МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«КУБАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
(ФГБОУ ВО «КубГУ»)

Факультет компьютерных технологий и прикладной математики
Кафедра анализа данных и искусственного интеллекта

Допустить к защите
заведующий кафедрой
д-р тех. наук, доцент
_____ А. В. Коваленко
_____ 2021 г.

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА (БАКАЛАВРСКАЯ РАБОТА)

РАЗРАБОТКА АВТОМАТИЗИРОВАННОЙ СИСТЕМЫ ОБНАРУЖЕНИЯ АНОМАЛИЙ В ДАННЫХ МОНИТОРИНГА

Работу выполнил_____А. Р. Чениб
(подпись)

Направление подготовки 02.03.03 Математическое обеспечение и администрирование информационных систем

Направленность (профиль) Технология программирования

Научный руководитель
доктор тех. наук, доц. _____А. В. Коваленко
(подпись)
Нормоконтролер
канд. физ.-мат. наук, доц. _____Г. В. Калайдина
(подпись)

Краснодар
2021

РЕФЕРАТ

Выпускная квалификационная работа 46 с., 5 ч., 25 рис., 1 табл., 17 источников.

АНОМАЛИИ, РАБОТА СЕРВЕРОВ, СИСТЕМЫ МОНИТОРИНГА, АНАЛИЗ ВРЕМЕННЫХ РЯДОВ, ПРОГНОЗИРОВАНИЕ, НЕЙРОННЫЕ СЕТИ, ЯЗЫК ПРОГРАММИРОВАНИЯ PYTHON

Объектом исследования являются аномалии в показателях работы серверов.

Цель выпускной квалификационной работы – разработка системы обнаружения аномалий в данных мониторинга сервера, используя методы машинного обучения и прогнозирования.

В выпускной квалификационной работе рассмотрены основные показатели работы серверов, проведен сравнительный анализ систем мониторинга вычислительных систем, настроена система мониторинга Zabbix. Также были рассмотрены основные виды аномалий и методы их обнаружения. Разработана система обнаружения аномалий с использованием авторегрессионной интегрированной скользящей средней (ARIMA) и нейронной сети с долгой краткосрочной памятью (LSTM).

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Одним из проявлений процесса информатизации общества является масштабное развитие сетевых сервисов. В связи с этим возрастает необходимость такой задачи администрирования информационных систем, как обеспечение надежности работы системы. К этому относится своевременная проверка состояния оборудования, поддержание оптимальной работы ПО, обеспечение сохранности данных. Эта задача решается в сферах компьютерной безопасности и системного администрирования. Ощутимый вклад в решении этих проблем вносят методы анализа данных. Осуществляя обработку служебных данных о статистике, предоставляемых системами мониторинга, данные инструменты реализуют проверку различных метрик работы информационной системы на предмет достижения аномальных значений.

Целью данной работы является разработка системы по обнаружению аномалий в показателях работы серверов. Для этого с поставлены следующие задачи: рассмотреть основные виды возможных аномалий в их работе и известные методы обнаружения аномалий, настроить мониторинг виртуального сервера, построить модели прогнозирования временных рядов и разработать с использованием реализованных моделей программу по обнаружению аномалий.

В работе было реализовано обнаружение аномалий для данных, собираемых системой мониторинга Zabbix. Для разработки системы были использованы такие модели прогнозирования временных рядов, как авторегрессионная интегрированная скользящая средняя (ARIMA) и нейронная сеть с долгой краткосрочной памятью (LSTM).

1 Сведения о предметной области

В настоящее время информационные системы становятся более масштабными, и объем инфраструктуры может достигать сотен и тысяч серверов. В результате этого наблюдение за состоянием работы систем является затруднительным процессом, связанным с работой с многочисленными данными информационно-вычислительных систем.

Основными средствами обеспечения автоматизации работы с подобными данными являются системы мониторинга, такие, как Zabbix, Nagios и др.

Наиболее широким определением мониторинга является мониторинг вычислительной сети. Под ним понимается постоянное наблюдение в пределах сети, в том числе анализ трафика и его диагностика, загрузка сетевых устройств и соединений, с целью поиска медленных или неисправных систем, а также оповещение сетевых администраторов о сбоях и иных неисправностях с использованием различных средств оповещения.

Своевременное выявление сбоев позволяет дать комплексную оценку работы системы на основе анализа функционирования и производительности её отдельных элементов – объектов мониторинга.

Под объектом мониторинга понимаются устройство либо служба, за которым осуществляется регулярное наблюдение с целью контроля над его состоянием, анализа протекающих с его участием процессов, выявления и прогнозирования нештатных состояний.

1.1 Основные показатели работы сервера

Мониторинг серверов – это процесс сбора и анализа данных о текущей производительности вычислительных систем. При мониторинге и анализе производительности серверов важно использовать системный подход и учитывать взаимное влияние аппаратных компонентов. Анализ

загруженности сервера заключается в сборе и обработке статистики следующих компонент:

- процессор;
- память;
- диск;
- сетевой интерфейс.

Рассмотрим подробнее характеристики наблюдаемых компонент.

1.1.1 Процессор

Процессорная подсистема, прежде всего, характеризуется показателями загруженности (процентом утилизации) центрального процессора (ЦП):

1) CPU usage (использование процессора) – процент времени на выполнение пользовательских задач и процент системного времени.

Счетчик включает в себя процессорное время, затрачиваемое на обработку аппаратных прерываний. Основной индикатор общего использования процессора. Значения варьируются по шкале от 0 до 100%.

Процент загруженности процессора – это процент времени, затраченного процессором на выполнение любого потока, кроме потока бездействия. Для вычисления этого значения измеряется процент времени, затраченного процессором на выполнение потока бездействия, а затем полученное значение вычитается из 100%. У каждого процессора есть поток бездействия, на который расходуется время, если отсутствуют другие потоки, готовые к выполнению. Этот счетчик является основным индикатором активности процессора и показывает средний процент времени занятости за определенное время. Следует отметить, что учет использования ресурсов процессором выполняется через внутренние интервалы, равные тактам системных часов. По этой причине в современных быстрых процессорах процент загруженности процессора может быть занижен, так

как процессор может затрачивать много времени на обработку потоков между соседними тактами системных часов.

2) CPU interrupts (прерывания процессора) – время на ожидание ввода-вывода блочных устройств. Состоит в том, что процессор прерывает обработку текущей программы (прерываемой программы) и переходит к выполнению некоторой другой программы (прерывающей программы), специально предназначенной для данного события. По завершении этой программы процессор возвращается к выполнению прерванной программы. Измеряется в относительном и абсолютном значении.

Процент времени прерываний – это доля времени выборочного интервала, которую процессор тратит на обработку аппаратных прерываний. Эта величина является косвенным показателем активности устройств, формирующих аппаратные прерывания, таких как системного таймера, мыши, драйверов дисков, линий передачи данных, сетевых адаптеров и других периферийных устройств. В этом режиме могут быть запущены только подпрограммы обслуживания прерываний (ISR, которые являются функциями драйверов устройств).

В отличие от процента абсолютное значение показывает количество прерываний в секунду. Поскольку прерывания принимаются от оборудования, высокие значения косвенно могут свидетельствовать о проблемах с устройствами или с их драйверами. Особенно актуальны показания счетчика после физического добавления нового устройства на сервер.

Средняя скорость прерываний, в событиях в секунду, с которой процессор получает и обслуживает аппаратные прерывания, не включает отложенные вызовы процедур, которые подсчитываются отдельно. Является косвенным показателем активности устройств, формирующих аппаратные прерывания, обычно при завершении своей работы или при возникновении необходимости обработки запроса. При этом обычное выполнение потока команд приостанавливается. Системный таймер обычно прерывает работу

процессора каждые 10 миллисекунд. Поэтому эта величина отображает разницу между значениями последних двух выборок, поделенную на длительность интервала выборки.

Абсолютные показания нужно анализировать вместе с процентом и при большом значении последнего делать вывод о существовании каких-либо проблем.

3) System load (процессорное время) – время, затраченное процессором компьютера на обработку задачи (программы).

Распределяется между процессами в соответствии с используемым режимом операционной системы. Процессорное время измеряется в тиках или секундах. Часто бывает полезно измерение процессорного времени в процентах от мощности процессора, который называется загрузкой процессора.

Расчет процессорного времени заключается в количественном измерении общей занятости системы. Высокая загрузка ЦП указывает на недостаточную вычислительную мощность. Либо вычислительная мощность процессора (процессоров) должна быть повышена или объём пользовательских задач должен быть уменьшен, например, путём применения конфигураций с более низкими требованиями.

1.1.2 Оперативная память

Производительность оперативной памяти зависит от множества факторов, однако ключевым среди них является счетчик доступных байтов, характеризующийся как в количественном (в мегабайтах), так и в процентном (% свободной памяти) измерении.

Низкие показатели означают нехватку памяти, в процентах – постоянное значение ниже 20–25% установленной оперативной памяти.

1) Memory usage – отслеживает количество доступной памяти в байтах для выполнения различных процессов. Объем физической памяти,

доступный для выделения. Если памяти недостаточно, файл подкачки будет использоваться более интенсивно, а число ошибок страниц в секунду увеличится. Рассчитывается путём сложения объёмов обнулённой, свободной и простаивающей памяти. Свободная память готова к использованию; обнуленная память – страницы памяти, заполненные нулями, чтобы каждый следующий процесс не имел доступа к данным, которые использовал предыдущий; простаивающая память – это память, изъятая из рабочего множества процесса (его физической памяти) и предназначенная для перемещения на диск, но она может быть вновь запрошена и использована без необходимости чтения данных с диска.

2) Memory utilization – рассчитывает процентное отношение объема выделенной памяти к пределу выделенной памяти. Эта величина отражает реально используемый объем доступной виртуальной памяти. Необходимо учитывать, что предел выделенной памяти может быть изменен, если файл подкачки будет увеличен. Представляет собой конкретное текущее значение, и не является средним значением по некоторому интервалу времени.

3) Swap usage (использование файла подкачки). Когда оперативная память заполнена полностью, система начинает заполнять файл подкачки (swap) – виртуальная память, которая позволяет одновременно выполняться большому количеству процессов, которые все сразу не смогли бы поместиться в физической памяти. Мониторинг использования swap позволяет проконтролировать необходимость увеличения физической памяти и правильно настроить файл подкачки для увеличения производительности.

1.1.3 Дисковая подсистема

Дисковая подсистема – одна из важных подсистем сервера, и от уровня нагрузки на дисковую подсистему зачастую зависят скоростные характеристики работы системы. Это в большей степени относится к

почтовым или файловым серверам, серверам баз данных. Ключевыми являются следующие метрики:

1) Disk average waiting time (отзывчивость устройства) – первичный индикатор быстродействия ввода/вывода информации на физическом диске, средняя длительность ожидания (в миллисекундах) для всех запросов блокировки, при которых потребовалось ожидание. Этот счетчик показывает, сколько в среднем процессам пользователей приходится проводить в очереди, чтобы наложить на ресурс блокировку. Сюда входят и время обработки, и время ожидания в очереди. Максимально допустимое значение этого счетчика полностью зависит от поставленной задачи.

Производительность зависит от конфигурации дисковой подсистемы, которая невидима для операционной системы. Влияние отдельного диска на продуктивность работы основано на его времени поиска, скорости вращения, плотности записи и скорости интерфейса. Показания этого счётчика могут указывать на высокую фрагментированность диска, его низкую скорость или сбой в его работе.

2) Disk read/write rate (текущие значения чтения/записи на устройство) – скорость чтения/записи на устройство в килобайтах. Соответственно read rate представляет собой количество килобайт, прочитанных с устройства в секунду, write rate – количество килобайт, записанных на устройство в секунду.

3) Disk utilization (утилизация) – процент времени, в течение которого диск был занят обслуживанием запросов чтения или записи.

4) Disk average queue size (текущая средняя длина очереди) – количество невыполненных запросов на диске на момент сбора данных о производительности. Время ожидания запроса в очереди может экспоненциально расти по мере того, как дисковая утилизация приближается к 100%, если это независимы запрос к диску.

1.1.4 Сетевые интерфейсы

Главным счетчиком производительности сетевых интерфейсов является показатель общей скорости в байт/с (Network traffic). В процессе мониторинга счетчик скорости сопоставляется с принятым базовым уровнем. Если сетевой обмен значительно возрастает по сравнению с базовым уровнем, то есть нагрузка на сервер существенно возросла по сравнению с нагрузкой, которая наблюдалась при определении базового уровня, то это может служить индикатором сетевой атаки или показывать необходимость разгрузки некоторых процессов.

Возможность обнаружения аномалий в мониторинге рассмотренных компонентов сервера позволяет упростить анализ нагрузки на сервер. Главным образом, станет возможным уменьшение сроков аудита работоспособности серверов, а также ускорение принятия решений по оптимизации их конфигурации.

1.2 Анализ существующих систем мониторинга сетевых ресурсов

Своевременное выявление и устранение неисправностей является залогом производительного функционирования любой вычислительной сети. Обнаружение сбоев является комплексной задачей, требующей анализа работы и взаимодействия программных и аппаратных компонентов системы, которая решается с помощью систем мониторинга.

Система мониторинга – это совокупность технических средств, осуществляющих постоянное наблюдение и сбор информации в локальной вычислительной сети на основе анализа статистических данных с целью выявления неисправных или некорректно работающих узлов и оповещения ответственных лиц [1].

Современные платформы, обеспечивающие устойчивое обнаружение аномалий в большой системе в режиме реального времени, можно

классифицировать по характеристикам, представленным на рисунке 1 [2].

Рисунок 1 – Классификация систем сетевого мониторинга

В процессе сравнения современных систем сетевого мониторинга из представленной классификации будут использованы такие параметры как способ реализации сбора данных и область применения, так как по характеру анализируемых данных все они в основном применяют наиболее точный статистический подход к анализу данных мониторинга.

Представим характеристику рассматриваемых систем мониторинга сети и проведем их сравнительный анализ.

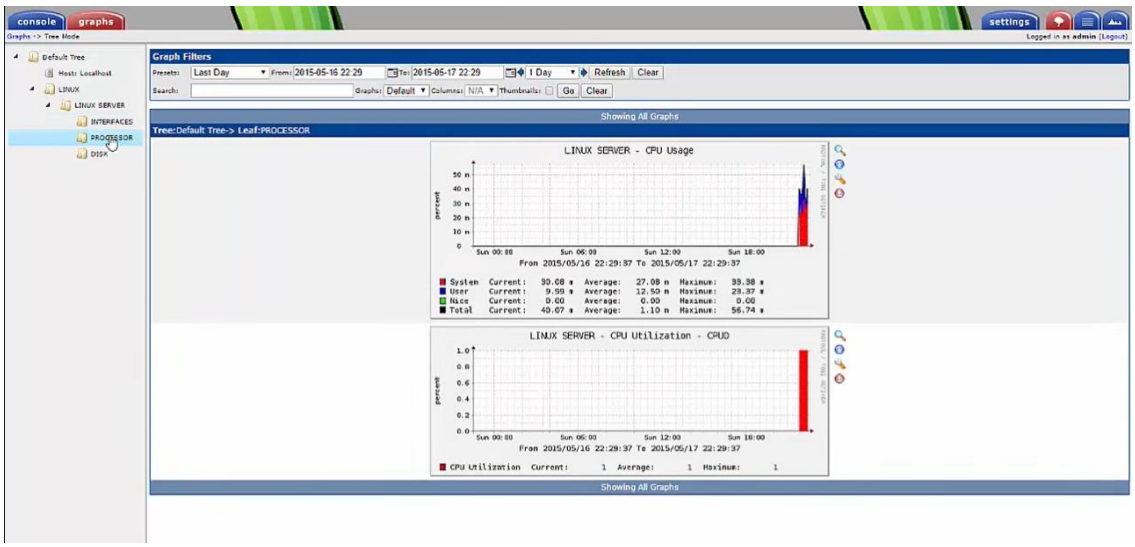


Рисунок 2 – Интерфейс системы Cacti

Cacti — это веб-инструмент мониторинга сети с открытым исходным кодом, разработанный как интерфейсное приложение (рис. 2). Анализ данных мониторинга в Cacti строится на сборе статистических показателей за интервал времени и позволяет визуализировать их в графическом виде. Главным образом применяются шаблоны для контроля над такими индикаторами, как процент загруженности процессора (CPU usage), производительность оперативной памяти (Memory usage) и использованию сетевого трафика (Network traffic) посредством опроса сетевых интерфейсов маршрутизатора или коммутатора.

Часто используется поставщиками веб-хостинга ввиду возможности интерфейса обеспечить обработку данных для оценки пропускной способности сети нескольких пользователей с индивидуальным набором графиков индикаторов. Интерфейс отображения статистики, собранной с сетевых устройств, представлен в виде дерева, структура которого задается самим пользователем. При этом с помощью сценариев оболочки и исполняемых файлов функционал Cacti можно применить к любому источнику.

Еще одним неоспоримым преимуществом Cacti следует назвать функцию хранения ретро-данных по каждому конкретному индикатору для

возможности его сопоставления с текущими значениями. Период для сравнения, как правило, задается в промежутках день, неделя, месяц или год. Также возможно сгенерировать график по заданным календарным параметрам, либо по выбранному интервалу временного ряда.

Изначально этот инструмент позиционировался для быстрой и простой сборки статистики, но с применением плагинов приобрел возможности системы мониторинга. Подходит для массового универсального использования ввиду большого количества базовых шаблонов для проверки различных типов устройств, однако для сбора сложных и нетипичных метрик неприменим.

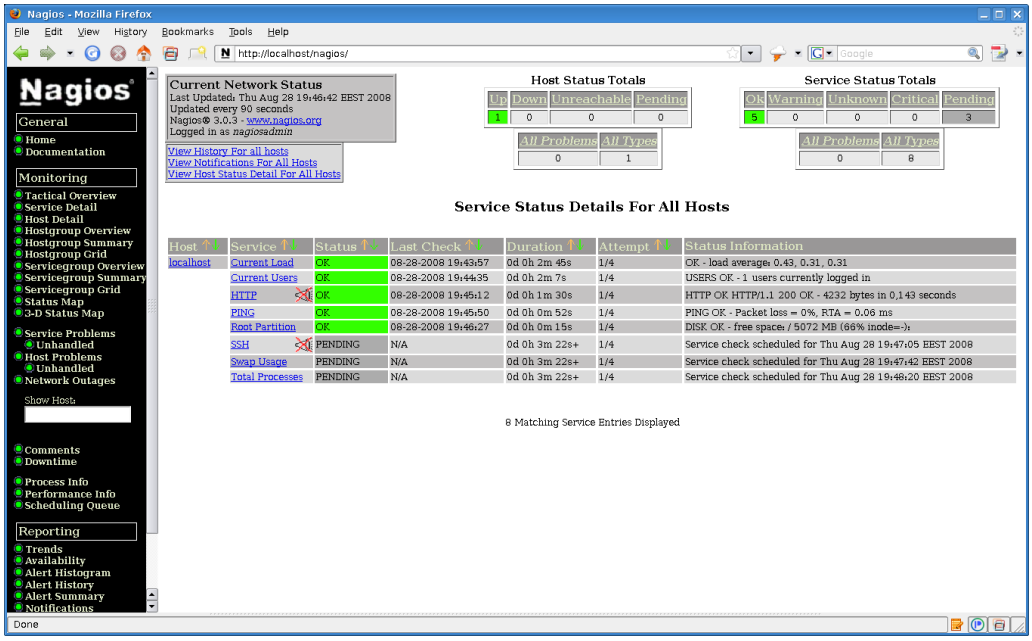


Рисунок 3 – Интерфейс системы Nagios

Nagios – это лицензируемое программное обеспечение с открытым исходным кодом, обеспечивающее непрерывный мониторинг серверов, сетевого оборудования, сервисов, коммутаторов и приложений с помощью широкого перечня инструментов (рис. 3). Предназначен для наблюдения, контроля состояния вычислительных узлов и служб, оповещает администратора в том случае, если какие-то из служб прекращают (или возобновляют) свою работу.

Первоначально Nagios был разработан для работы под Linux, но он также работает и под другими Unix-подобными системами. В сравнении с описанным выше Cacti является более прогрессивным средством мониторинга, для чего применятся сложный инструментарий настройки системы, требующий понимания принципов его устройства. При этом с учетом наличия базовых шаблонов, плюсом является высокая гибкость настройки параметров идентификации аномалий для индивидуального структурирования системы с учетом потребностей конкретного пользователя.

Также имеется возможность удаленного контроля над нагрузкой ключевых компонентов системы (процессор, оперативная память, диски и сетевой интерфейс): проверка процессов, проверка работоспособности сервера, его запуска, обработка статистики, формирование отчетов и т. д.

Особое значение имеет обнаружение сетевых аномалий и система оповещения о них: настройка уведомлений об отказе сервера на электронную почту или через мобильного оператора, возможность настройки уведомлений определенных событий только для назначенной группы или конкретного пользователя. Широкий диапазон инструментов: интегрируется с приложениями, работает на разных операционных системах, несколько серверов баз данных.

В попытках исправить недостатки и усовершенствовать функционирование Nagios была создана система *Icinga*, имеющая современный пользовательский интерфейс, дополнительные соединители для баз данных, что позволяет администраторам добавлять множество расширений без внесения изменений в ядро.

Так как Icinga это ответвление Nagios, то их функции мониторинга аналиогичны, с некоторыми дополнениями, такими как дополнительные модули отчетности с улучшенной точностью, дополнительные соединители для систем управления базами данных Oracle и PostgreSQL и распределенные системы вычисления для избыточного мониторинга. Так же, для упрощения

миграции между системами мониторинга, Icinga совместима с плагинами Nagios.

Отличительными чертами этой системы мониторинга являются [3]:

– наличие реализации API (REST) для управления системой, получения информации о состоянии устройств и сервисов или отправки информации в саму систему;

– гибкая модель построения конфигураций мониторинга, удовлетворяющих практически любые потребности;

– гибкая архитектурная реализация, позволяющая создавать от самых простых до сложных территориально-распределенных систем мониторинга;

– работа в полном безагентном режиме или с различными клиентскими решениями с защитой канала передачи данных между клиентом и сервером;

– возможность взаимодействия со специализированным дополнительным ПО (визуализация метрических данных, системы обработки заявок и т. д.).

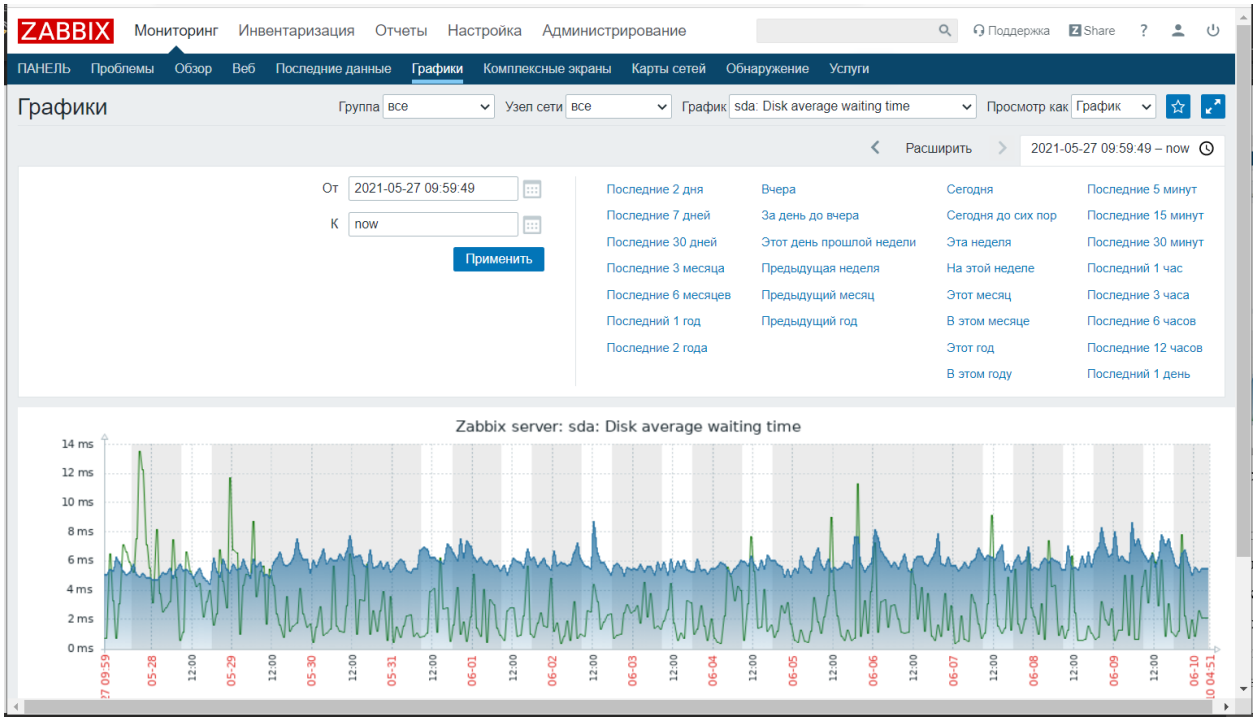


Рисунок 4 – Интерфейс системы Zabbix

Zabbix — это свободное программное обеспечение для мониторинга многочисленных параметров сети, жизнеспособности и целостности серверов. В основном оно предназначено для систем, которые обладают многосерверной архитектурой – одновременное управление сотнями сетевых узлов позволяет обеспечить эффективный мониторинг на крупномасштабных предприятиях. Существенными достоинствами являются возможность создания персональных шаблонов и сценариев, а также множество поддерживаемых плагинов. В качестве недостатков можно выделить сложный пользовательский интерфейс и высокую аппаратную нагрузку. Среди самых полезных сервисов выделяют комплексный мониторинг индикаторов производительности всех компонентов серверов, визуализация полученной информации о значениях их индикаторов в диаграммы и информирование администратора оповещениями.

Еще одной достаточно простой и при этом функциональной системой сетевого мониторинга является PRTG. Не требует установки агентов на объекты мониторинга и использует стандартные протоколы и технологии, например, SNMP. Также является более узкоориентированной системой с удобным графическим интерфейсом, предназначенной конкретно для сетевого мониторинга, а не всей инфраструктуры. Аналогично предыдущим данная система может производить инспекцию пакетов, анализ и сохранение статистических данных в базу, просмотр карты сети в режиме реального времени (также доступна возможность получения исторических сведений о поведении сети), сбор технических параметров об устройствах, подключенных к сети, а также анализ уровня нагрузки на сетевое оборудование.

Выделим ключевые параметры необходимые для балльной оценки (наличие функции по параметру 1 балл, отсутствие 0 баллов) описанных систем:

1) прогнозирование тенденций – наличие алгоритмов, предназначенных для прогнозирования сетевой статистики;

2) плагины – наличие плагинов, обеспечивающих дополнительную функциональность;

3) оповещения – способность обнаружения и предупреждения об аномалиях;

4) полный контроль – наличие веб-интерфейса, обеспечивающего возможность доступа ко всем сервисам системы;

5) распределенный мониторинг – возможность использования множества серверов.

Синтезируя приведенную информацию о рассматриваемых системах сетевого мониторинга, представим комплексную оценку их параметров в таблице 1.

Таблица 1 – Оценка характеристик систем сетевого мониторинга

Параметр оценки	**Cacti**	**Icinga**	**Nagios**	**PRTG**	**Zabbix**
Прогнозирование тенденций	+	-	-	+	+
Плагины	+	+	+	+	+
Оповещения	+	+	+	+	+
Полный контроль	+	+	-	-	+
Распределенный мониторинг	+	+	+	+	+
Итоговая оценка	5	4	3	4	5

По результатам проведенной оценки наиболее многофункциональными являются системы Cacti и Zabbix, которым содержат в себе весь перечень анализируемых инструментов сетевого мониторинга. Однако следует подчеркнуть преимущества Zabbix в работе с большим числом сетевых узлов и персональную настройку систем сканирования, в отличие от шаблонного Cacti. Отсутствие возможности прогнозирования тенденций временных рядов на основе ретро-данных достаточно весомый недостаток Icinga и Nagios, что ограничивает пользователя в планировании использования сетевых компонентов в перспективе. Чуть менее существенным на наш взгляд является лишь частичный контроль через веб-интерфейс в системах

PRTG и Nagios, так как отражает лишь удобство (что не сказывается на возможности) применения инструментов для пользователя. Таким образом, необходимо признать, несмотря на равное количество баллов, PRTG имеет преимущество относительно Icinga. В то же время Icinga обладает большим набором механизмов мониторинга сети относительно своего прародителя Nagios.

2 Классификация аномалий

В общем случае аномалия – это отклонение от ожидаемого поведения. При исследовании аномалий в контексте данных системного и сетевого мониторинга, они могут рассматриваться как изменения в поведении работы системы и ее показателях во времени [4].

Существуют различные классификации аномалий [5]. Например, аномалии могут иметь 3 типа: точечная, контекстуальная, коллективная.

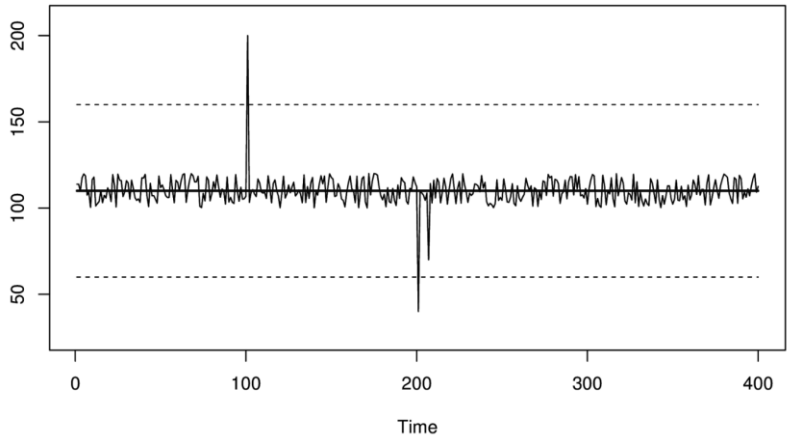


Рисунок 5 – Точечная аномалия

При точечной аномалии (рис. 5) происходит единичный выброс измерения, проявляющийся в большом отклонении от всех остальных измерений. Обнаружение точечных аномалий может быть выполнено нахождением математического ожидания и стандартного отклонения. Показатели, превышающие определенный порог после стандартного отклонения, считаются аномальными.

Контекстная аномалия возможна, когда измерение аномально лишь в определенном контексте. Например, если на сервере выполняется резервное копирование в ночное время по расписанию, то повышение процента загруженности процессора ночью считается нормой, а в дневное время – аномальной. На рисунке 6 этот промежуток отмечен 6 декабря с 0 до 12

часов. Здесь требуется помимо поведения измерения определять контекст, в частности величину промежутка, относительно значений в котором измерение называется аномальным.

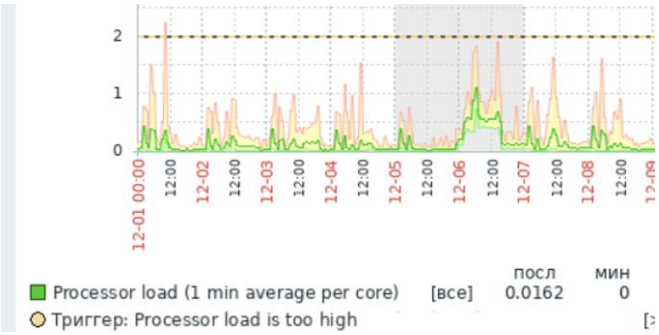


Рисунок 6 – Контекстная аномалия

В случае коллективных аномалий последовательность связанных экземпляров данных (например, участок временного ряда) является аномальной по отношению к целому набору данных. Среди коллективных аномалий можно выделить аномалии сдвига, изменяющие математическое ожидание во временном ряде (рис. 7), и аномалии изменения распределения, проявляющиеся в изменении среднеквадратического отклонения (рис. 8) [6].

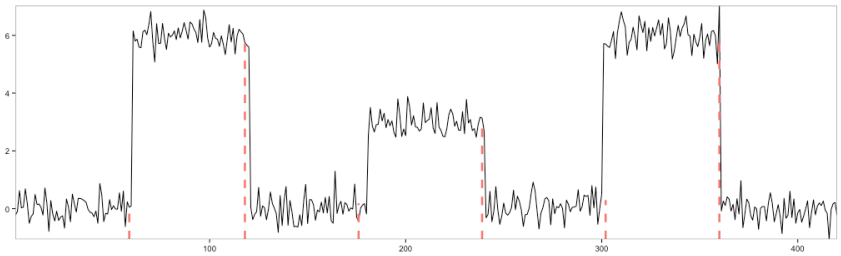


Рисунок 7 – Аномалии сдвига

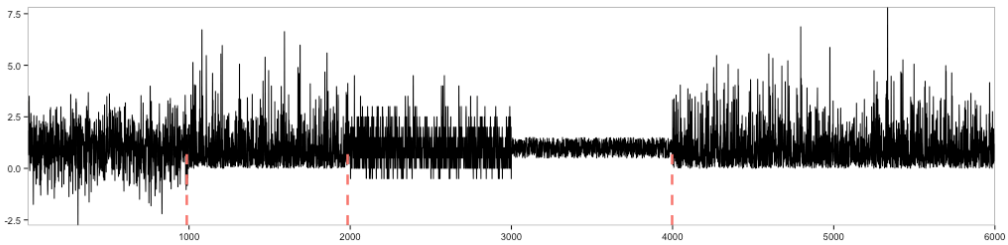


Рисунок 8 – Аномалии изменения распределения

3 Обзор методов обнаружения аномалий

Как отмечено в книге [4], обнаружение аномалий – это совокупность методов и систем по поиску необычного поведения и необычных состояний в системах и их наблюдаемых показателях.

Среди методов обнаружения аномалий можно выделить статистические и методы машинного обучения.

В свою очередь, методы машинного обучения подразделяются на методы с учителем (Дерево решений, SVM, LSTM) и методы без учителя (K-средних, иерархическая кластеризация, DBSCAN). Методы с учителем требуют обучающую выборку для применения модели, отличающей норму от аномалии.

Методы машинного обучения могут находить более сложные аномалии, чем статистические методы. Так, с помощью нейронных сетей возможно обнаружение и классификация сетевых атак [7].

Из методов машинного обучения можно выделить следующие:

LSTM (нейронная сеть с долгой краткосрочной памятью) – одна из архитектур рекуррентных нейронных сетей. По входному временному ряду предсказывает значение на следующем временном промежутке посредством сохранения состояния. Обучается на нормальных данных временного ряда, и по результату определяется аномалия.

SVM (метод опорных векторов) – метод классификации, который переводит исходные вектора, представляющие значения, в пространство более высокой размерности и находит разделяющую гиперплоскость с максимальным зазором в этом пространстве, которая разделяет нормальные значения от аномальных.

Метод K-средних – алгоритм кластеризации, относящийся к методам машинного обучения без учителя, идеей которого является объединение данных в кластеры вокруг главных точек, количество которых задается

заранее. Элементы, которые не были включены ни в один кластер, считаются аномальными.

К статистическим методам обнаружения аномалий можно отнести:

– модель Хольта-Винтерса [8];

– авторегрессионное интегрированное скользящее среднее (ARIMA).

Статистические методы основаны на идее, что нормальные временные ряды генерируются из статистического процесса, а значения, не соответствующие этому процессу, считаются аномальными [9]. Ключевой составляющей данных методов является изучение параметров статистического процесса на обучающем временном ряде и оценка соответствия тестового временного ряда полученным параметрам.

При использовании статистических методов анализа учитывается тот факт, что временной ряд может быть разбит на три компоненты (рис. 9) [10]:

– тренд – отражает общее поведение ряда в плане возрастания или убывания значений;

– сезонность – периодичные колебания значений, связанные, например, с днем недели или месяца;

– случайное значение – данные, которые останутся от ряда после исключения других компонент; в них обычно осуществляется поиск аномалии.

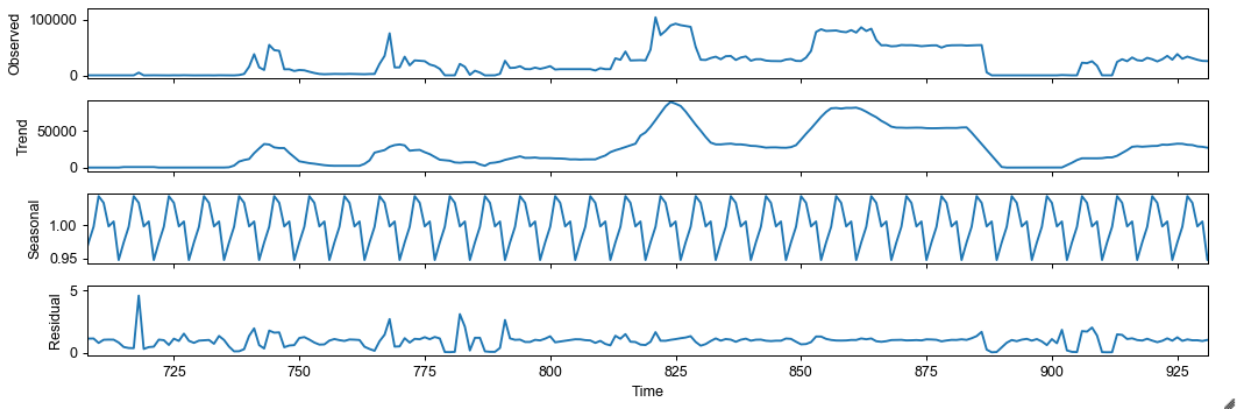


Рисунок 9 – Компоненты временного ряда

Работа моделей ARIMA и Хольта-Винтерса заключается в прогнозировании значения во временном ряде. ARIMA позволяет проводить вычисления на рядах с выраженным трендом, а модель Хольта-Винтерса может работать с трендом и сезонностью во временном ряде.

3.1 Модель Хольта-Винтерса

Данная модель предназначена для прогноза временных рядов, которые имеют сезонность (цикличность).

Модель использует значения за прошлые временные отрезки. Она включает в себя 3 компоненты: сезонность, тренд, очищенные значения.

Прогноз $\hat{Y}[t+h]$ на период номер h определяется по формуле (1).

$$\hat{Y}[t+h] = a[t] + h \cdot b[t] + s\left[t - p + 1 + (h-1) \bmod p\right], \quad (1)$$

где $a[t]$, $b[t]$ и $s[t]$ рассчитываются по формуле (2).

$$\begin{aligned} a[t] &= \alpha\left(Y[t] - s[t-p]\right) + (1-\alpha)\cdot\left(a[t-1] + b[t-1]\right), \\ b[t] &= \beta\left(a[t] - a[t-1]\right) + (1-\beta)\cdot b[t-1], \\ s[t] &= \gamma\left(Y[t] - a[t]\right) + (1-\gamma)\cdot s[t-p]. \end{aligned} \quad (2)$$

В данной формуле α, β, γ являются параметрами, которые необходимо находить перед прогнозированием. Все они изменяются от 0 до 1. Параметр p является периодом сезонности. Компонент $a[t]$ описывает ряд, очищенный от тренда и сезонности, $b[t]$ служит для оценки тренда, а $s[t]$ служит для оценки сезонности.

3.2 Модель ARIMA

Для обнаружения аномалий во временных рядах возможно использование модели ARIMA [4] – авторегрессионное интегрированное скользящее среднее, которое обобщает авторегрессионное скользящее среднее для случая нестационарных временных рядов.

В основе ARIMA лежат такие модели, как авторегрессионная модель и модель скользящего среднего.

В авторегрессионной модели значение временного ряда представляется линейной комбинацией значений ряда в предыдущие моменты времени по формуле (3):

$$AR(p): X_t = c + \sum_{i=1}^{p} a_i X_{t-i} + \varepsilon_t. \quad (3)$$

В данной формуле X_t, X_{t-i} – значения временного ряда в моменты времени $t,\ t-i$ соответственно, $a_1 ... a_p$ – параметры модели, c – постоянная, ε_t – белый шум, то есть последовательность независимых и одинаково распределённых случайных величин (как правило, нормальных), случайная ошибка измерения. Число p называется порядком модели.

Модель скользящего среднего показана в формуле (4):

$$MA(q): X_t = \mu + \sum_{i=1}^{q} b_i \varepsilon_{t-i} + \varepsilon_t, \quad (4)$$

где μ – среднее значение ряда,

$b_1 ... b_q$ – параметры модели,

ε_t – белый шум,

q – порядок модели.

В данной модели элемент временного ряда в текущий момент времени вычисляется как сумма среднего, белого шума в текущий момент времени и линейная комбинация белого шума в предыдущие моменты времени [11].

Комбинируя AR(p) и MA(q), мы можем получить авторегрессионное скользящее среднее ARMA(p, q), которое выражается формулой (5):

$$ARMA(p,q): X_t = c + \sum_{i=1}^{p} a_i X_{t-i} + \sum_{i=1}^{q} b_i \varepsilon_{t-i} + \varepsilon_t. \qquad (5)$$

Построение модели ARMA(p, q) предполагает стационарность временного ряда. Для решения практических задач обычно используется понятие слабо стационарного временного ряда. Слабо стационарным называется временной ряд в случае, если его теоретические математическое ожидание и дисперсия не зависят от времени, а ковариация между его значениями в моменты t и t + s зависит только от s, а не от времени [12] На практике показатели работы информационно-вычислительных систем не всегда являются стационарными.

Однако временной рад можно сделать стационарным, взяв разности некоторого порядка от исходного временного ряда. Именно дифференцирование временного ряда отличает модель ARMA(p, q) от ARIMA(p, d, q), которая представляется формулой (6):

$$ARIMA(p,d,q): \Delta^d X_t = c + \sum_{i=1}^{p} a_i (\Delta^d X_{t-i}) + \sum_{i=1}^{q} b_i \varepsilon_{t-i} + \varepsilon_t, \qquad (6)$$

где Δ^d – оператор разности временного ряда порядка d.

Оператор разности представляется в виде выражения (7):

$$\begin{aligned}
&\Delta^1 X_t = X_t - X_{t-1},\\
&\Delta^2 X_t = \Delta^1 X_t - \Delta^1 X_{t-1},\\
&\ldots\\
&\Delta^d X_t = \Delta^{d-1} X_t - \Delta^{d-1} X_{t-1}.
\end{aligned} \qquad (7)$$

Если разности $\Delta^d X_t$ временного ряда X_t стационарны, а разности меньшего порядка нестационарны, то X_t называется интегрированным рядом порядка *d*.

Для подбора параметров модели ARIMA *p*, *d* и *q* используется информационный критерий Акаике, представляемый формулой (8):

$$AIC = 2k - 2\ln\left(\hat{L}\right), \qquad (8)$$

где $\hat{L}$ – функция правдоподобия, k – общее параметров в модели.

Наиболее подходят те параметры *p*, *d*, *q*, при которых критерий Акаике достигает наименьшего значения, то есть небольшое число параметров *p*, *q* и большая функция правдоподобия.

4 Искусственные нейронные сети

Искусственные нейронные сети (ИНС) представляют собой математическую модель, в основе которой лежат принципы функционирования биологических нейронных сетей. Подобно им, ИНС состоят из взаимосвязанных групп нейронов, где каждый нейрон принимает входные сигналы с определенным весом. Для входов вычисляется взвешенная сумма со смещением, выполняется преобразование с помощью функции активации, и в результате получается выходной сигнал [13]

Нейронные сети состоят из отдельных слоев, из которых первый слой называется входным, средний слой – скрытым и последний называется выходным (рис. 10). На входном слое в нейроны подаются значения входных переменных, на выходном слое выходные сигналы нейрона присваиваются выходным переменным.

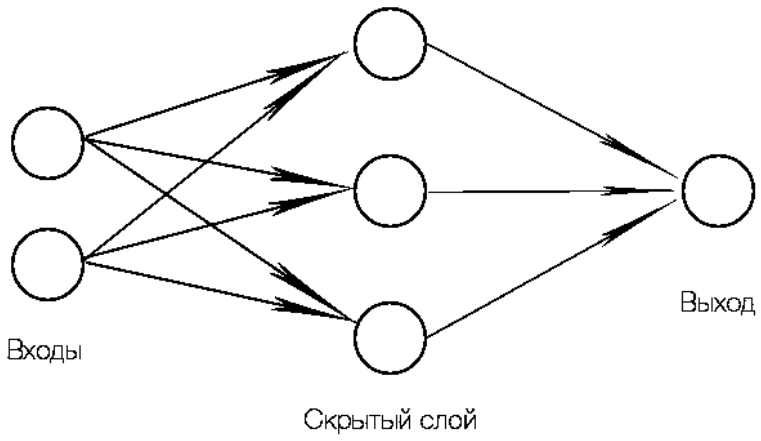


Рисунок 10 – Обобщенная структура ИНС

Каждый нейрон, по сути, выполняет нелинейное преобразование входного сигнала в выходной по формуле (9):

$$y_k = \varphi_k\left(\sum_i w_{ki}x_i + b_k\right). \qquad (9)$$

Схема нейрона, описываемого формулой 9, представлена на рисунке 11:

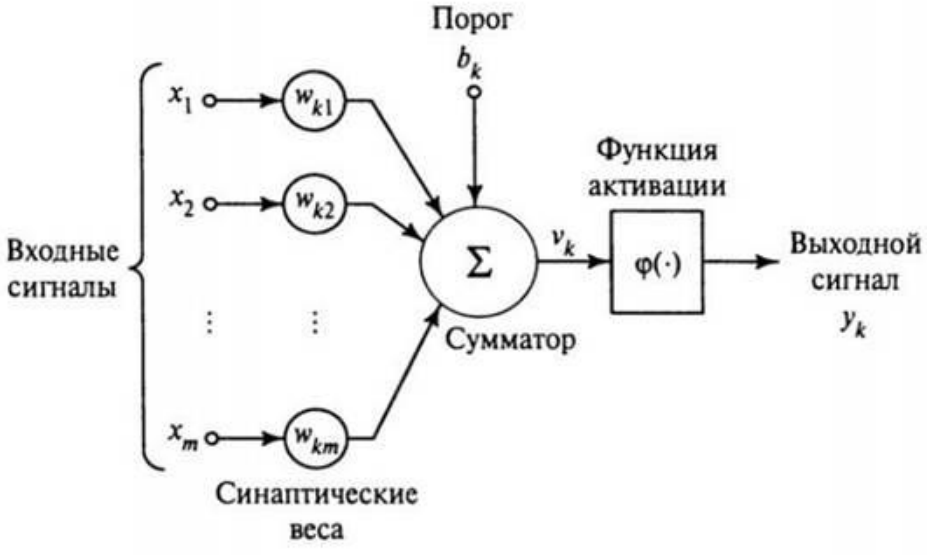


Рисунок 11 –Схема отдельного нейрона

Функция активации определяет силу выходного сигнала. Существуют различные нелинейные функции, которые можно использовать для активации. Наиболее распространенными являются сигмоида, гиперболический тангенс, линейный выпрямитель (Rectified linear unit, ReLU) (рис. 12) [14].

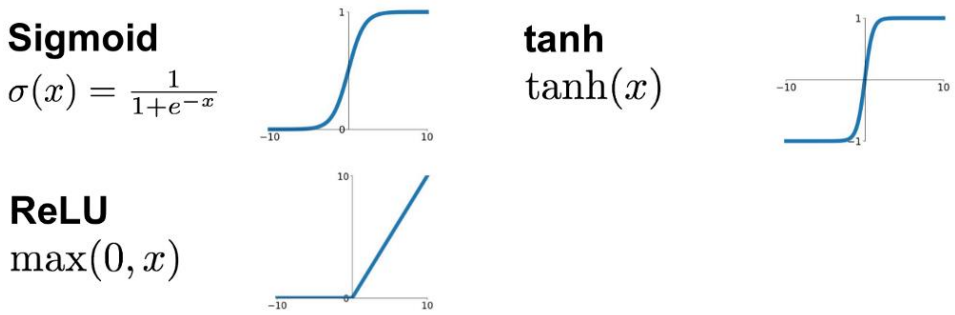


Рисунок 12 – Функции активации

Создание нейронной сети включает в себя обучение сети, главной целью которого является способность выдавать правильные результаты

классификации, как для обучающей выборки данных, так и для схожих данных, неидентичных обучающей выборке.

Обучение состоит из следующих этапов:

1) Предсказание для входа из обучающего набора данных выходного сигнала с учетом весов, проинициализированных некоторым начальным значением.

2) Вычисление ошибки предсказания с помощью функции потерь. Чаще всего в качестве нее используется среднеквадратическое отклонение (MSE) описываемого формулой (10):

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(Y_i - Y_i')^2, \qquad (10)$$

где n – размерность одного образца,

Y_i – истинное значение, которое должно быть на выходе сети,

Y_i' – предсказанное значение, которое получилось на выходе сети.

3) Уменьшение потерь выполняется оптимизатором, который реализует алгоритм обратного распространения – корректировку параметров для сокращения потерь.

4) Процесс предсказания повторяется для откорректированных весов.

Данные этапы составляют одну эпоху в обучении нейронной сети (рис. 13). Сеть считается обученной, если она на выходе дает предсказание с достаточно минимальными потерями [15].

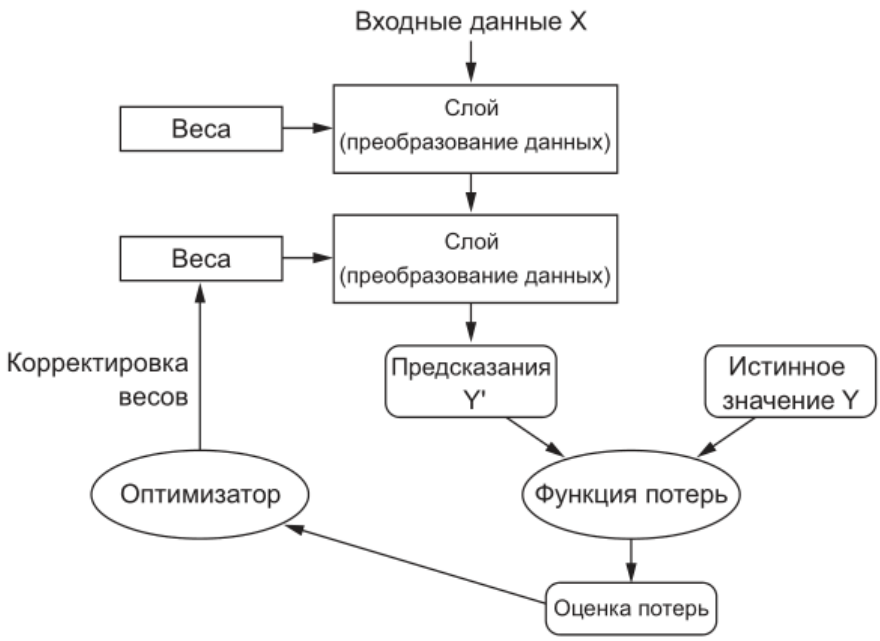


Рисунок 13 – Принцип действия обучения нейронной сети

4.1 Нейронные сети с долгой краткосрочной памятью (LSTM)

Для обнаружения аномалий во временных рядах возможно использование методов прогнозирования. Прогнозирование с помощью нейронных сетей предполагает работу с входными данными, представляющими последовательность, поэтому немаловажно чтобы нейронная сеть, обрабатывая очередной показатель в определенный момент времени, умела основываться на значениях в предыдущие моменты времени. Это позволяют сделать рекуррентные нейронные сети (Recurrent Neural Network, RNN) (рис. 14).

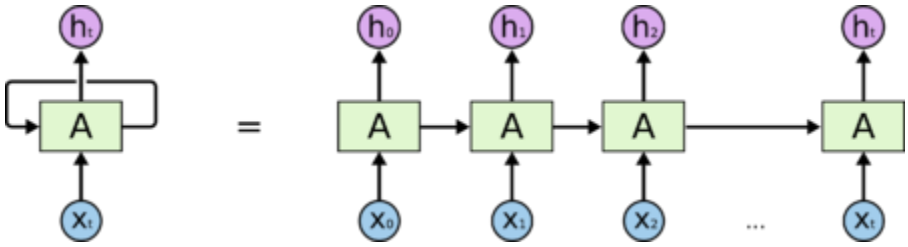


Рисунок 14 – Рекуррентная нейронная сеть и ее развернутое представление

В такой сети ячейка получает входную последовательность и обрабатывает ее в цикле (рис. 15): на вход подается элемент, и преобразованное значение отправляется как на выходной слой, так и на вход со следующим элементов последовательности. На этом начинается новая итерация цикла с учетом результата для предыдущего значения.

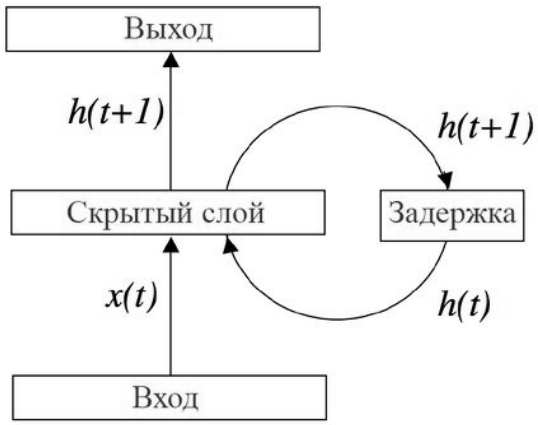


Рисунок 15 – Схема работы скрытого слоя

Однако у рекуррентных сетей проблема в том, что по мере получения новых значений влияние предыдущих значений ослабевает. Эту проблему призваны решить сети с долгой краткосрочной памятью (Long short-term memory; LSTM) [16].

Долгая краткосрочная память – разновидность рекуррентной нейронной сети, способная к обучению долговременным зависимостям. Ее особенностью является наличие у ячейки состояния, которое регулируется так называемыми фильтрами (англ. «gate») (рис. 16).

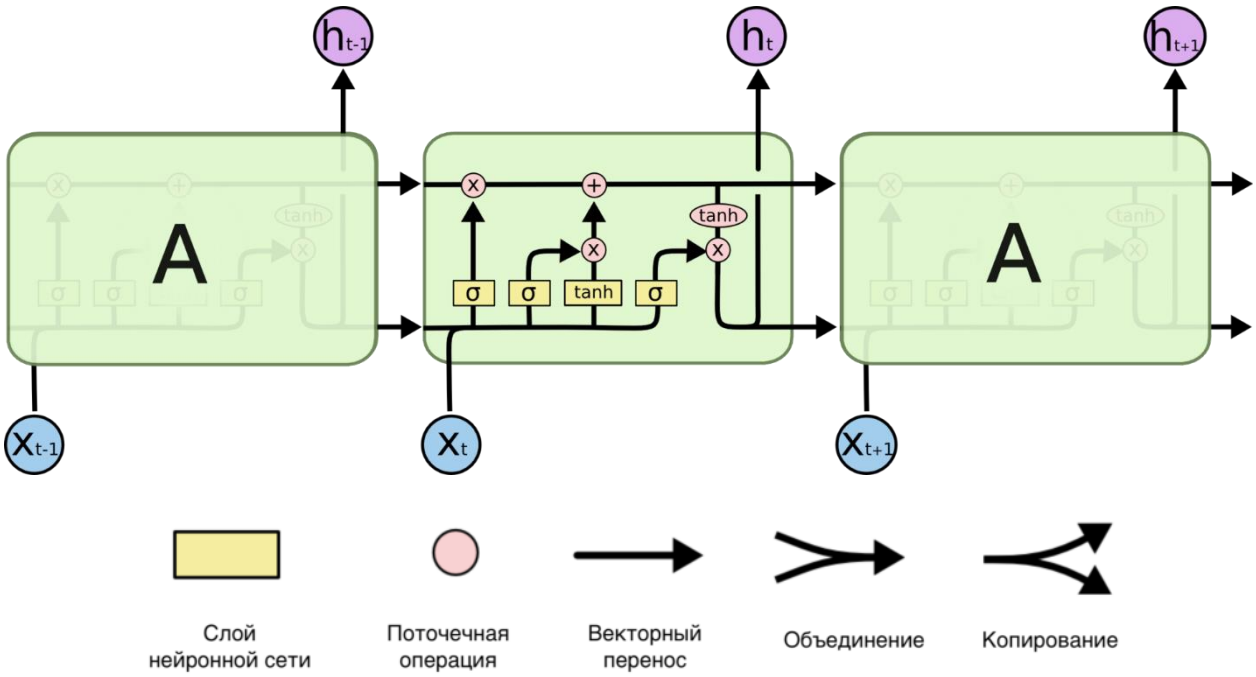


Рисунок 16 – Схема ячейки LSTM

Фильтры представляют собой сигмоидальные слои (формулы (11), (12) и (15)). Они вычисляют взвешенную сумму элементов конкатенации предыдущего выходного значения h_{t-1} и текущего входного значения x_t со смещением, после передавая сумму сигмоидной функции активации. Первый слой – слой фильтра забывания – определяет веса f_t для каждого числа из вектора состояния C_{t-1} и выражается формулой (11):

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f), \tag{11}$$

Учитывая предыдущее выходное значение h_{t-1} и текущее входное значение x_t, он возвращает вектор значений от 0 до 1 f_t, где 0 – «забыть» элемент состояния, 1 – «сохранить».

Затем определяется слой входного фильтра, определяющий веса i_t для каждого числа из $\tilde{C}_t$ – значения-кандидаты, которое можно добавить в состояние ячейки. Они вычисляются по формулам (12) и (13):

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i), \tag{12}$$

$$\tilde{C}_t = \sigma(W_C \cdot [h_{t-1}, x_t] + \tilde{b}_C). \tag{13}$$

Далее происходит обновление состояния C_{t-1} на C_t по формуле (14):

$$C_t = f_t \cdot C_{t-1} + i_t \cdot \tilde{C}_t. \tag{14}$$

В конце происходит генерация выходных данных h_t из текущего состояния C_t с применением фильтра по формуле (15):

$$\begin{aligned} o_t &= \sigma(W_o \cdot [h_{t-1}, x_t] + b_o), \\ h_t &= o_t \cdot \tanh(C_t). \end{aligned} \tag{15}$$

По сравнению с моделью ARIMA сеть LSTM позволяет предсказывать значения нелинейными преобразованиями имеющихся значений временного ряда. Причем за счет вычисления вектора состояния в преобразованиях могут участвовать не только ближайшие предыдущие значения, но и значения в более ранние моменты времени.

5 Программная реализация системы

Для обнаружения аномалий был применены алгоритмы прогнозирования на основе модели ARIMA и нейросетевой модели LSTM, которые на основе обучающей последовательности вычисляют элементы последовательности в последующих отрезках времени. Если значения, которые спрогнозировала программа, расходятся с поданными на вход значениями на заданное отклонение, то в этом месте отмечается аномалия типа выброса.

Для подготовки входных данных был настроен виртуальный сервер с системой мониторинга Zabbix, в которой собираются данные о работе сервера, среди которых загрузка процессора, объем оперативной памяти, скорость сетевых интерфейсов, нагрузка жестких дисков и др.

5.1 Используемые в работе средства

5.1.1 Язык Python

Python является интерпретируемым языком программирования с динамической типизацией, поддерживающий процедурное, объектно-ориентированное, функциональное и другие парадигмы программирования. Преимуществом Python является наличие интерпретаторов для многих операционных систем, а также богатая стандартная библиотека и обширный набор загружаемых библиотек. Благодаря библиотекам numpy, pandas, scipy Python получил распространение в области анализа данных.

5.1.2 Zabbix API

API, предоставляемый системой Zabbix, дает возможность отправлять запросы для настройки серверной части системы и для получения данных о

конфигурации Zabbix. Также API позволяет получать историю по тому или иному показателю работы системы. Отправка запросов реализуется с помощью вызова процедур в протоколе JSON-RPC. Запросы и ответы между клиентами и API закодированы с использованием формата JSON. Для использования данного API в Python существует библиотека pyzabbix [17]

5.2 Программная реализация модели ARIMA

С помощью API системы мониторинга Zabbix программа собирает показатели системы по заданному ключу элемента данных в заданные промежутки времени и сохраняет полученные значения в файл типа csv. Рассмотрим для примера показатель активного времени работы жесткого диска в процентах (рис. 17):

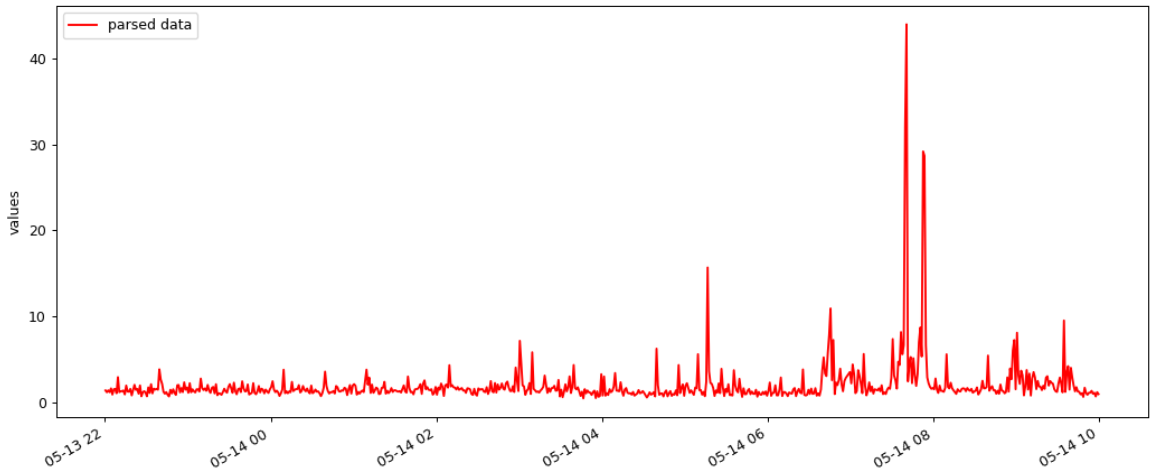


Рисунок 17 – Активное время работы жесткого диска в процентах

После этого из csv-файла создается таблица Pandas.Dataframe, в которой ряды разделяются на два отрезка времени: на обучающую и прогнозируемую часть. На обучающей части выполняется подбор коэффициентов для ARIMA, которые будут использоваться для прогнозирования.

Для подбора параметров p, d, q используется функция auto_arima. С ее помощью методом перебора ищутся такие параметры, которые дадут

наименьшее значение информационного критерия Акаике.

После обучения модели в цикле идет прогнозирование пакетами по 10 значений, что задано в переменной forecast_length. Затем предсказанные значения добавляются в модель, чтобы следующие предсказания вычислять на их основе, то есть осуществляется обновление модели.

Например, предсказание для модели ARIMA(4, 1, 4) показано на рисунке 18. На данном графике среднеквадратическая ошибка составила 3,084.

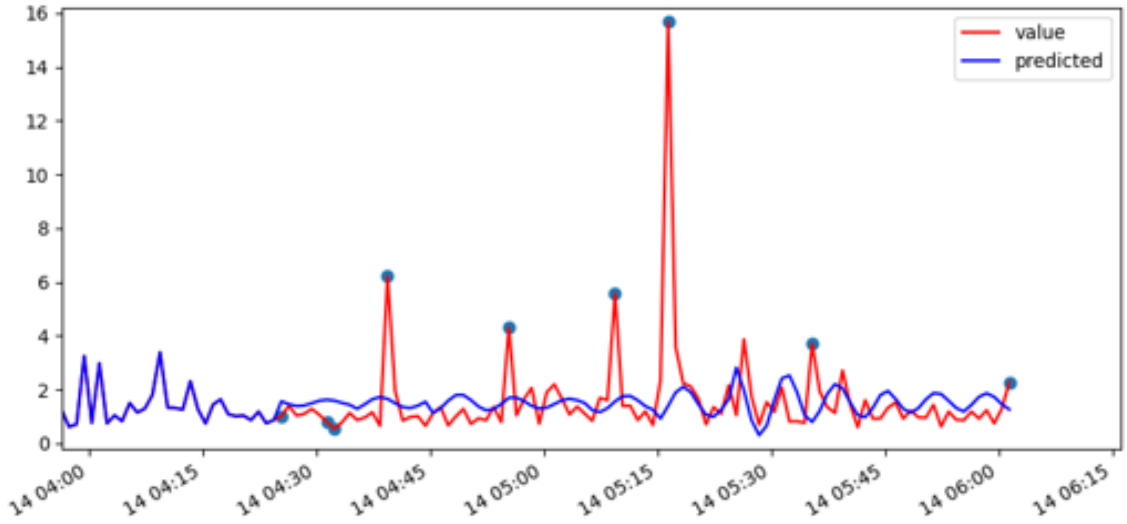


Рисунок 18 – Прогноз модели

Для определения моментов времени с аномалиями для каждого значения рассчитывается ошибка прогноза:

```
test_df['error'] = test_df['value'] - test_df['predicted']
```

Для ошибки вычисляется скользящее среднее и скользящее среднеквадратическое отклонение на 12 предыдущих значениях:

```
df['err_meanval'] = df['error'].rolling(window=12).mean()
df['err_deviation'] = df['error'].rolling(window=12).std()
```

На основе скользящего среднего и среднеквадратическое отклонения строится доверительный интервал в два среднеквадратических отклонения:

```
df['-lim'] = df['err_meanval'] - (2 * df['err_deviation'])
df['+lim'] = df['err_meanval'] + (2 * df['err_deviation'])
```

Если значение ошибки выходит за доверительный интервал, то в момент времени, в который это произошло, отмечается аномалия:

```
df['anomaly_points'] = np.where(((df['error'] > df['+lim']) | (df['error'] < df['-lim'])), df['value'], np.nan)
```

5.3 Программная реализация нейросетевой модели

Аналогично модели ARIMA, программная реализация сети LSTM работает с данными из файла csv, которые содержат собранные данные из системы Zabbix.

Перед запуском обучения нейронной сети выполняется обработка временного ряда.

На начальном этапе из значений создается массив Pandas.Series. Далее выполняется дифференцирование полученного ряда, что в большинстве случаев позволяет добиться стационарности (рис. 19).

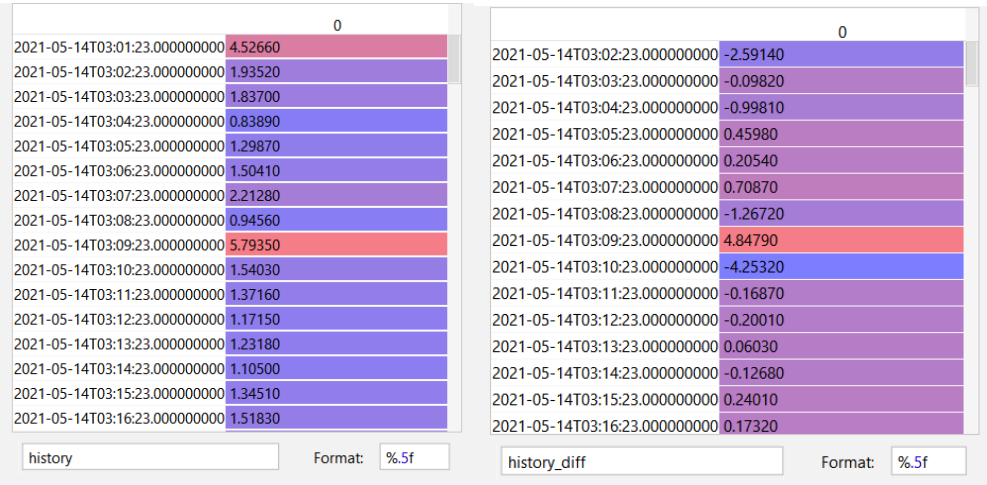
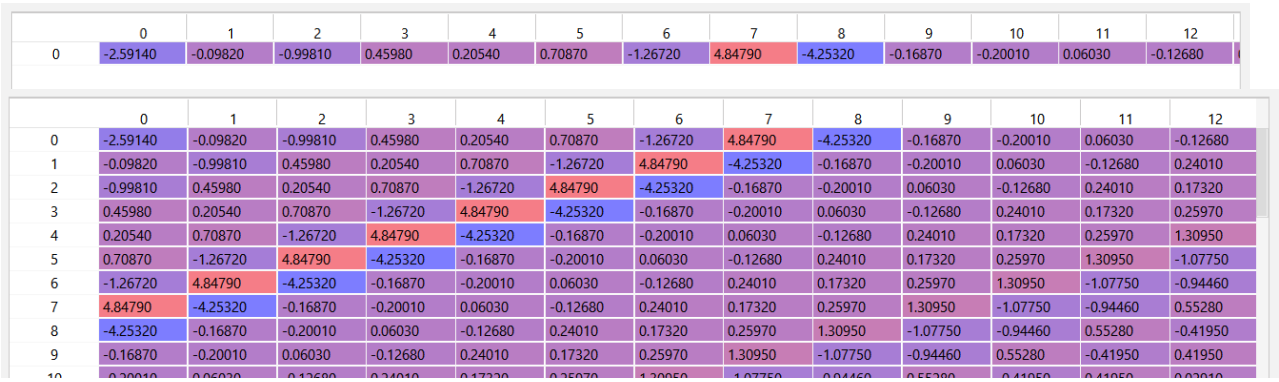


Рисунок 19 – Дифференцирование временного ряда

Чтобы данные были пригодны для обучения, на основе массива создается матрица, у которой каждая строка содержит фрагмент временного ряда длиной 20, определяемой переменной time_steps. Каждая следующая строка в этой матрице содержит сдвиг на 1 значение относительно предыдущей (рис. 20).



	0	1	2	3	4	5	6	7	8	9	10	11	12
0	-2.59140	-0.09820	-0.99810	0.45980	0.20540	0.70870	-1.26720	4.84790	-4.25320	-0.16870	-0.20010	0.06030	-0.12680

	0	1	2	3	4	5	6	7	8	9	10	11	12
0	-2.59140	-0.09820	-0.99810	0.45980	0.20540	0.70870	-1.26720	4.84790	-4.25320	-0.16870	-0.20010	0.06030	-0.12680
1	-0.09820	-0.99810	0.45980	0.20540	0.70870	-1.26720	4.84790	-4.25320	-0.16870	-0.20010	0.06030	-0.12680	0.24010
2	-0.99810	0.45980	0.20540	0.70870	-1.26720	4.84790	-4.25320	-0.16870	-0.20010	0.06030	-0.12680	0.24010	0.17320
3	0.45980	0.20540	0.70870	-1.26720	4.84790	-4.25320	-0.16870	-0.20010	0.06030	-0.12680	0.24010	0.17320	0.25970
4	0.20540	0.70870	-1.26720	4.84790	-4.25320	-0.16870	-0.20010	0.06030	-0.12680	0.24010	0.17320	0.25970	1.30950
5	0.70870	-1.26720	4.84790	-4.25320	-0.16870	-0.20010	0.06030	-0.12680	0.24010	0.17320	0.25970	1.30950	-1.07750
6	-1.26720	4.84790	-4.25320	-0.16870	-0.20010	0.06030	-0.12680	0.24010	0.17320	0.25970	1.30950	-1.07750	-0.94460
7	4.84790	-4.25320	-0.16870	-0.20010	0.06030	-0.12680	0.24010	0.17320	0.25970	1.30950	-1.07750	-0.94460	0.55280
8	-4.25320	-0.16870	-0.20010	0.06030	-0.12680	0.24010	0.17320	0.25970	1.30950	-1.07750	-0.94460	0.55280	-0.41950
9	-0.16870	-0.20010	0.06030	-0.12680	0.24010	0.17320	0.25970	1.30950	-1.07750	-0.94460	0.55280	-0.41950	0.41950

Рисунок 20 – Составление матрицы из временного ряда

Далее производится нормализация значений с помощью функции MinMaxScaler пакета scikit-learn, чтобы они находились в промежутке [-1, 1].

При наборе данных в виде матрицы сдвигов нейронная сеть обучается предсказывать последний столбец матрицы на основе предыдущих столбцов.

После осуществляется разделение данных на обучающую и тестовую выборки, где тестовая выборка состоит из значений, число которых задано в переменной forecast_length, например 40. Это последние строки матрицы, в которых нужно выполнить предсказание последнего столбца.

Для реализации нейросетевой модели LSTM была использована библиотека Keras, для которой определение сети в программе выглядит следующим образом:

```
def fit_lstm(train, batch_size, epochs, neurons):
   X, y = train[:, :-1], train[:, -1:]
   X = X.reshape(X.shape[0], X.shape[1], 1)
   y = y.reshape(y.shape[0], y.shape[1], 1)

   model = models.Sequential()
   model.add(layers.recurrent.LSTM(
      neurons,
      batch_input_shape=(batch_size, X.shape[1], X.shape[2]),
      stateful=True)
   )
   model.add(layers.Dense(1))

   model.compile(loss='mean_squared_error', optimizer='adam')

   for i in range(epochs):
      model.fit(
         X, y, epochs=1,
```

```
            batch_size=batch_size,
            verbose="auto", shuffle=False
        )

        model.reset_states()

    return model
```

В качестве параметров были использованы значения time_steps = 20, batch_size = 10, epochs = 10, neurons = 40. Параметры обозначают следующее:

– time_steps – длина временного отрезка, по которому предсказывается следующее значение;

– batch_size – размер пакета входной последовательности, после обработки которого происходит обновление весов;

– epochs – количество эпох нейронной сети;

– neurons – число элементов в векторе состояния, что в развернутом виде LSTM означало бы число нейронов.

Сеть состоит из слоя LSTM, который сохраняет состояние для каждой следующей строки из матрицы. Следующий слой Dense – нейрон, который объединяет 40 значений (выход слоя LSTM) в одно, считающееся прогнозом.

В качестве функции потерь используется среднеквадратическая ошибка, и для ее минимизации выбран оптимизатор Adam.

Так как модель сохраняет состояние от каждой строки, то после каждого прохода по всей матрице (окончания эпохи) нужно явно сбрасывать состояние. Поэтому обучение модели выполняется в цикле по одной эпохе.

Поскольку входные данные образуют последовательность, при обучении мы запрещаем перемешивание образцов – строк матрицы, поставив параметр shuffle=False.

После обучения модели она используется для получения последних столбцов матрицы на тестовой выборке. Предсказание выполняется по частям, имеющим число строк, равное batch_size; далее сеть обучается с новыми значениями. После получения всех значений из тестовой выборки выполняется обратное преобразование из нормализованных значений к

дифференцированному виду, и от дифференцированного к недифференцированному.

Далее начинается поиск аномалий – тех значений, которые наиболее отклоняются от предсказанных. Их определение выполняется аналогично модели ARIMA с помощью скользящего среднего ошибки прогнозирования.

Результат работы модели представлен на рисунке 21. Среднеквадратическая ошибка на прогнозируемом промежутке равна 5,310.

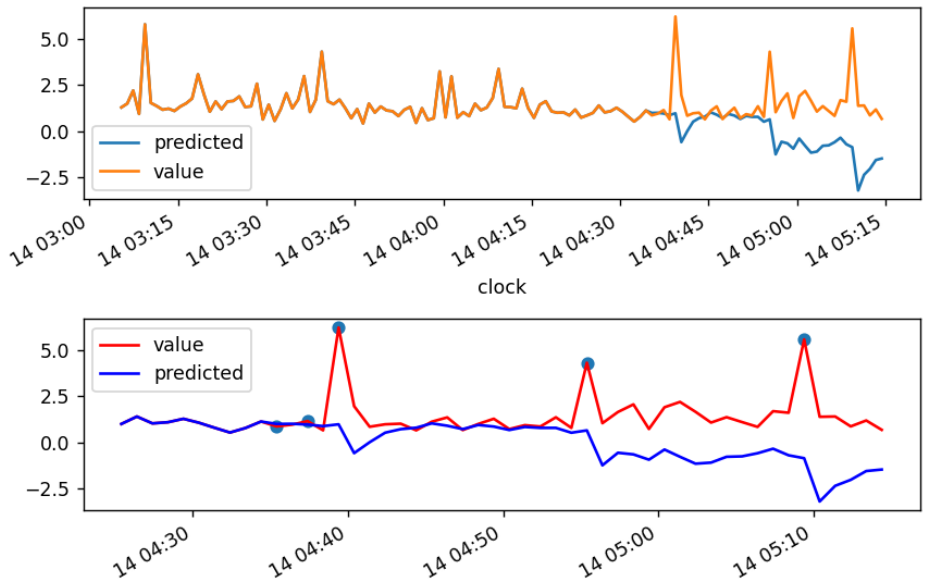


Рисунок 21 – Обнаружение аномалий помощью LSTM

Модель LSTM способна давать прогнозы с увеличивающеимся отклонением при большом объеме входных данных, хотя при их малом объеме (90 минут истории на 40 прогноза) обе модели правильно фиксируют аномальные значения в одинаковых участках. Исходя из этого модель ARIMA более предпочтительна благодаря стабильности работы и быстродействию (10 секунд обучения против более 60 у LSTM).

5.4 Разработка программы с графическим интерфейсом

Для разработки программы была использована библиотека Tkinter для языка Python. Начальный экран программы показан на рисунке 22:

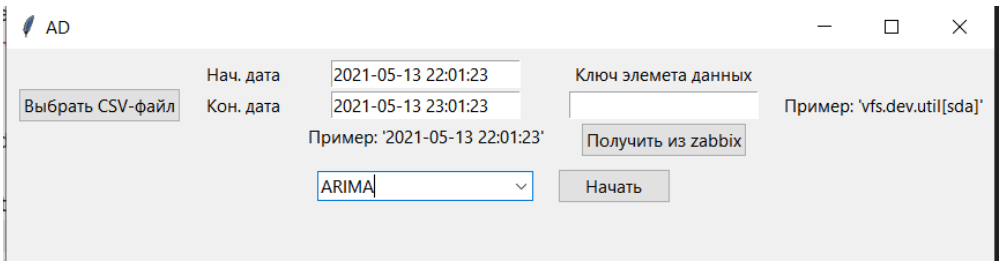


Рисунок 22 – Вид программы после запуска

Пользователь вводит интервал дат, с какого по какой промежуток времени загрузить данные в модель, и далее ему предлагается 2 способа выбора данных:

1) Выбрать из уже имеющихся csv-файлов. Для этого по кнопке «Выбрать CSV» открывается диалоговое окно выбора файлов.

2) Получить данные по ключу из Zabbix посредством ввода ключа элемента данных и нажатия кнопки «Получить из Zabbix». Эти данные также сохраняются локально в виде csv-файла.

В системе Zabbix каждый показатель работы сервера, так называемый «Элемент данных», имеет свое уникальное имя – ключ (например, system.cpu.util[,idle] для времени простоя процессора). Перечень ключей доступен в разделе «Последние данные» веб-интерфейса Zabbix (рис. 23).

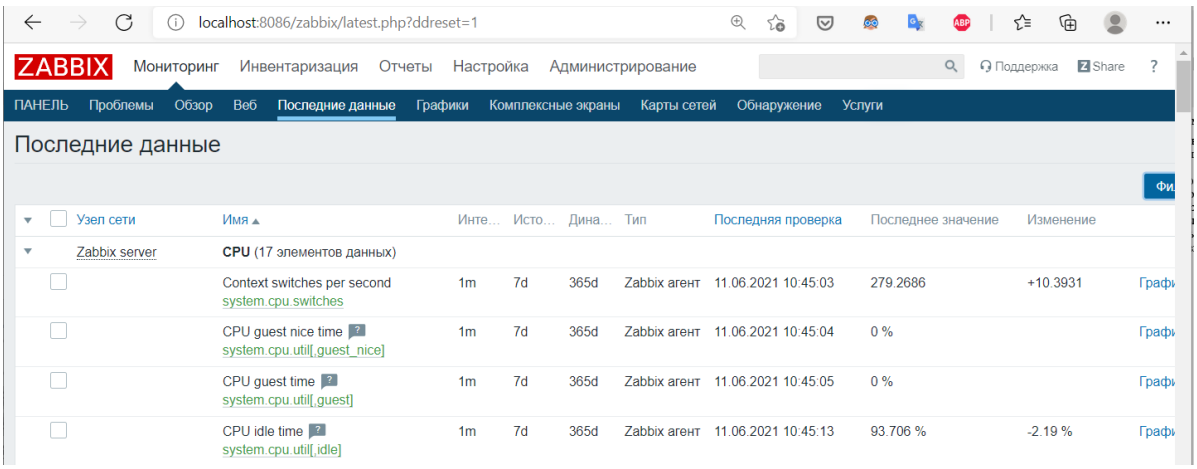


Рисунок 23 – Ключи метрик работы сервера

После того, как был определен временной ряд, пользователь во всплывающем списке выбирает модель прогнозирования для обнаружения аномалий: ARIMA или LSTM (рис. 24).

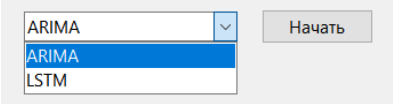


Рисунок 24 – Доступные модели прогнозирования

По нажатию кнопки «Начать» начинается работа модели на полученных данных, и в программе появляется график с результатом, где отмечены аномальные промежутки времени (рис. 25).

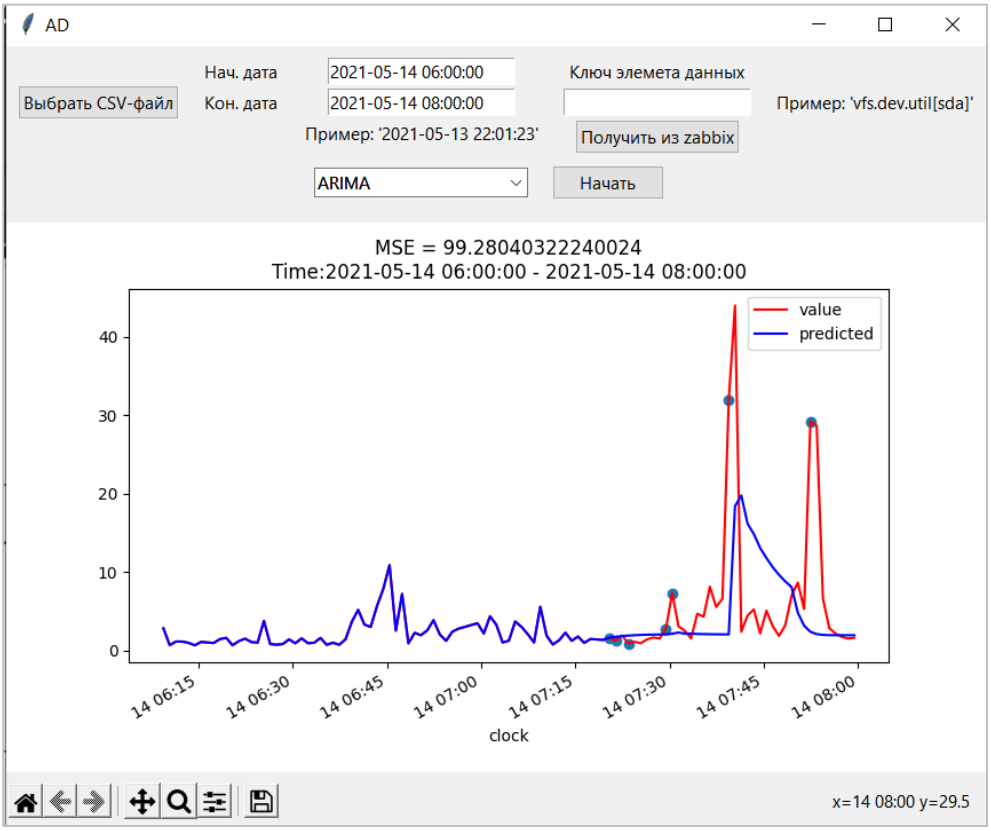


Рисунок 25 – Результат работы программы

ЗАКЛЮЧЕНИЕ

Разработка системы обнаружения аномалий направлена на оказание помощи при администрировании вычислительных систем и сетей. Главным образом, выявление аномального поведения позволяет упростить анализ нагрузки на сервер, уменьшить сроки аудита работоспособности серверов и исследования инцидентов, а также ускорить принятие решений по оптимизации их конфигурации.

В ходе выполнения выпускной квалификационной работы были выполнены следующие задачи:

– рассмотрены основные показатели работы серверов;

– выполнен анализ систем мониторинга серверов;

– рассмотрены основные виды аномалий и методы их обнаружения;

– настроен мониторинг сервера с помощью системы Zabbix для подготовки исходных данных;

– для обнаружения аномалий построены модели прогнозирования временных рядов авторегрессионной интегрированной скользящей средней (ARIMA) и нейронной сети с долгой краткосрочной памятью (LSTM);

– разработана программа, реализующая обнаружение аномалий в данных настроенной системы мониторинга.

Таким образом, в данной работе были выполнены поставленные задачи и достигнута цель: разработана система обнаружения аномалий в данных мониторинга с использованием методов машинного обучения и прогнозирования.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1 Система мониторинга сети [Электронный ресурс] // ИТ база знаний. – URL: https://wiki.merionet.ru/servernye-resheniya/sistema-monitoringa-seti/ (дата обращения 23.05.2021).

2 Успенский М. Б. Обзор подходов к обнаружению сбоев в системах хранения данных / М. Б. Успенский // Научно-технические ведомости СПбГПУ. Информатика. Телекоммуникации. Управление. 2019. Т. 12. № 4. с. 145—158.

3 Развёртывание и настройка Icinga 2 на Debian 8.6 [Электронный ресурс] // Блог IT-KB. – URL: https://blog.it-kb.ru/2016/11/21/deploy-and-configure-icinga-2-on-debian-8-6-part-1-installation-of-icinga-2-core-plugins-and-db-ido/ (дата обращения 23.05.2021).

4 Preetam J. Anomaly detection for monitoring: A statistical approach to time series anomaly detection / J. Preetam, B. Schwartz. – O'Reilly, 2015 – 75 с.

5 Кибер-оракул: поиск аномалий в данных мониторинга с помощью нейросети [Электронный ресурс] // Блог компании ITSumma / Хабр. – URL: https://habr.com/ru/company/itsumma/blog/341598/ (дата обращения 13.05.2021).

6 Обнаружение аномалий в данных сетевого мониторинга методами статистики [Электронный ресурс] // Хабр – URL: https://habr.com/ru/post/344762/ (дата обращения 13.05.2021).

7 Гайфулина Д. А., Котенко И. В. Анализ моделей глубокого обучения для задач обнаружения сетевых аномалий интернета вещей // Информационно-управляющие системы, 2021, № 1, с. 28–37.

8 Szmit M., Szmit A. Use of Holt-Winters method in the analysis of network traffic: Case study / M. Szmit, A. Szmit // Communications in Computer and Information Science. – 2011. Vol. 160. – с. 224–231.

9 Соболев К. В. Автоматический поиск аномалий во временных рядах [Электронный ресурс]. – URL: http://cs.mipt.ru/wp/wp-content/uploads/2018/06/Соболев-К.В..pdf (дата обращения 10.05.2021).

10 Яковлева А. В. Компоненты временного ряда - Эконометрика [Электронный ресурс]. – URL: https://be5.biz/ekonomika/e008/70.html (дата обращения 10.05.2021).

11 Common Approaches to Univariate Time Series [Электронный ресурс]. – URL: https://www.itl.nist.gov/div898/handbook/pmc/section4/pmc444.htm (дата обращения 14.05.2021).

12 Доугерти К. Введение в эконометрику / К. Доугерти. – М: ИНФРА-М, 2009. – 465 с.

13 Нейронные сети (statsoft.ru) [Электронный ресурс]. – URL: http://statsoft.ru/home/textbook/modules/stneunet.html (дата обращения 04.05.2021)

14 Nwankpa C., Ijomah W., Gachagan A., Marshall S. Activation functions: Comparison of trends in practice and research for deep learning // arXiv preprint arXiv:1811.03378. – 2018. – URL: https://arxiv.org/pdf/1811.03378v1.pdf

15 Шолле Ф. Глубокое обучение на Python / Ф. Шолле. – СПб.: Питер, 2018. — 400 с.

16 Christopher Olah. Understanding LSTM Networks [Электронный ресурс]. – URL: http://colah.github.io/posts/2015-08-Understanding-LSTMs/ (дата обращения 01.05.2021).

17 API [Zabbix Documentation 4.4] [Электронный ресурс]. – URL: https://www.zabbix.com/documentation/4.4/ru/manual/api/ (дата обращения 14.05.2021).

ПРИЛОЖЕНИЕ А

Реализация чтения данных из Zabbix

```
from pyzabbix import ZabbixAPI
import pandas as pd
import json
import datetime
import time


def get_zabbix_csv(key_,
               datetime_from=(datetime.datetime.now() -
datetime.timedelta(hours=6)).strftime('%Y-%m-%d %H:%M:%S'),
               datetime_till=datetime.datetime.now().strftime('%Y-%m-%d
%H:%M:%S')):
   ZBX_URL = 'http://127.0.0.1:8086/zabbix'
   ZBX_USER = 'a.chenib'
   ZBX_PASSWORD = 'zabbix'

   zapi = ZabbixAPI(url=ZBX_URL, user=ZBX_USER, password=ZBX_PASSWORD)
   print("Connected to Zabbix API Version %s" % zapi.api_version())

   # получить ид элемента данных
   item = zapi.item.get(filter={'host': 'Zabbix server', 'key_': key_})
   itemid = item[0]['itemid']
   print(itemid)

   # Создать временной отрезок
   time_from_dt = datetime.datetime.strptime(datetime_from, '%Y-%m-%d %H:%M:%S')
   time_till_dt = datetime.datetime.strptime(datetime_till, '%Y-%m-%d %H:%M:%S')
   time_from = int(time.mktime(time_from_dt.timetuple()))
   time_till = int(time.mktime(time_till_dt.timetuple()))

   hist_req_params = {"history": 0, "output": "extend",
                  "itemids": str(itemid),
                  "time_from": str(time_from), "time_till": str(time_till)}
   hist = zapi.do_request('history.get', hist_req_params)['result']

   time_now = datetime.datetime.now().strftime('%Y-%m-%d_%H-%M-%S')
   filename = r'pyzab\zabbixdata_%s_.csv' % time_now

   df = pd.read_json(json.dumps(hist))
   df.to_csv(filename, encoding='utf-8', index=False)
   return filename


def read_zabbix_csv(csv_path, time_from, time_till):
   # Читать из csv и привести к подходящему виду
   df = pd.read_csv(csv_path)

   if df.columns[0] == "timestamp":
      df = df.rename(columns={df.columns[0]: "clock"})
      df["clock"] = pd.to_datetime(df["clock"], format='%Y-%m-%d %H:%M:%S')
   elif df.columns[1] == "clock":
```

```
      df["clock"] = pd.to_datetime(df["clock"], unit='s')

   df.index = df["clock"]
   tseries_parsed = df["value"]

   time_mask = (tseries_parsed.index >= time_from) & (tseries_parsed.index <=
time_till)

   return tseries_parsed.loc[time_mask]
```

Реализация разметки аномальных значений

```
import numpy as np
import pandas as pd


def detect_anomalies(df, window, confid_interval=2):
   df.fillna(0, inplace=True)
   df['error'] = df['value'] - df['predicted']
   df['error, %'] = (df['value'] - df['predicted']) / df['predicted'] * 100
   df['err_meanval'] = df['error'].rolling(window=window).mean()
   df['err_deviation'] = df['error'].rolling(window=window).std()

   df['-lim'] = df['err_meanval'] - (confid_interval * df['err_deviation'])
   df['+lim'] = df['err_meanval'] + (confid_interval * df['err_deviation'])
   df['anomaly_points'] = np.where(((df['error'] > df['+lim']) | (df['error'] <
df['-lim'])), df['value'], np.nan)

   return df


def anom_plot(df, window, ax):
   df.plot(ax=ax, kind='line', use_index=True, y=['value', 'predicted'],
color=['red', 'blue'])
   ax.scatter(df['anomaly_points'].index, df['anomaly_points'])
```

Реализация модели ARIMA

```
from pmdarima import auto_arima
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

from sklearn.metrics import mean_squared_error
from math import sqrt
from warnings import filterwarnings
import time

from zabbix_load import get_zabbix_csv, read_zabbix_csv
from mark_anomalies import detect_anomalies, anom_plot


FORECAST_LENGTH = 10
```

```
def main_arima(csv_path, time_from='2021-05-13 22:01:23', time_till='2021-05-13
23:10:23'):

   tseries_parsed = read_zabbix_csv(csv_path, time_from, time_till)

   # train and prediction data variables
   train = tseries_parsed[0:-40]
   test = tseries_parsed[-40:]

   print("loaded data count: %d \n" % (tseries_parsed.size))
   print("train data count : %d \n" % (train.size))
   print("test data count : %d \n" % (test.size))

   filterwarnings('ignore')

   X = train.tolist()
   model_arima = auto_arima(train, start_p=4, max_p=7, start_q=4, max_q=7,
trace=True)

   predictions = []
   for i in range(0, test.size, FORECAST_LENGTH):\

      predicted_values = model_arima.predict(n_periods=FORECAST_LENGTH)
      for val in predicted_values:
         predictions.append(val)
         X.append(val)
      model_arima.update(test[i:i+FORECAST_LENGTH])

   predictions = np.array(predictions)[:test.size]

   test_df = test.to_frame()
   test_df.insert(test_df.shape[1], 'predicted', predictions)

   predictions_series = pd.Series(predictions, index=test.index)
   history_with_predicted = pd.concat([train, predictions_series])
   history_df = tseries_parsed.to_frame()
   history_df.insert(history_df.shape[1], 'predicted', history_with_predicted)

   mse = mean_squared_error(test, predictions)
   rmse = sqrt(mse)

   print("Test result: {} ".format(mse))
   print("RMSE : %3.f \n" % rmse)

   fig = plt.figure()
   ax = fig.add_subplot()

   anom_df = detect_anomalies(history_df, 10, 2)
   anom_plot(anom_df, 10, ax)

   ax.set_title("MSE = {}\nTime:{} - {}".format(mse, time_from, time_till))
   return fig, ax


if __name__ == "__main__":
   start_time = time.time()

   fig, ax = main_arima('D:/pyzab/zabbixdata_2021-05-15_23-58-30_.json.csv',
      '2021-05-14 02:00:00', '2021-05-14 08:00:00')
```

```
    stop_time = time.time()
    print("Time duration: {} sec.".format(stop_time - start_time))
    plt.show()
```

Реализация модели LSTM

```
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
import time
from math import sqrt

from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import MinMaxScaler
from keras import models, layers

from zabbix_load import get_zabbix_csv, read_zabbix_csv
from mark_anomalies import detect_anomalies, anom_plot


FORECAST_LENGTH = 40
TIME_STEPS = 20
BATCH_SIZE = 10
EPOCHS = 10
NEURONS = 40


def timeseries_to_stacked(series, time_steps):
    values = series.values
    sequences = []
    for i in range(0, len(values) + 1 - time_steps):
        sequences.append(values[i: (i + time_steps)])
    return np.stack(sequences)


def difference(tseries):
    tseries_diff = tseries.diff(periods=1)
    return tseries_diff[1:]


def scale(train, test=None):
    # преобразовать train (требуется матрица)
    train = train.reshape(train.shape[0], train.shape[1])
    # обучить нормализатор
    scaler = MinMaxScaler(feature_range=(-1, 1))
    scaler = scaler.fit(train)
    train_scaled = scaler.transform(train)

    if not (test is None):
        # transform test
        test = test.reshape(test.shape[0], test.shape[1])
        test_scaled = scaler.transform(test)

    return scaler, train_scaled, test_scaled
```

```
def invert_difference(history, predicted_diff):
   predicted_undiff = pd.Series(predicted_diff)
   predicted_undiff[0] = history[-len(predicted_diff) - 1] + predicted_diff[0]
   for i in range(1, len(predicted_undiff)):
      predicted_undiff[i] = predicted_undiff[i-1] + predicted_diff[i]
   predicted_undiff.index = history.index[-len(predicted_undiff):]
   return predicted_undiff


def invert_scale(scaler, X):
   X = scaler.inverse_transform(X)
   return X


def fit_lstm(train, batch_size, epochs, neurons):
   X, y = train[:, :-1], train[:, -1:]
   X = X.reshape(X.shape[0], X.shape[1], 1)
   y = y.reshape(y.shape[0], y.shape[1], 1)

   model = models.Sequential()
   model.add(layers.recurrent.LSTM(
      neurons,
      batch_input_shape=(batch_size, X.shape[1], X.shape[2]),
      stateful=True)
   )
   model.add(layers.Dense(1))

   model.compile(loss='mean_squared_error', optimizer='adam')
   # plot_model(model, to_file='model_plot.png', show_shapes=True,
show_layer_names=True)
   print(model.summary())
   for i in range(epochs):
      model.fit(
         X, y, epochs=1,
         batch_size=batch_size,
         verbose="auto", shuffle=False
      )

      model.reset_states()

   return model


def main_lstm(csv_path, time_from='2021-05-14 00:01:23', time_till='2021-05-14
05:10:23'):
   global_start_time = time.time()

   history = read_zabbix_csv(csv_path, time_from, time_till)
   history = history[len(history) % BATCH_SIZE:]

   history_diff = difference(history)
   stacked_history = timeseries_to_stacked(history_diff, time_steps=TIME_STEPS)
   train, test = stacked_history[:-FORECAST_LENGTH, :], stacked_history[-
FORECAST_LENGTH:, :]
   scaler, train_scaled, test_scaled = scale(train, test)

   print("Model fit started")
   fit_start_time = time.time()
```

```
   model = fit_lstm(train_scaled,
                batch_size=BATCH_SIZE, epochs=EPOCHS, neurons=NEURONS)
   print("Model fit complete. Duration: {} sec.".format(time.time() -
fit_start_time))

   X_test = test_scaled[:BATCH_SIZE, :-1]
   X_test = X_test.reshape(X_test.shape[0], X_test.shape[1], 1)

   X_train = train_scaled[:, :-1]
   X_train = X_train.reshape(X_train.shape[0], X_train.shape[1], 1)
   X_train = np.concatenate((X_train, X_test), axis=0)

   print("Prediction started")
   pred_start_time = time.time()
   y = model.predict(X_test, batch_size=BATCH_SIZE)
   print("Prediction complete. Duration: {} sec.".format(time.time() -
pred_start_time))
   predicted = y

   y_train = train_scaled[:, -1:]
   y_train = y_train.reshape(y_train.shape[0], 1)
   y_train = np.concatenate((y_train, y), axis=0)

   # выполнить прогноз и обновить модель
   for i in range(1, int(FORECAST_LENGTH / BATCH_SIZE)):

      for _ in range(EPOCHS):
         model.fit(
            X_train, y_train, epochs=1,
            batch_size=BATCH_SIZE,
            verbose="auto", shuffle=False
         )
         model.reset_states()

      start = i * BATCH_SIZE

      X_test = test_scaled[start: start + BATCH_SIZE, :-1]
      X_test = X_test.reshape(X_test.shape[0], X_test.shape[1], 1)

      y = model.predict(X_test, batch_size=BATCH_SIZE)
      predicted = np.concatenate((predicted, y), axis=0)

      X_train = np.concatenate((X_train, X_test), axis=0)
      y_train = np.concatenate((y_train, y), axis=0)

   test_scaled_with_predicted = np.concatenate((test_scaled[:, :-1], predicted),
axis=1)

   # invert_scale нужно подавать матрицу
   test_with_predicted = invert_scale(scaler, test_scaled_with_predicted)

   predicted_unscaled = test_with_predicted[:, -1]
   predicted_undiff = invert_difference(history, predicted_unscaled)

   print("Global time duration: {} sec.".format(time.time() -
global_start_time))

   def anom_detect(history_series, predicted_series):
```

```
      history_with_predicted = pd.concat([history_series[:-
len(predicted_series)], predicted_series])
      history_df = history_series.to_frame()
      history_df.insert(history_df.shape[1], 'predicted',
history_with_predicted)
      fig, axes = plt.subplots(nrows=2, ncols=1)
      fig.tight_layout()
      history_df.plot(ax=axes[0], kind='line', y=['predicted', 'value'])

      ROLL_WINDOW = 10
      anom_df = history_df[-FORECAST_LENGTH - ROLL_WINDOW:].copy()
      anom_df = detect_anomalies(anom_df, ROLL_WINDOW, 2)
      anom_plot(anom_df, ROLL_WINDOW, axes[1])
      # plt.show()
      return fig, axes


   fig, axes = anom_detect(history, predicted_undiff)

   mse = mean_squared_error(
      history[-len(predicted_undiff):].values,
      predicted_undiff.values)
   rmse = sqrt(mse)

   print("Test result: {} ".format(mse))
   print("RMSE : %3.f \n" % rmse)

   axes[0].set_title("MSE = {}\nTime:{} - {}".format(mse, time_from, time_till))
   return fig, axes


if __name__ == "__main__":
   fig, axes = main_lstm('D:/pyzab/zabbixdata_2021-05-15_23-58-30_.json.csv',
                  '2021-05-14 03:40:00', '2021-05-14 05:40:00')
   plt.show()
```

Реализация графического интерфейса программы

```
import tkinter as tk
from tkinter import filedialog
from tkinter import ttk

import matplotlib

matplotlib.use("TkAgg")
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg,
NavigationToolbar2Tk

from lstm import main_lstm
from arima import main_arima, get_zabbix_csv


def btn_getzabbix_click():
   filename.set(get_zabbix_csv(zbx_key.get(), time_from.get(), time_till.get()))
   state_message.set(filename.get())
```

```
def btn_openfile_click():
   filename.set(tk.filedialog.askopenfilename(
      initialdir="pyzab", filetypes=[("CSV file", "*.csv"), ("All files",
"*.*")]
   ))
   state_message.set(filename.get())


def btn_startdetect_click():
   state_message.set(cbx_methods.current())
   fig = matplotlib.figure.Figure()

   if cbx_methods.current() == 0:
      fig, ax = main_arima(filename.get(), time_from.get(), time_till.get())
      ax.plot()

   if cbx_methods.current() == 1:
      fig, axes = main_lstm(filename.get(), time_from.get(), time_till.get())
      for ax in axes:
         ax.plot()

   for widget in frame_plot.winfo_children():
      widget.destroy()

   canvas = FigureCanvasTkAgg(fig, frame_plot)
   canvas.get_tk_widget().pack(side=tk.BOTTOM, fill=tk.BOTH, expand=True)

   toolbar = NavigationToolbar2Tk(canvas, frame_plot)
   toolbar.update()
   canvas._tkcanvas.pack(side=tk.TOP, fill=tk.BOTH, expand=True)




window = tk.Tk()
window.title('AnomDetect')

# ------ Настройка -------

frame_setting = tk.Frame(window)
frame_setting.pack(pady=10)

filename = tk.StringVar()

btn_openfile = ttk.Button(frame_setting, text='Выбрать CSV-файл',
command=btn_openfile_click)
btn_openfile.grid(row=1, column=0, padx=10)

time_from = tk.StringVar(frame_setting, value="2021-05-13 22:01:23")
ent_time_from = tk.Entry(frame_setting, textvariable=time_from)
ent_time_from.grid(row=0, column=2, padx=10)

time_till = tk.StringVar(frame_setting, value="2021-05-13 23:01:23")
ent_time_till = tk.Entry(frame_setting, textvariable=time_till)
ent_time_till.grid(row=1, column=2, padx=10)

zbx_key = tk.StringVar()
ent_zbx_key = tk.Entry(frame_setting, textvariable=zbx_key)
```

```
ent_zbx_key.grid(row=1, column=4, padx=10)

btn_getzabbix = ttk.Button(frame_setting, text='Получить из zabbix',
command=btn_getzabbix_click)
btn_getzabbix.grid(row=2, column=4, padx=10)

cbx_methods = ttk.Combobox(frame_setting, values=["ARIMA", "LSTM"])
cbx_methods.grid(row=3, column=2, columnspan=2)

btn_startdetect = ttk.Button(frame_setting, text='Начать',
command=btn_startdetect_click)
btn_startdetect.grid(row=3, column=4, pady=10, sticky=tk.W)

ttk.Label(frame_setting, text='Нач. дата').grid(row=0, column=1, padx=10,
sticky=tk.E)
ttk.Label(frame_setting, text='Кон. дата').grid(row=1, column=1, padx=10,
sticky=tk.E)
ttk.Label(frame_setting, text="Пример: '2021-05-13 22:01:23'").grid(row=2,
column=2, padx=10, sticky=tk.N)
ttk.Label(frame_setting, text="Ключ элемента данных").grid(row=0, column=4,
padx=10)
ttk.Label(frame_setting, text="Пример: 'vfs.dev.util[sda]'").grid(row=1,
column=5, padx=10, sticky=tk.W)

# ------ График -------

frame_plot = tk.Frame(window, width=600, height=400)
frame_plot.pack(fill=tk.BOTH, expand=True)

# ------ Строка состояния -------

frame_state = tk.Frame(window)
frame_state.pack(side=tk.BOTTOM, fill=tk.X)

state_message = tk.StringVar()
lbl_messages = tk.Label(frame_state, textvariable=state_message)
lbl_messages.pack(side=tk.LEFT)

window.mainloop()
```