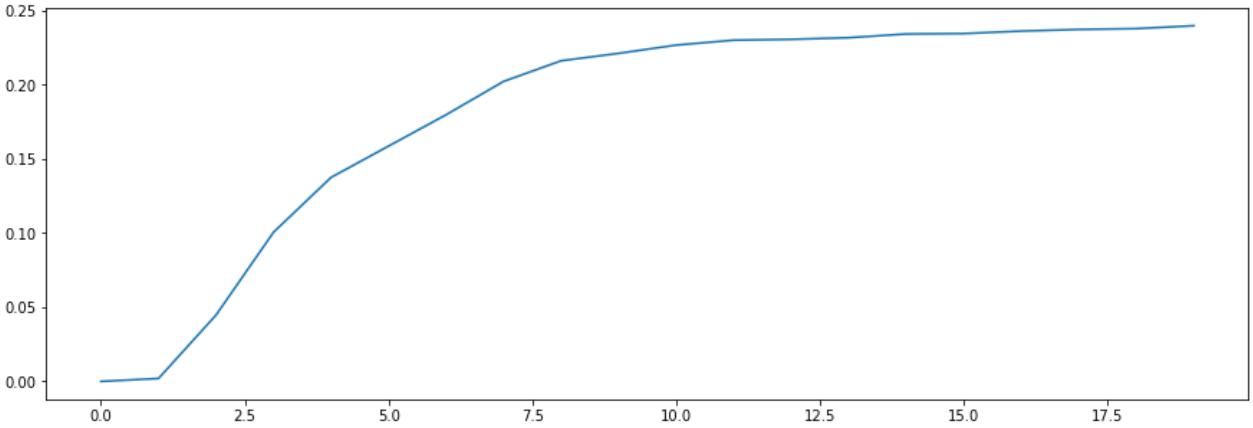

Рисунок 3.12 – График значений метрики SparsityThetaScore при применении регуляризатора с коэффициентом τ=-10

Метрика SparsityThetaScore ожидаемо не ощутила почти никакого влияния применения данного регуляризатора. Значение метрики на последнем проходе – 0.239.

Такой прирост метрики SparsityPhiScore трудно назвать существенным. Попробуем увеличить данный показатель путем уменьшения коэффициента τ до -1000.

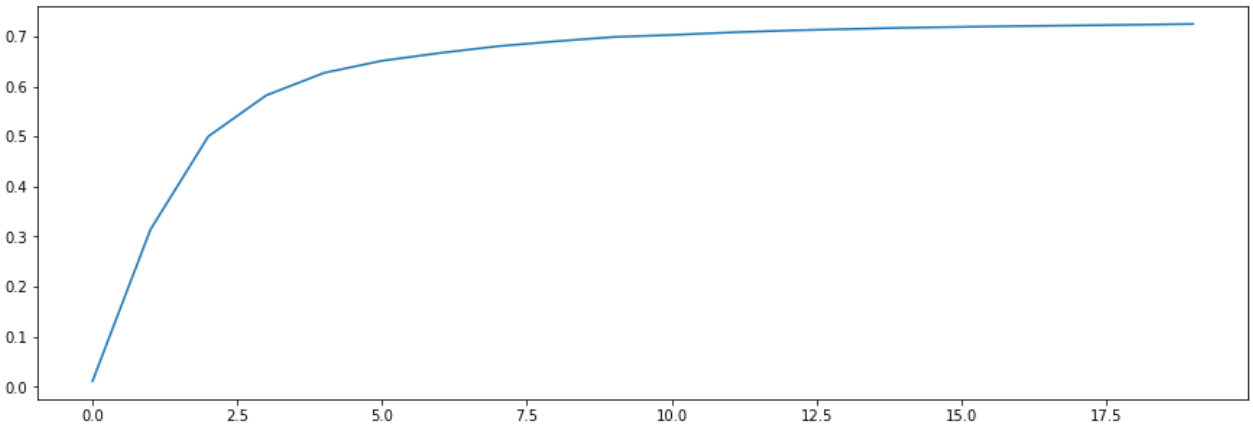

Рисунок 3.13 – График значений метрики SparsityPhiScore с коэффициентом τ=-1000

Значение метрики SparsityPhiScore на последнем проходе по коллекции текстов при повышении коэффициента τ до -1000 стало 0.724. Последнее значение данной метрики при использовании коэффициента τ равным 10 и таком же количестве проходов по коллекции было равным 0.699.

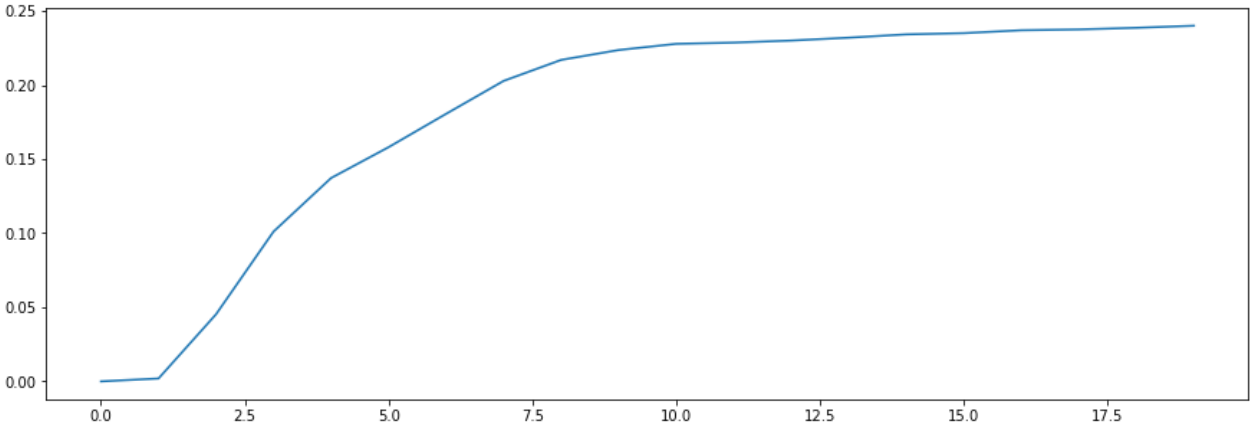

Рисунок 3.14 – График значений метрики SparsityThetaScore при применении регуляризатора с коэффициентом τ=-1000

Изменение коэффициента τ до -1000 также не оказало влияние на значение метрики SparsityThetaScore. Значение метрики на последнем проходе осталось таким же – 0.239.

Попробуем же увеличить показатель разреженности матрицы Φ еще на некоторую величину. Изменим коэффициент τ до -10000.

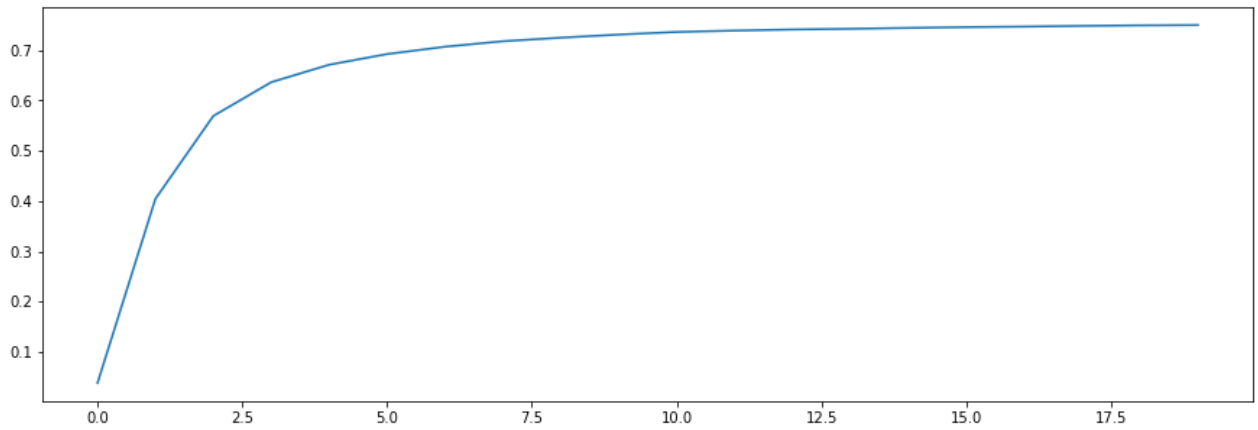

Рисунок 3.15 – График значений метрики SparsityPhiScore с коэффициентом τ=-10000

Значение метрики SparsityPhiScore на последнем проходе по коллекции текстов при повышении коэффициента τ до -10000 стало равным 0.75. Последнее значение данной метрики при использовании коэффициента τ равным 1000 и таком же количестве проходов по коллекции было равным 0.724.

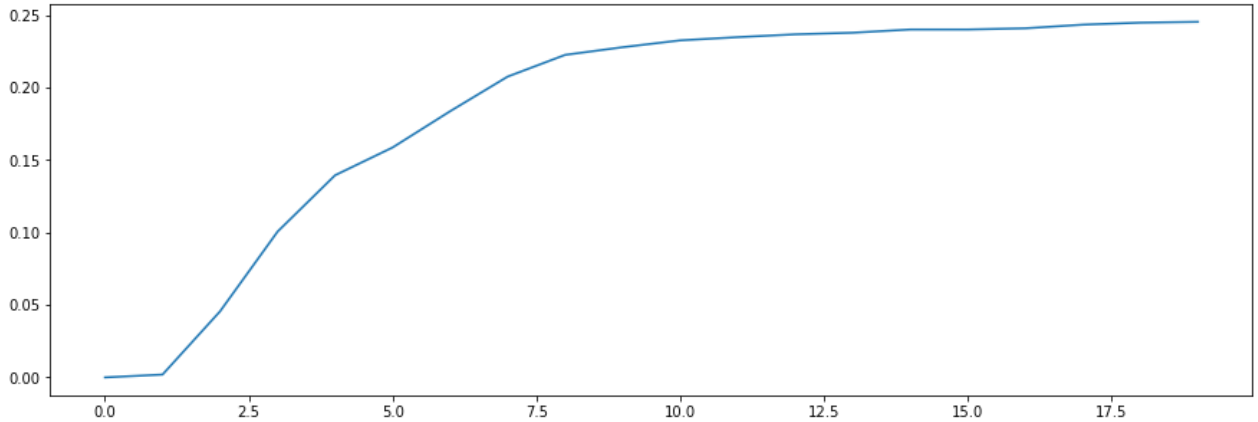

Рисунок 3.16 – График значений метрики SparsityThetaScore при применении регуляризатора с коэффициентом τ=-10000

При значениии коэффициента τ до -10000 значение метрики SparsityThetaScore на последнем проходе повысилось и стало равным 0.245.

Здесь стоит отметить, что зависимость между коэффициентом τ и получаемым значниями метрики SparsityPhiScore нелинейна. По этой причиние постоянное повышение коэффициента τ не рационально. При опередленном значении частота токенов в коллекции текстом просто занулится. Поэтому стоит остановиться на значении коэффициента τ равным -10000.

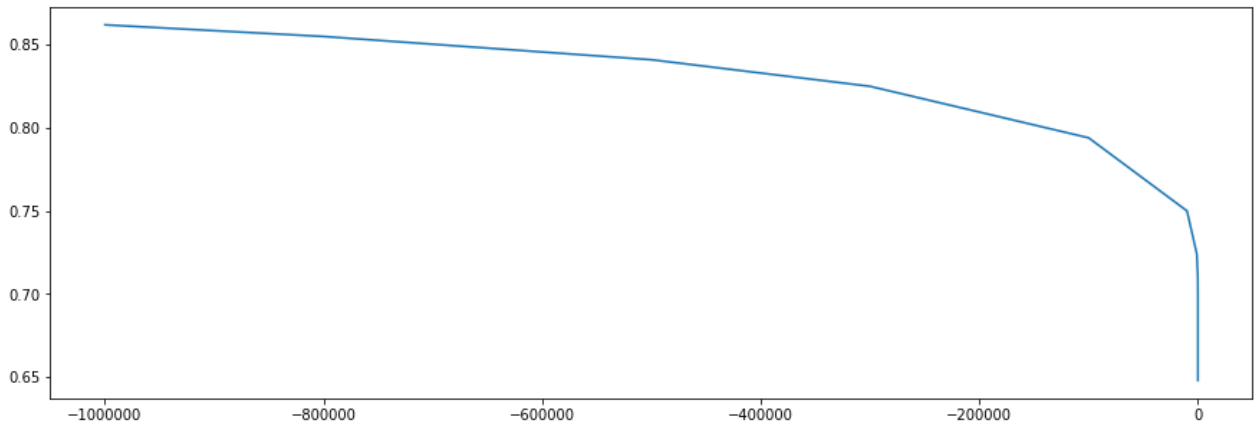

Рисунок 3.17 – График зависимости значений метрики SparsityPhiScore от коэффициента τ

3.5 Анализ событий с помощью модели ARTM с регуляризатором SmoothSparseThetaRegularizer

После того как путем введения регуляризатора SmoothSparsePhiRegularizer удалось добиться улучшения метрики SparsityPhiScore, необходимо проделать эти же действия с матрицей Θ. Для этого будет использоваться специальный регуляризатор SmoothSparseThetaRegularizer.

SmoothSparseThetaRegularizer – регуляризатор сглаживания разреживания матрицы Θ. Реализован по следующей формуле:

$$\varphi_{wt} = \text{norm}(n_{td} + \tau\alpha_i\alpha_{td}f(\theta_{td})) \quad (13)$$

Функция f играет ту же роль, что и в регуляризаторе сглаживания разреживания Ф. Массив множителей α_i позволяет управлять воздействием регуляризатора на каждой i-й внутренней итерации обработки документа. Вектор или матрица (α_{td}) позволяет управлять воздействием регуляризатора на элементы матрицы Θ.

Для этого необходимо создать объект регуляризатора с необходимыми параметрами и добавить его в список регуляризаторов модели (Приложение А.4). В данном регуляризаторе, как и в предыдущем для матрицы Ф, присутсвует коэффициент τ. Оптимальная величина данного коэффициента подбирается экспериментальным путем, т.е. необходимо провести несколько итераций экспериментов с изменением коэффициента в отрицательную сторону.

Для начала возьмем коффициент τ раным -1. Количество эпох оставим неизменным равным 20. Параметры регуляризатора SmoothSparsePhiRegularizer будут использоваться из предыдущего этапа экспримента.

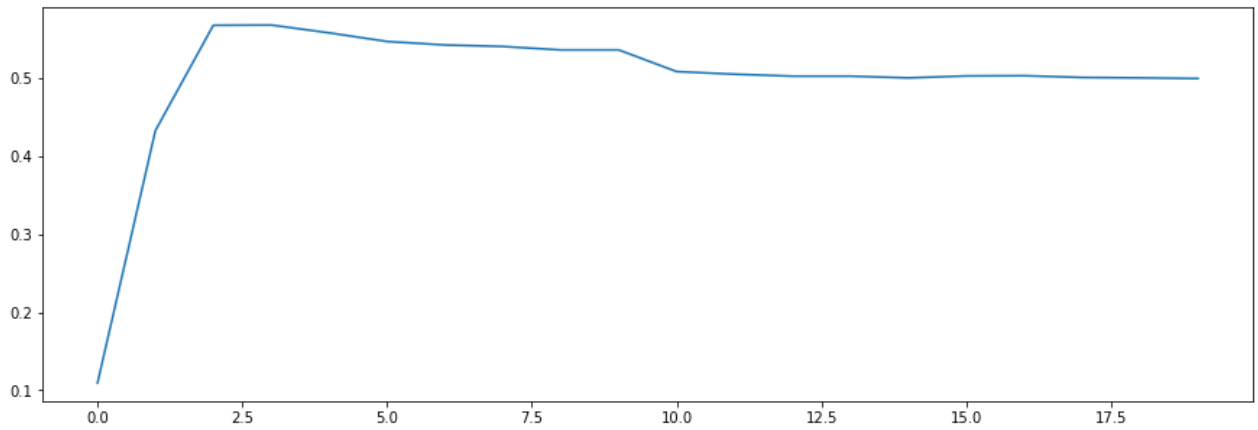

Рисунок 3.18 – График значений метрики SparsityThetaScore при применении регуляризатора с коэффициентом τ=-1

На данном графике становится достаточно сильно заметным влияние регуляризатора SmoothSparseThetaRegularizer. При столь незначительном размере коэффициента τ форма кривой значительно отличается от формы кривой без использования данного регуляризатора. Значение метрики SparsityThetaScore на последнем проходе равно 0.499, что больше значения данной метрики без использования регуляризатора в 2 раза.

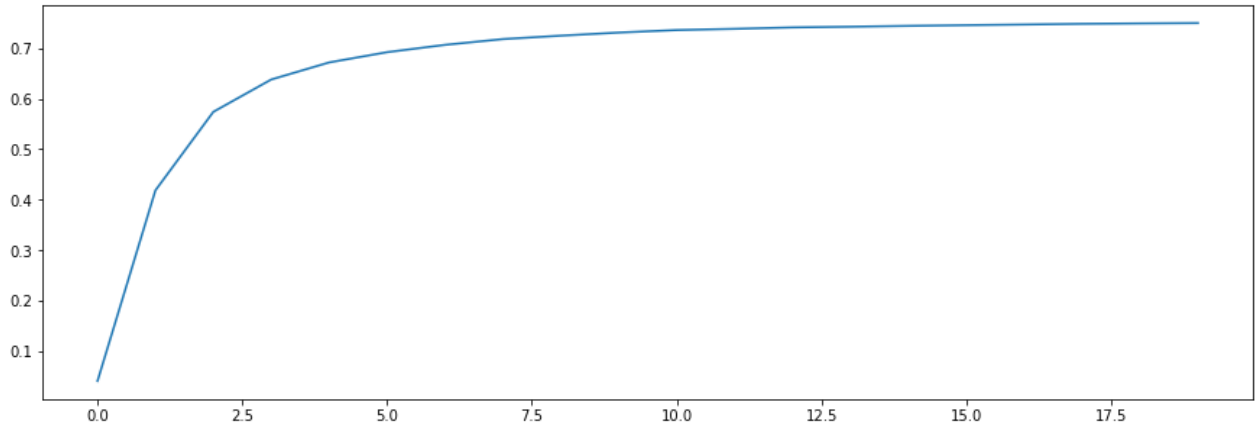

Рисунок3.19 – График значений метрики SparsityPhiScore с коэффициентом τ=-1

Введение данного регуляризатора, как и следовало ожидать, не оказало никакого влияния форму графика значений метрики SparsityPhiScore. Значение метрики на последнем проходе равно 0.749.

Повторим обучение модели с коэффициентом τ равным -10.

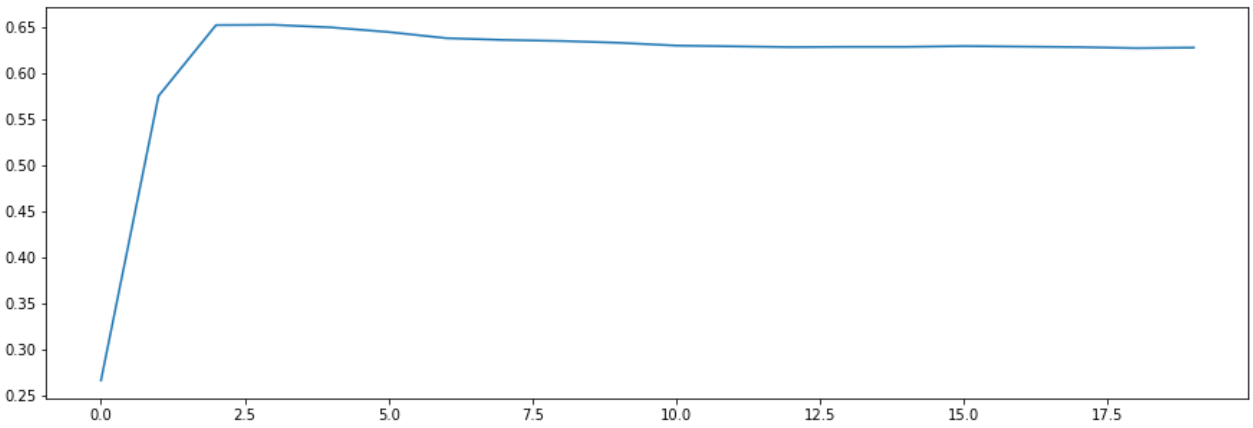

Рисунок 3.20 – График значений метрики SparsityThetaScore при применении регуляризатора с коэффициентом τ=-10

Полученная форма кривой в целом повторяет форму предыдущей с коэффициентом τ=-1. Значение метрики SparsityThetaScore на последнем проходе равно 0.627.

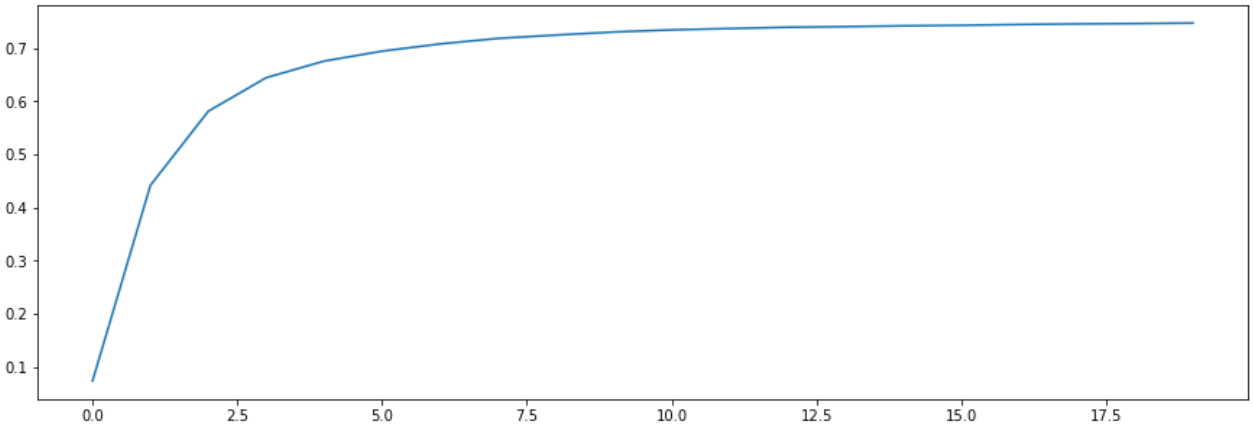

Рисунок 3.21– График значений метрики SparsityPhiScore с коэффициентом τ=-10

Значение метрики SparsityPhiScore на двадцатом проходе по коллекции равно 0.747. Изменение коэффициента τ в регуляризаторе SmoothSparseThetaRegularizer некоторым образом уменьшает значение данной метрики.

Значение метрики SparsityThetaScore равное 0.627 видится не самым большим. Присутствует вероятность того, что этот показатель можно улучшить изменив коэффициент τ до -100.

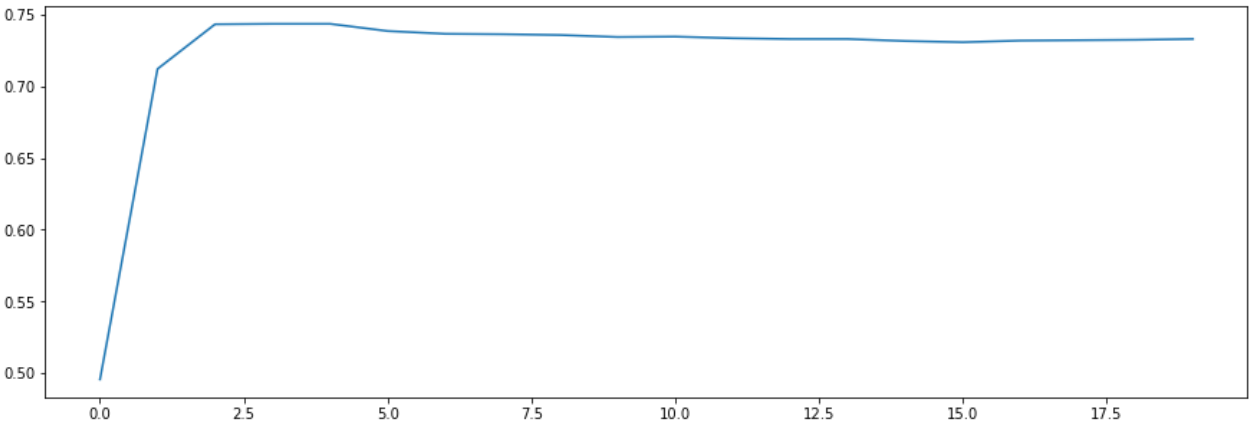

Рисунок 3.22 – График значений метрики SparsityThetaScore при применении регуляризатора с коэффициентом τ=-100

Значение метрики SparsityThetaScore на последнем проходе по коллекции равно 0.733.

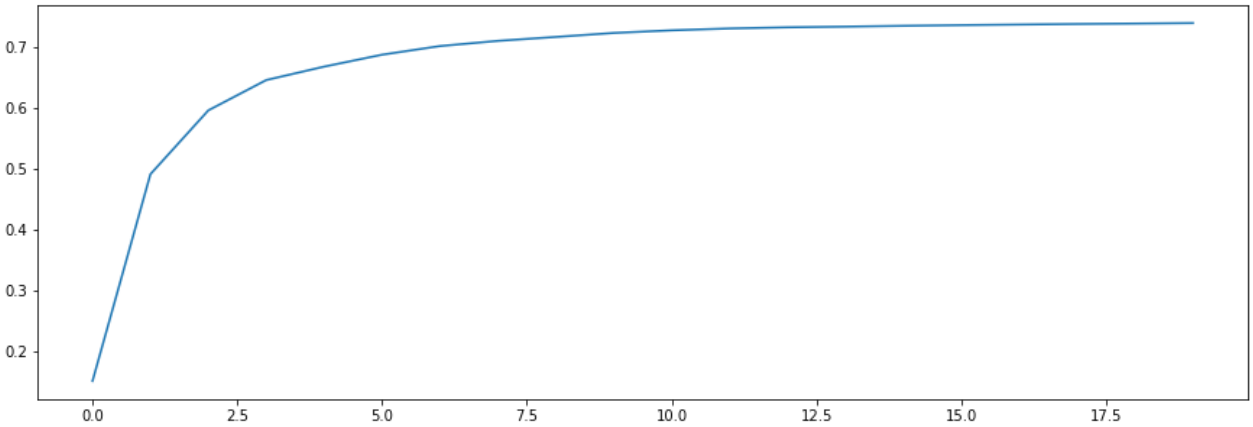

Рисунок 3.23 – График значений метрики SparsityPhiScore с коэффициентом τ=-100

Значение метрики SparsityPhiScore на последнем проходе по коллекции равно 0.739.

Дальнейшее изменение коэффициента τ в отрицательную сторону негативно отражается на метрике SparsityPhiScore. Полученное же значение метрики SparsityThetaScore с таким коэффициентом τ является достаточным. Кроме того, дальнейшее увеличение коэффициента τ негативно влияет на саму же метрику SparsityThetaScore.

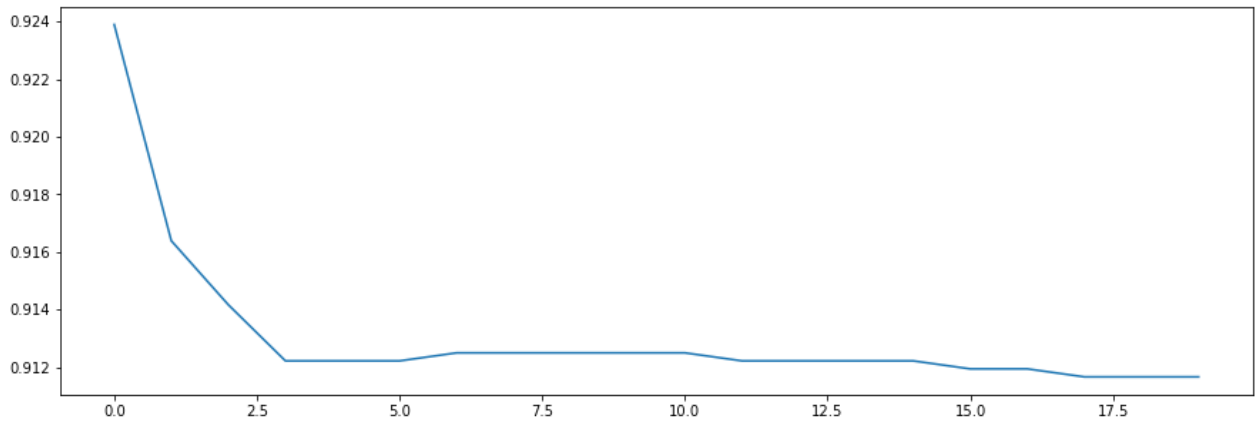

Рисунок 3.24 – График значений метрики SparsityThetaScore при применении регуляризатора с коэффициентом τ=-10000

На данном рисунке можно увидеть, при коэффициенте τ=-10000 график значений метрики SparsityThetaScore меняется координальным образом.

Помимо этого, еще одной причиной, по которой не имеет смысла увеличивать коэффициент τ, служит нелинейность зависимости значение метрики SparsityThetaScore от коэффициент τ.

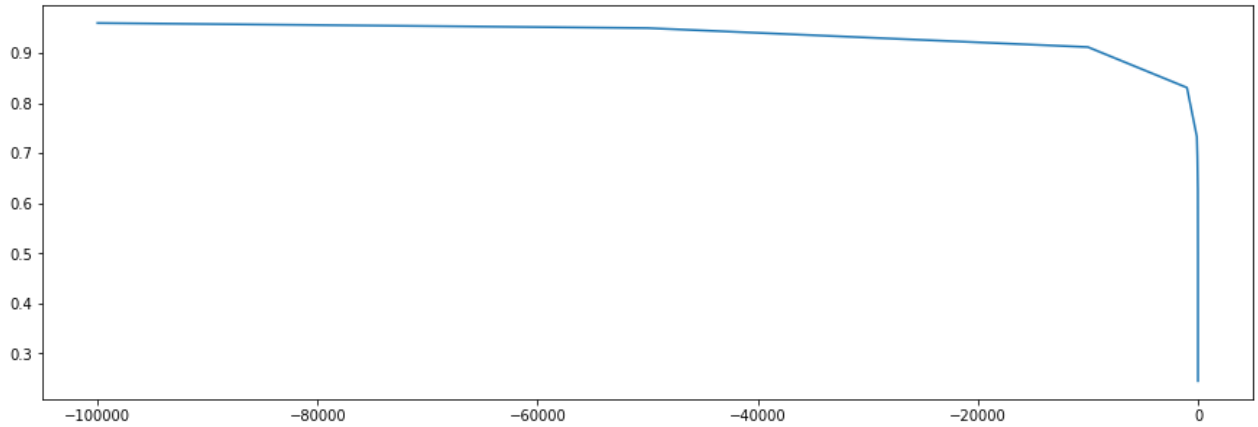

Рисунок 3.25 – Зависимость метрики SparsityThetaScore от коэффициента τ

На графике достаточно хорошо видно, что необязательно выбирать слишком отрицательные значения, так как это не окажет почти никакого влияния на метрику.

После того как оптимальные параметры модели заданы и модель обучена, необходимо оценить результат. Посмотрим список тем полученый от модели с данными параметрами:

–

sbj0: sm_Spacelink_Status scu_mysqld_test_high scu_mysql_port_unavailable_high scu_snmp_port_unavailable_high scu_max_data_diff_test_high

– sbj1: sm_icmp_check_high srv-pw_AC2_State_info srv-pw_AC1_AC2_State_high scu_local_time_high enens1_RFT2_1_EtrPcrAccuracyError_warning

– sbj2: enens2_IP2_EtrPcrRepErr_warning sw1-mgmt_port14_high esw1-f_SHV-1_VoltageL3_N_warning esw1-f_SHV-1_VoltageL2_N_warning esw1-f_SHV-1_VoltageL1_N_warning

– sbj3: esw1-f_SHV-1_PhaseAngleL1_L2_warning esw1-f_SHV-1_PhaseAngleL1_L3_warning esw1-f_SHV-1_VoltageL3_N_warning icam3_icmp_check_high esw1-f_SHV-1_VoltageL1_N_warning

– sbj4: mnu-vm_slms4ESFrozenErrTrap_high mnu-vm_slms4TSvcPMTLossErrTrap_high tsm1_txSummary.TxB_warning scu2_matrix_info tsm1_txSummary.TxB_disaster

– sbj5: enens1_RFT2_1_EtrCrcErr_warning dvms-t2_measSigInterfacesDvbt2AGCLocked_2_high enens2_IP1_EtrCrcErr_warning dvms-t2_measSigInterfacesDvbt2AGCLocked_1_high enens2_IP2_EtrCrcErr_warning

– sbj6: dvms-s2_PCRrepetition_2_0 dvms-t2_PCRrepetition_1_warning dvms-s2_PCRrepetition_1_0 dvms-s2_Ccerror_2_0 tsm1_txSummary_warning

– sbj7: enens2_IP2_EtrPcrAccuracyError_warning enens2_IP1_EtrTsContErr_high enens2_IP2_EtrTsContErr_high enens1_RFT2_1_EtrCrcErr_warning enens1_RFT2_1_EtrPcrAccuracyError_warning

– sbj8: lsw_DEV-2183_Data_Logger_Amplifier_1_status_disaster enens2_IP1_EtrPcrRepErr_warning enens1_RFT2_1_EtrTsContErr_high esw2-f_SHV-1_SHV-2_PhaseAngleL1_L3_high esw2-f_SHV-1_SHV-2_PhaseAngleL1_L2_high

– sbj9: enens1_RFT2_1_EtrTsContErr_high enens1_RFT2_1_EtrTransportErr_warning enens1_RFT2_1_EtrPcrRepErr_warning enens1_RFT2_1_EtrPcrDiscIndError_warning enens1_RFT2_1_EtrTsSyncLoss_disaster

Заключение

В результате этой работы был предложен, реализован и оценен эффективный алгоритм оценки состояния телекоммуникационной сети в целом и отдельных её узлов в частности. Предлагаемый алгоритм позволяет эффективно вычислять проблемные места в системе. Результат подобной оценки позволяет операторам заранее подготовиться решению большого количества аварий, а также, при регулярном анализе системы, отслеживать на сколько эффективно определенные мероприятия уменьшают вероятность появления критических аварий.

Однако, несмотря на то, что были применены регуляризаторы, позволяющие сделать матрицы Φ и Θ более разреженными и тем самым избавиться от случайных и незначительных токенов, это не принесло значимого результата. Тематики, полученные с помощью модели PLSA, и темы, полученные с помощью модели ARTM, почти не имеют отличий. Может ли это говорить о неэффективности регуляризаторов как механизма? Очевидно, нет. Это видно из того, что использование регуляризаторов действительно оказывало влияние на метрики обеих матриц. Однако, из этого возникает логичный вопрос: почему введение регуляризаторов модели не оказало почти никакого влияния на результат? Ответ на этот вопрос стоит искать не в модели, а в источнике данных. Причина заключается в том, что клиентами системы мониторинга являются люди. По объективным причинам человек может работать только с ограниченным количеством информации, поступающей с ограниченной скоростью. То есть, для максимального удобства работы пользователя с событиями системы мониторинга, событий должно быть как можно меньше, а информации, заключенной в каждом событии, напротив, должно быть как можно больше. Следствием этого является тот факт, что события имеют миниальную связь между собой. В свою очередь, поиск этих связей и является задачей тематического моделирования.

Для более лучшего выявления тематик из данных мониторинга с помощью метода тематического моделирования нужно изменить сам подход к сбору информации. Для этого необходимо собирать данные о большем количестве параметров оборудования. Такой подход позволит гораздо более отчетливо описывать потенциальные проблемы и улучшит прогностические способности модели.

Список литературы

1. Tanenbaum A., Van Steen M. Distributed systems. Pearson Prentice Hall, 2007.

2. A. Avizienis, J. C. Laprie, B. Randell, and C. Landwehr. Basic concepts and taxonomy of dependable and secure computing. IEEE Transactions on Dependable and Secure Computing, 1(1):11–33, 2004.

3. Воронцов К.В. Аддитивная регуляризация тематических моделей коллекций текстовых документов // Доклады РАН. – 2014. – Т. 456, № 3. – С. 268–271.

4. M. Y. Chen, E. Kidman, E. Fratkin, A. Fox, and E. Brewer. Pinpoint: Problem determination in large, dynamic internet services. In Proceedings of the 2002 International Conference on Dependable Systems and Networks, DSN '02, pages 595–604, Washington, DC, USA, 2002. IEEE Computer Society.

5. L.Lamport, R.Shostak, and M.Pease. The byzantine generals problem. ACM Transactions on Programming Languages and Systems, 4:382–401, 1982.

6. P. M. Melliar-Smith and B. Randell. Software reliability: The role of programmed exception handling. In Proceedings of an ACM conference on Language design for reliable software, pages 95-100, New York, NY, USA, 1977. ACM.

7. B. Randell and J. Xu. The evolution of the recovery block concept. In *Software Fault Tolerance*, pages 1–22. John Wiley & Sons Ltd, 1994.

8. J. Shore. Fail fast. *IEEE Software*, pages 21–25, September/October 2004.

9. A. Avizienis and L. Chen. On the implementation of n-version programming for software fault tolerance during execution. In Proceedings of the IEEE International Computer Software and Applications Conference, pages 149–155, 1977.

10. M. Franz. Understanding and countering insider threats in software development. In Proceedings of the International MCETECH Conference on e-Technologies, pages 81–90, January 2008.

11. Тихонов А.Н., Арсенин В.Я. Методы решения некорректных задач. – М.: Наука, 1986.

12. Tarkoma S. Overlay Networks, Toward Information Networking. CRC Press, Boca Raton, FL, 2010. Cited on 3, 81

13. Bernstein P. Middleware: A Model for Distributed System Services. Communications of the ACM, 39(2):87–98, Feb. 1996. Cited on 5, 34

14. Alonso G., Casati F., Kuno H., and Machiraju V. Web Services: Concepts, Architectures and Applications. Springer-Verlag, Berlin, 2004. Cited on 6, 34

15. Renesse R.van , Birman K., Cooper R., Glade B., and Stephenson P. The Horus System. In Birman K. and Renesse R.van , editors, Reliable and Distributed Computing with the Isis Toolkit, pages 133–147. IEEE Computer Society Press, Los Alamitos, CA., 1994. Cited on 6

16. Hohpe G. and Woolf B. Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions. Addison-Wesley, Reading, MA., 2004. Cited on 34, 38, 212

17. Amar L., Barak A., and Shiloh A. The MOSIX Direct File System Access Method for Supporting Scalable Cluster File Systems. Cluster Computing, 7(2):141–150, Apr. 2004. Cited on 27

18. Engelmann C., Ong H., and Scott S. Middleware in Modern High Performance Computing System Architectures. In International Conference on Computational Science,volume 4488 of Lecture Notes in Computer Science, pages 784–791, Berlin, May 2007. Springer-Verlag. Cited on 27

19. Foster I., Kesselman C., and Tuecke S. The Anatomy of the Grid, Enabling Scalable Virtual Organizations. Journal of Supercomputer Applications, 15(3):200–222, Fall 2001. Cited on 28, 29

20. Joseph J., Ernest M., and Fellenstein C. Evolution of grid computing architecture and grid adoption models. IBM Systems Journal, 43(4):624–645, Apr. 2004. Cited on 29

21. Foster I. and others . The Open Grid Services Architecture, Version 1.5. GGF Informational Document GFD-I.080, June 2006. Cited on 29

22. Vaquero L. M., Rodero-Merino L., Caceres J., and Lindner M. A Break in the Clouds: Towards a Cloud Definition. ACM Computer Communications Review, 39(1):50–55, Dec. 2008. Cited on 30

23. Zhang Q., Cheng L., and Boutaba R. Cloud Computing: State of the Art and Research Challenges. Journal of Internet Services and Applications, 1(1):7–18, May 2010. Cited on 30

24. Murty J. Programming Amazon Web Services. O'Reilly & Associates, Sebastopol, CA., 2008. Cited on 31, 65

25. Armbrust M., Fox A., Griffith R., Joseph A. D., Katz R. H., Konwinski A., Lee G., Patterson D. A., Rabkin A., Stoica I., and Zaharia M. A View of Cloud Computing. Communications of the ACM, 53(4):50–58, Apr. 2010. Cited on 31

26. Li A., Yang X., Kandula S., and Zhang M. CloudCmp: Comparing Public Cloud Providers. In 10th Internet Measurement Conference, pages 1–14, New York, NY, Nov. 2010. ACM Press. Cited on 32

27. OMG. UML 2.0 Superstructure Specification. OMG Document ptc/04-10-02, Object Management Group, Framingham, MA, Oct. 2004. Cited on 56

28. Shaw M. and Clements P. A Field Guide to Boxology: Preliminary Classification of Architectural Styles for Software Systems. In 21st International Computer Software and Applications Conference, pages 6–13, Aug. 1997. Cited on 56

29. Mehta N., Medvidovic N., and Phadke S. Towards A Taxonomy Of Software Connectors. In 22nd International Conference on Software Engineering, pages 178–187, New York, NY, June 2000. ACM, ACM Press. Cited on 56

30. Fielding R. Architectural Styles and the Design of Network-based Software Architectures. Ph.d., University of California, Irvine, 2000. Cited on 64

31. Pautasso C., Zimmermann O., and Leymann F. Restful Web Services vs. "Big" Web Services: Making the Right Architectural Decision. In 17th International

World Wide Web Conference, pages 805–814, New York, NY, Aug. 2008. ACM Press. Cited on 64, 65

32. Gelernter D. and Carriero N. Coordination Languages and their Significance. Communications of the ACM, 35(2):96–107, Feb. 1992. Cited on 67

33. Frei O., Apishev M. Parallel non-bloking deterministic algorithm for online topic modeling // AIST'2016, Analysis of Images, Social networks and Texts. – Vol. 661. – Springer International Publishing Switzerland, Communications in Computer and Information S ien e (CCIS), 2016. – P. 132–144.

34. Hoffman T. Probabilistic latent semanting indexing // Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval. – New York, NY, USA: ACM, 1999. – Pp. 50-57.

35. Vorontsov K., Frei O., Apishev M., Romov P., Suvorova M. BigARTM: Open source library for regularized multimodal topic modeling of large collections // AIST'2015, Analysis of Images, Social networks and Texts. – Springer International Publishing Switzerland, Communications in Computer and Information Science (CCIS), 2015. – P. 370–384.

36. Kochedykov D., Apishev M., Golitsyn L., Vorontsov K. Fast and modular regularized topic modelling // Proceeding Of The 21St Conference Of FRUCT (Finnish-Russian University Cooperation in Telecommunications) Association. The seminar on Intelligence, Social Media and Web (ISMW). Helsinki, Finland, November 6-10, 2017. – IEEE, 2017. – P. 182–193.

37. Dempster A. P., Laird N. M., Rubin D. B. Maximum likelihood from in omplete data via the EM algorithm // J. of the Royal Statisti al So iety, Series B. – 1977. – no. 34. – Pp. 1–38.

38. Цветкова В.Я., Алпатова А.Н. Проблемы распределенных систем. Журнал «Перспективы науки и образования» Выпуск №6(12)/2014

Приложение А.1
Алгоритм запроса и получения данных системы мониторинга

```
from zabbix.api import ZabbixAPI
from groups import groupids

zapi = ZabbixAPI(url='http://**********/zabbix/',
                 user='********',
                 password='********')

for groupid in groupids:
    print(groupid['groupname'])
    station_events = []
    eventid = 167137276
    while eventid < 232292839:
        problems = zapi.do_request('event.get',
                                   {'groupids': groupid['groupid'],
                                            'eventid_from': eventid,
                                            'eventid_till': eventid+500000,
                                            'selectHosts': ['host']})
        eventid += 500001
        for problem in problems['result']:
            station_events.append(problem['hosts'][0]['host'].split('.')[0]
                                  + '_' +problem['name'])
    document = groupid['groupname'] + ' |text ' + ' '.join(station_events) + '\n'
    f = open('station_events.txt', 'a')
    f.writelines(document)
    f.close()
```

Приложение А.2
Код анализа событий с помощью модели PLSA

```
import artm
from matplotlib import pyplot as plt

bv = artm.BatchVectorizer(data_path="station_events.txt.backup",
                          data_format="vowpal_wabbit",
                          target_folder="station_batches",
                          batch_size=10)
T = 10
model_plsa = artm.ARTM(num_topics=T, dictionary=bv.dictionary,
topic_names=["sbj"+str(i) for i in range(T)], class_ids={"text": 1})

model_plsa.scores.add(artm.PerplexityScore(name='PerplexityScore',
dictionary=bv.dictionary))
model_plsa.scores.add(artm.SparsityPhiScore(name='SparsityPhiScore',
class_id="text"))
model_plsa.scores.add(artm.SparsityThetaScore(name='SparsityThetaScore'))
model_plsa.scores.add(artm.TopTokensScore(name="top_words",
num_tokens=5, class_id="text"))

model_plsa.initialize(dictionary=bv.dictionary)

model_plsa.fit_offline(bv, num_collection_passes=20)

plt.plot(model_plsa.score_tracker["PerplexityScore"].value)

model_plsa.score_tracker["PerplexityScore"].last_value

for topic in model_plsa.score_tracker['top_words'].last_tokens:

print(topic + ': ' + " ".join(model_plsa.score_tracker['top_words'].last_tokens[topic])
+ '\n')

plt.plot(model_plsa.score_tracker["SparsityPhiScore"].value)

model_plsa.score_tracker["SparsityPhiScore"].last_value

plt.plot(model_plsa.score_tracker["SparsityThetaScore"].value)

model_plsa.score_tracker["SparsityThetaScore"].last_value
```

Приложение А.3
Полный вывод метрики TopTokensScore для PLSA-модели

sbj0: sm_Spacelink_Status scu_mysqld_test_high scu_mysql_port_unavailable_high scu_snmp_port_unavailable_high scu_max_data_diff_test_high scu_matrix_disaster scu1_mysqld_status_high scu1_galera_port_1_unavailable_high scu_max_diff_test_warning scu_zbx_api_unavailable_high sw1-mgmt_port18_high rcv2_demodulator_lock_signal_4_disaster esw1-f_SHV-1_SentronAvailability_info srv-pw_FAS_SecurityPinStatus_warning rcv1_demodulator_lock_signal_4_disaster

– sbj1: sm_icmp_check_high srv-pw_AC2_State_info srv-pw_AC1_AC2_State_high enens1_RFT2_1_EtrPcrAccuracyError_warning scu_local_time_high esw-b_SHBP-1_VoltageL1_N_high sm_icmp_check_failed srv-pw_AC1_State_info srv-pw_AC2_AlarmStatus_high srvpw_AC1_AmbientTemp_warning srv-pw_AC2_AmbientTemp_warning srv-pw_AC1_AlarmStatus_high icam3_icmp_check_high sm_Spacelink_Status tsm1_icmp_check_high

sbj2: enens2_IP2_EtrPcrRepErr_warning sw1-mgmt_port14_high esw1-f_SHV-1_VoltageL3_N_warning esw1-f_SHV-1_VoltageL2_N_warning esw1-f_SHV-1_VoltageL1_N_warning enens1_RFT2_1_EtrPcrAccuracyError_warning vb120_cofdm1_PTS_high srv-pw_AC1_State_info tsm1_current_active_TS_input_info ac1_tempIndoor_high tsm2_txSummary.TxB_warning sw1-mgmt_port20_high tsm2_TxExciterSummary.2_warning scu_diff_test_high enens2_IP2_EtrTransportErr_warning

sbj3: esw1-f_SHV-1_PhaseAngleL1_L2_warning esw1-f_SHV-1_PhaseAngleL1_L3_warning esw1-f_SHV-1_VoltageL3_N_warning icam3_icmp_check_high esw1-f_SHV-1_VoltageL1_N_warning esw1f_SHV1_VoltageL2_N_warning sm_Spacelink_Status scu_mariadb_status_high srv-pw_AC1_State_info srv-pw_AC1_AC2_State_high srv-pw_FAS_FaultPinStatus_UPS_warning tsm1_vars_tx_output_statE_TRANSMITTER_OUTPUT_POWER_disaster srvpw_AC2_AmbientTemp_warning scu_local_time_high tsm1_txInpAutoActiveInput.1.1_1_info

sbj4: mnu-vm_slms4ESFrozenErrTrap_high mnuvm_slms4TSvcPMTLossErrTrap_high tsm1_txSummary.TxB_warning scu2_matrix_info tsm1_txSummary.TxB_disaster scu1_matrix_disaster tsm1_txAmpSummary.2_disaster scu2_matrix_disaster drd_signal-lock_warning tsm1_TxExciterSummary.2_warning esw1-g_SHGP-1_VoltageL3_N_high tsm1_TxExciterSummary.2_disaster mnuvm_slms4TSPATLossErrTrap_high esw1-g_SHGP-1_VoltageL1_N_high scu1_matrix_info

sbj5: enens1_RFT2_1_EtrCrcErr_warning dvms t2_measSigInterfacesDvbt2AGCLocked_2_high enens2_IP1_EtrCrcErr_warning dvmst2_measSigInterfacesDvbt2AGCLocked_1_high enens2_IP2_EtrCrcErr_warning dvms-s2_Ccerror_2_0 dvms-t2_CCerror_2_warning dvms-t2_CCerror_1_warning dvms-t2_measSigInterfacesDvbt2LevelValue_1_high dvmst2_measSigInterfacesDvbt2Le

velValue_1_warning dvms-s2_TSsync_1_0 rcv2_fan_failure_module2_warning ops1_faultPinStatus_warning dvms-s2_TSsync_2_0 sw1-mgmt_port17_high

sbj6: dvms-s2_PCRrepetition_2_0 dvms-t2_PCRrepetition_1_warning dvms-s2_PCRrepetition_1_0 dvms-s2_Ccerror_2_0 tsm1_txSummary_warning tsm1_TxExciterSummary.1_warning srv-nmx_afaTSInputCCError_high srv-nmx_afaGenericTransportFault_high srv-nmx_afaFifoOverrun_high dvms-t2_measSigInterfacesDvbt2AGCLocked_2_high dvms-t2_PCRjitter_2_warning dvms-t2_TSsync_2_warning dvms-t2_CCerror_2_warning dvms-t2_PCRrepetition_2_warning dvms-t2_TSerror_2_warning

sbj7: enens2_IP2_EtrPcrAccuracyError_warning enens2_IP1_EtrTsContErr_high enens2_IP2_EtrTsContErr_high enens1_RFT2_1_EtrCrcErr_warning enens1_RFT2_1_EtrPcrAccuracyError_warning enens2_IP2_EtrCrcErr_warning enens2_IP1_EtrCrcErr_warning esw1-f_SHV-1_VoltageL1_N_warning esw1-f_SHV-1_VoltageL3_N_warning esw1-f_SHV-1_VoltageL2_N_warning enens2_IP2_EtrPmtErr2_high enens2_IP1_EtrPmtErr2_high enens1_RFT2_1_EtrTsContErr_high scu_matrix_disaster tsm1_txInpAutoActiveInput.1.1_1_info

sbj8: lsw_DEV-2183_Data_Logger_Amplifier_1_status_disaster enens2_IP1_EtrPcrRepErr_warning enens1_RFT2_1_EtrTsContErr_high esw2-f_SHV-1_SHV-2_PhaseAngleL1_L3_high esw2-f_SHV 1_SHV2_PhaseAngleL1_L2_high enens2_IP1_EtrTsContErr_high enens2_IP2_EtrTransportErr_warning esw-ng_SHNP-1_VoltageL2_N_high esw-ng_SHNP-1_VoltageL1_N_high enens2_IP2_EtrPcrDiscIndError_warning esw-ng_SHNP-1_VoltageL3_N_high enens1_RFT2_1_EtrPcrRepErr_warning enens2_IP2_EtrPmtErr2_high srvpw_AC2_AmbientTemp_warning enens2_IP2_EtrTsPatErr2_high

sbj9: enens1_RFT2_1_EtrTsContErr_high enens1_RFT2_1_EtrTransportErr_warning enens1_RFT2_1_EtrPcrRepErr_warning enens1_RFT2_1_EtrPcrDiscIndError_warning enens1_RFT2_1_EtrTsSyncLoss_disaster enens1_RFT2_1_RfSignalLevelError_high enens1_RFT2_1_EtrPmtErr2_high enens1_RFT2_1_EtrPidErr_high enens1_RFT2_1_EtrTsPatErr2_disaster enens1_RFT2_1_EtrPcrAccuracyError_warning enens2_IP1_EtrPidErr_high enens2_IP1_EtrTransportErr_warning enens2_IP1_EtrPcrDiscIndError_warning enens1_RFT2_1_RfFeLockStatusError_high enens1_RFT2_1_EtrCrcErr_warning

Приложение А.4
Код анализа событий с помощью модели ARTM

```
from matplotlib import pyplot as plt
import artm

bv = artm.BatchVectorizer(data_path="station_events.txt.backup",
                          data_format="vowpal_wabbit",
                          target_folder="station_batches",
                          batch_size=10)
T = 10
model_artm = artm.ARTM(num_topics=T, dictionary=bv.dictionary,
topic_names=["sbj"+str(i) for i in range(T)], class_ids={"text": 1})

model_artm.scores.add(artm.PerplexityScore(name='PerplexityScore',
dictionary=bv.dictionary))
model_artm.scores.add(artm.SparsityPhiScore(name='SparsityPhiScore',
class_id="text"))
model_artm.scores.add(artm.SparsityThetaScore(name='SparsityThetScore'))
model_artm.scores.add(artm.TopTokensScore(name="top_words",
num_tokens=5, class_id="text"))
model_artm.initialize(dictionary=bv.dictionary)
model_artm.regularizers.add(artm.SmoothSparsePhiRegularizer(name='Spars
ePhi', tau=-10000, dictionary=bv.dictionary))
model_artm.regularizers.add(artm.SmoothSparseThetaRegularizer(name='Sp
arseTheta', tau=-100))
model_artm.fit_offline(bv, num_collection_passes=20)
plt.plot(model_artm.score_tracker["PerplexityScore"].value)
model_artm.score_tracker["PerplexityScore"].last_value
for topic in model_artm.score_tracker['top_words'].last_tokens:
print(topic + ': ' + "
".join(model_artm.score_tracker['top_words'].last_tokens[topic]) + '\n')

plt.plot(model_artm.score_tracker["SparsityPhiScore"].value)

model_artm.score_tracker["SparsityPhiScore"].last_value

plt.plot(model_artm.score_tracker["SparsityThetaScore"].value)

model_artm.score_tracker["SparsityThetaScore"].last_value
```

Приложение А.5
Полный вывод метрики TopTokensScore для ARTM-модели

sbj0: sm_Spacelink_Status scu_mysqld_test_high scu_mysql_port_unavailable_high scu_snmp_port_unavailable_high scu_max_data_diff_test_high scu1_mysqld_status_high scu1_galera_port_1_unavailable_high scu_matrix_disaster scu_max_diff_test_warning scu_zbx_api_unavailable_high

sbj1: sm_icmp_check_high srv-pw_AC2_State_info srv-pw_AC1_AC2_State_high scu_local_time_high enens1_RFT2_1_EtrPcrAccuracyError_warning esw-b_SHBP-1_VoltageL1_N_high sm_icmp_check_failed srv-pw_AC1_State_info srv-pw_AC2_AlarmStatus_high srv-pw_AC1_AmbientTemp_warning

sbj2: enens2_IP2_EtrPcrRepErr_warning sw1-mgmt_port14_high esw1-f_SHV-1_VoltageL3_N_warning esw1-f_SHV-1_VoltageL2_N_warning esw1-f_SHV-1_VoltageL1_N_warning enens1_RFT2_1_EtrPcrAccuracyError_warning srv-pw_AC1_State_info vb120_cofdm1_PTS_high tsm1_current_active_TS_input_info ac1_tempIndoor_high

sbj3: esw1-f_SHV-1_PhaseAngleL1_L2_warning esw1-f_SHV-1_PhaseAngleL1_L3_warning esw1-f_SHV-1_VoltageL3_N_warning icam3_icmp_check_high esw1-f_SHV-1_VoltageL1_N_warning esw1-f_SHV-1_VoltageL2_N_warning sm_Spacelink_Status scu_mariadb_status_high srv-pw_AC1_State_info srv-pw_AC1_AC2_State_high

sbj4: mnu-vm_slms4ESFrozenErrTrap_high mnu-vm_slms4TSvcPMTLossErrTrap_high tsm1_txSummary.TxB_warning scu2_matrix_info tsm1_txSummary.TxB_disaster scu1_matrix_disaster tsm1_txAmpSummary.2_disaster scu2_matrix_disaster drd_signal-lock_warning tsm1_TxExciterSummary.2_warning

sbj5: enens1_RFT2_1_EtrCrcErr_warning dvms-t2_measSigInterfacesDvbt2AGCLocked_2_high enens2_IP1_EtrCrcErr_warning dvms-t2_measSigInterfacesDvbt2AGCLocked_1_high enens2_IP2_EtrCrcErr_warning dvms-s2_Ccerror_2_0 dvms-t2_CCerror_2_warning dvms-t2_CCerror_1_warning dvms-t2_measSigInterfacesDvbt2LevelValue_1_high dvms-t2_measSigInterfacesDvbt2LevelValue_1_warning

sbj6: dvms-s2_PCRrepetition_2_0 dvms-t2_PCRrepetition_1_warning dvms-s2_PCRrepetition_1_0 dvms-s2_Ccerror_2_0 tsm1_txSummary_warning tsm1_TxExciterSummary.1_warning srv-nmx_afaTSInputCCError_high srv-nmx_afaGenericTransportFault_high srv-nmx_afaFifoOverrun_high dvms-t2_measSigInterfacesDvbt2AGCLocked_2_high

sbj7: enens2_IP2_EtrPcrAccuracyError_warning enens2_IP1_EtrTsContErr_high enens2_IP2_EtrTsContErr_high enens1_RFT2_1_EtrCrcErr_warning enens1_RFT2_1_EtrPcrAccuracyError_warning enens2_IP2_EtrCrcErr_warning

enens2_IP1_EtrCrcErr_warning esw1-f_SHV-1_VoltageL1_N_warning esw1-f_SHV-1_VoltageL3_N_warning esw1-f_SHV-1_VoltageL2_N_warning

sbj8: lsw_DEV-2183_Data_Logger_Amplifier_1_status_disaster enens2_IP1_EtrPcrRepErr_warning enens1_RFT2_1_EtrTsContErr_high esw2-f_SHV-1_SHV-2_PhaseAngleL1_L3_high esw2-f_SHV-1_SHV-2_PhaseAngleL1_L2_high enens2_IP1_EtrTsContErr_high enens2_IP2_EtrTransportErr_warning esw-ng_SHNP-1_VoltageL2_N_high esw-ng_SHNP-1_VoltageL1_N_high enens2_IP2_EtrPcrDiscIndError_warning

sbj9: enens1_RFT2_1_EtrTsContErr_high enens1_RFT2_1_EtrTransportErr_warning enens1_RFT2_1_EtrPcrRepErr_warning enens1_RFT2_1_EtrPcrDiscIndError_warning enens1_RFT2_1_EtrTsSyncLoss_disaster enens1_RFT2_1_RfSignalLevelError_high enens1_RFT2_1_EtrPmtErr2_high enens1_RFT2_1_EtrPidErr_high enens1_RFT2_1_EtrTsPatErr2_disaster enens1_RFT2_1_EtrPcrAccuracyError_warning