ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Актуальность темы исследования. В последнее время большое распространение получили различные сервисы-агрегаторы, объединяющие данные из нескольких источников определённой тематики в один, что уменьшает затрачиваемое пользователем время на поиск необходимой информации.

Для обработки запроса от одного пользователя, агрегатору может потребоваться совершить десятки или даже сотни запросов к сторонним ресурсам. В силу того, что пользователи могут запрашивать одну и ту же информацию, самым распространённым способ снижения времени ответа является использование промежуточных буферов, так называемых кэшей. Но данный способ приводит к снижению актуальности предоставляемых агрегатором данных и может быть использован для предупреждения избыточной нагрузки системы, либо для хранения редко обновляемых данных, таких как словари. Словарь – это набор статических данных используемых агрегатором для объединения информации получаемой с нескольких источников (например, для агрегатора отелей словарём будет являться список всех отелей по которым осуществляется поиск).

Таким образом, разработка алгоритма распределения нестационарной нагрузки, определяющего источник данных агрегатора, является актуальной задачей. Однако для её решения необходим анализ или прогноз нагрузки системы.

Степень теоретической разработанности темы. В открытом доступе существует множество научных работ, описывающих методы прогнозирования временных рядов. Но материалы, описывающие проблему выбора метода прогнозирования для разработки алгоритма распределения нестационарной нагрузки, найти не удалось. Из сказанного выше можно сделать вывод о том, что рассматриваемая тема имеет низкую степень теоретической проработанности.

Объектом исследования является прогнозирование нестационарной нагрузки сервиса-агрегатора.

Предметом исследования являются методы прогнозирования временных рядов.

Область исследования. Проведённое исследование методов прогнозирования нагрузки вычислительной системы в пределах разработки алгоритма рас-

пределения запросов пользователей полностью соответствует специальности «Вычислительные машины, комплексы, системы и сети», а содержание выпускной квалификационной работы – техническому заданию.

Цель и задачи исследования. Целью работы является улучшение качества обслуживания пользователей сервиса-агрегатора за счет снижения среднего времени ответа.

Для достижения данной цели были поставлены следующие задачи:

1. Выполнить сравнительный анализ методов прогнозирования.
2. Разработать алгоритм распределения нестационарной нагрузки и его программную реализацию на основе выбранного метода прогнозирования.
3. Выполнить сравнение среднего времени ответа исходной и использующей разработанный алгоритм систем.

Теоретическую основу исследования составляют научные труды отечественных и зарубежных авторов в области компьютерных технологий и математической статистики.

Методологическую основу исследования составляет эксперимент.

Научная новизна работы заключается в следующем:

1. В настоящее время не существует структурированных рекомендаций по выбору метода прогнозирования при разработке алгоритмов распределения нагрузки, представленных в данной работе.
2. Разработанный алгоритм распределения нестационарной нагрузки может обеспечить улучшение качества обслуживания пользователей сервиса-агрегатора.

Практическая значимость данной работы заключается в том, что разработанный алгоритм распределения нестационарной нагрузки будет интегрирован в разрабатываемый сервис-агрегатор. Кроме того, сформулированные рекомендации могут быть использованы для осуществления выбора метода прогнозирования при разработке собственного алгоритма распределения запросов пользователей.

Апробация результатов исследования. Сформулированные рекомендации по выбору метода прогнозирования обсуждались на VII Конгрессе молодых учёных, а их сравнительный анализ был представлен на XLVII научной и учебно-методической конференции.

Объем и структура работы. Выпускная квалификационная работа содержит 40 страниц машинописного текста, восемь рисунков, семь таблиц, девять формул и список литературы, включающий 25 источников. Структурно работа состоит из введения, трёх разделов и заключения. Во введении обоснована актуальность выбранной темы, а также новизна исследования. Определены цели и задачи, объект и предмет исследования. Представлена апробация результатов исследования. Первый раздел содержит обзор предметной области, рассматриваются широко используемые методы прогнозирования временных рядов. Проводится сравнительный анализ методов и формулируются рекомендации по выбору моделей прогнозирования. Второй раздел посвящён процессу разработки алгоритма распределения нестационарной нагрузки, в нём описывается составление требований для нового алгоритма, а также проблемы его разработки. Приведено описание разработанных алгоритмов аккумуляции обучающей выборки и распределения запросов пользователей. Третий раздел посвящен тестированию разработанного комплекса. Приводятся результаты тестирования под двумя типами нагрузок. В заключении сформулированы основные результаты работы.

1 СРАВНИТЕЛЬНЫЙ АНАЛИЗ МЕТОДОВ ПРОГНОЗИРОВАНИЯ

Необходимость в точном прогнозе возникает во многих областях науки, индустрии, коммерческой и экономической деятельностях. Ситуации его использования весьма разнообразны, и в зависимости от исходной проблемы, прогнозирование может осуществляться вперед на несколько лет, а может на несколько секунд.

Прогнозирование является важной составляющей эффективного планирования ресурсов. В качестве примера может быть приведено прогнозирование электрических нагрузок в электроэнергетике: на основе предсказанных величин рассчитываются оптимальные режимы электро-энергетических систем, а точность прогноза влияет на экономичность загрузки генерирующего оборудования, и, следовательно, на стоимость электроэнергии.

Метод прогнозирования - это процедура вычисления прогноза из текущих и прошлых значений. Это может быть лишь алгоритмическое правило не зависящее от базовой вероятностной модели, поведение которой необходимо предсказать. В другом случае, метод может использоваться с целью определения конкретной модели для предоставленных данных или поиска оптимальных условий прогнозирования для такой модели. Поэтому стоит различать понятия «модель» и «метод» [1].

Существует широкий спектр методов прогнозирования, по оценкам зарубежных и отечественных систематиков прогностики их насчитывается уже свыше ста [2]. Каждый имеет собственные характеристики (такие как точность или вычислительная сложность), которые должны быть учтены при выборе метода.

Методы прогнозирования можно разделить на три основных типа [1, 3, 4, 5]:

1. *Методы суждений* основанные на субъективных суждениях, интуиции, «внутренних» коммерческих знаниях, или любой другой подобной информации.

2. *Одномерные методы* где предсказание зависит только от прошлых и текущего значений предсказываемого ряда.

3. *Многомерные методы* в которых прогнозирование величины, по крайней

мере частично, так же зависит от одного или бóльшего количества временных рядов. Такую совокупность, состоящую из нескольких рядов, называют многомерными временными рядами [6].

Все рассматриваемые в данной главе методы осуществляли прогнозирование поведения временных рядов. Временной ряд - это ряд наблюдений, произведенных последовательно во времени [1, 3, 4, 7, 8, 5]. Он включает два обязательных элемента: время и конкретное значение показателя, или уровень ряда [6, 9].

Временным рядом можно назвать огромное количество последовательностей, являющихся следствием измерений некоторого показателя. Например, показатели (характеристики) экономических, природных, промышленных, информационных и других систем. Фактически, временной ряд может быть построен тремя способами: путем отбора значений из непрерывного ряда, путем агрегирования данных за установленные периоды времени, и взятием серии дискретных наблюдений [1, 7]. Для всех трех типов запись данных производится с равными промежутками времени.

Временные ряды подразделяются на стационарные и нестационарные. Ряд называется стационарным, если его среднее значение, дисперсия и ковариация не зависят от времени [6]. Другими словами, это ряд параметры которого не зависят от того, в какой момент времени производились замеры [5]. Ряд не соответствующий данным условиям называется нестационарным.

Одним из наиболее важных видов временных рядов является последовательность некоррелированных (не взаимосвязанных) случайных величин с нулевым математическим ожиданием и постоянной дисперсией [10, 11], такой ряд называется белым шумом.

Далее в главе производится сравнительный анализ наиболее популярных одномерных и многомерных методов прогнозирования. Методы суждений не рассматриваются в виду того, что их невозможно автоматизировать.

1.1 Методы прогнозирования временных рядов, методология ARIMA

Среди методов прогнозирования наибольшее распространение получила методология Бокса – Дженкинса или ARIMA (autoregressive integrated moving average: интегрированная модель авторегрессии – скользящего среднего). Класс моделей ARIMA - это важный инструмент прогнозирования и базис многих фундаментальных идей в анализе временных рядов [1]. Он удобен тем, что сочетает в себе модели ARMA, AR, MA и может использоваться для прогнозирования как стационарных так и нестационарных временных рядов. Данным классом учитывается связь между случайными остатками временного ряда, которые можно получить вычтя из исходного ряда его неслучайную составляющую (тренд) [2]. Далее в разделе представлено описание моделей.

1.1.1 Модели скользящего среднего MA(q)

В моделях скользящего среднего текущее значение ряда представляется в виде линейной комбинации текущего и прошедших значений ошибки ε_t, ε_{t-1}, ..., ε_{t-q}, по своим свойствам соответствующей «белому шуму». Модель скользящего среднего порядка q может быть выражена следующим уравнением [11]:

$$y_t = \varepsilon_t - \gamma_1\varepsilon_{t-1} - \gamma_2\varepsilon_{t-2} - \ldots - \gamma_q\varepsilon_{t-q}, \tag{1.1}$$

где:

y_t – значение ряда в момент времени t

$\gamma_1, \gamma_2, \ldots, \gamma_q$ – параметры модели,

ε_t – случайные ошибки образующие «белый шум».

В эквивалентной форме:

$$y_t = (1 - \gamma_1 B - \gamma_2 B^2 - \ldots - \gamma_q B^q)\varepsilon_t, \tag{1.2}$$

или:

$$y_t = \gamma(B)\varepsilon_t, \tag{1.3}$$

где B - оператор сдвига назад:

$$By_t = y_{t-1}.$$

Таким образом, процесс скользящего среднего можно трактовать как выход y_t линейного фильтра с передаточной функцией $\gamma(B)$, на вход которого подается процесс белого шума ε_t [7].

1.1.2 Модели авторегрессии AR(p)

Модель авторегрессии порядка p может быть представлена в следующем виде [11]:

$$y_t = \beta_1 y_{t-1} + \beta_2 y_{t-2} + ... + \beta_p y_{t-p} + \varepsilon_t, \tag{1.4}$$

где $\beta_1, \beta_2, ..., \beta_p$ – параметры модели. Таким образом прогнозируемое значение представляется в виде линейной зависимости от p известных значений временного ряда.

В эквивалентной форме:

$$y_t = \frac{\varepsilon_t}{1 - \beta_1 B - \beta_2 B^2 - ... - \beta_p B^p}, \tag{1.5}$$

или:

$$y_t = \beta^{-1}(B)\varepsilon_t, \tag{1.6}$$

отсюда процесс авторегрессии может быть интерпретирован как выход y_t линейного фильтра с передаточной функцией $\beta^{-1}(B)$, на вход которого подается процесс белого шума ε_t [7].

1.1.3 Модели авторегрессии – скользящего среднего ARMA(p, q)

Конечный процесс авторегрессии может быть представлен как бесконечный процесс скользящего среднего MA(∞) [12, 1], однако с увеличением порядка модели её расчет значительно усложняется и потому если процесс действительно типа AR, то его представление в виде скользящего среднего не может быть экономичным. Точно так же процесс MA не может быть экономично пред-

ставлен с помощью процесса авторегресии. Для того, чтобы сделать параметризацию более экономичной в модель могут быть включены не только члены моделирующие скользящее среднее, но и члены описывающие авторегрессию.

Общий вид авторегрессии – скользящего среднего ARMA(p, q) определяется следующим уравнением [11]:

$$y_t = \beta_1 y_{t-1} + \beta_2 y_{t-2} + ... + \beta_p y_{t-p} + \varepsilon_t - \gamma_1 \varepsilon_{t-1} - \gamma_2 \varepsilon_{t-2} - ... - \gamma_p \varepsilon_{t-p}, \quad (1.7)$$

Как легко заметить, при порядке p равном нулю будет получен процесс MA, а при обнулении порядка q - процесс AR.

Процесс может быть записан в эквивалентной форме:

$$y_t = \frac{1 - \gamma_1 B - \gamma_2 B^2 - ... - \gamma_q B^q}{1 - \beta_1 B - \beta_2 B^2 - ... - \beta_p B^p} \varepsilon_t, \quad (1.8)$$

то есть, смешанный процесс авторегрессии – скользящего среднего может быть истолкован как выход y_t линейного фильтра, с передаточной функцией в виде отношения двух полиномов, на вход которого подается белый шум ε_t [7].

1.1.4 Модели авторегрессии – проинтегрированного скользящего среднего ARIMA(p, d, q)

Перечисленные выше модели используются для прогнозирования стационарных процессов. Однако, существуют временные ряды, среднее значение которых может изменяться в течение времени. Однако даже они могут демонстрировать однородность. И, возможно, если не учитывать локальный уровень, или локальный уровень и тренд, то любая часть подобного временного ряда может по своему поведению оказаться подобна любой другой части. Сделав предположение, что некая подходящая разность процесса стационарна, можно получить модели, которые будут описывать подобное однородное нестационарное поведение. Модели, в которых d-я разность есть стационарный смешанный процесс авторегрессии – скользящего среднего называются процессами авторегрессии – проинтегрированного скользящего среднего ARIMA(p, d, q) [7]. Данный процесс

может быть представлен уравнением вида:

$$\Delta^d y_t = \beta_1 \Delta^d y_{t-1} + \beta_2 \Delta^d y_{t-2} + ... + \beta_p \Delta^d y_{t-p} + \varepsilon_t - \gamma_1 \varepsilon_{t-1} - \gamma_2 \varepsilon_{t-2} - ... - \gamma_p \varepsilon_{t-p}, \quad (1.9)$$

где Δ^d – оператор разности временного ряда порядка d, например:

$$\Delta y_t = y_t - y_{t-1},$$

для первого порядка;

$$\Delta^2 y_t = \Delta y_t - \Delta y_{t-1} = (y_t - y_{t-1}) - (y_{t-1} - y_{t-2}) = y_t - 2y_{t-1} + y_{t-2},$$

для второго.

1.2 Нейросетевые методы прогнозирования

Искусственные нейронные сети (далее - нейронные сети) возникли на основе знаний о функционировании нервной системы живых существ. Они представляют собой попытку использования процессов, происходящих в нервных системах для выработки новых технологических решений [13].

Основным элементом обработки информации в нейронной сети является модель нейрона. В его основе лежат [14]:

1. Набор связей (connecting link) или синапсов (synapse), при этом каждый синапс имеет свой вес (weight). Например, если на вход синапса j, который связан с нейроном k, поступает сигнал x_j, то этот сигнал умножается на вес w_{kj}. При этом веса могут иметь как положительное, так и отрицательное значение.

2. Сумматор, который складывает все входные сигналы перемноженные на соответсвующие им веса.

3. Функция активация (activation function), ограничивающая амплитуду выходного сигнала нейрона. Как правило, значение на выходе нейрона лежит в интервале $[0, 1]$ или $[-1, 1]$. Обычно выделяют 3 основных типа активационных функций: единичного скачка, кусочко-линейные и сигмоидальные. Наибольшее распространение получили последние.

Одним из важнейших преимуществ нейронных сетей является возможность обучения. Обучение можно рассмотреть как корректировку весов сети, выполняемую по обучающим примерам (или обучающим данным), в следствие которой сеть меняет свою реакцию на входные воздействия.

Следует отметить, что нейронные сети позволяют получить результат на ранее не виденных примерах данных. Поэтому нейросетевые методы хорошо себя зарекомендовали как средство моделирования динамических систем при неизвестной априори математической модели динамической системы. Существует два базовых метода наделения нейронных сетей свойствами, необходимых для прогнозирования поведения динамических систем: добавление линий задержек и добавление рекуррентных связей [15]. Оба метода могут использоваться как одномерные или двумерные и потому представляют особый интерес в данной работе. Их описание представлено в разделе далее.

1.2.1 Многослойный персептрон с линией задержек

В многослойном персептроне (Multilayer Perceptron, MLP) с линией задержек все нейроны расположены слоями, при этом имеется один входной слой, один выходной и как минимум один скрытый слой. Пример схемы данной сети порядка N и одним скрытым слоем изображен на рисунке 1.1. Сеть содержит нейроны с линейной функцией активации во входном слое и нейроны с сигмоидальной функцией активации в скрытом и выходном слоях. Весовые коэффициенты задаются матрицами W_1 и W_2. На вход нейронная сеть получает текущее значение временного ряда $y(k)$, а так же задержанные значения $y(k-1)$, $y(k-2)$, ... $y(k-N)$, полученные при помощи элементов запаздывания Z^{-1}, Z^{-2}, ... Z^{-N}. По полученным данным сеть обучается делать прогноз следующего значения временного ряда $y(k+1)$.

1.2.2 Рекуррентный многослойный персептрон

Рекуррентный многослойный персептрон (Recurrent Multilayer Perceptron, RMLP) отличается от многослойного персептрона тем, что его выходные значе-

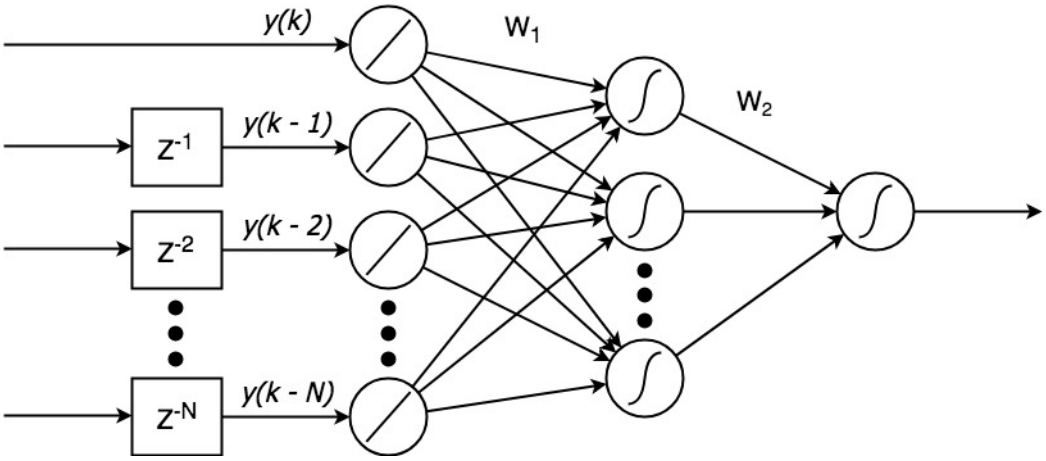

Рисунок 1.1 — Схема многослойного персептрона с линией задержек порядка N

ния зависят не только от значений на входе в данный момент времени, но и от предыдущих входных значений или состояния сети. Поэтому данный вид нейронных сетей получил широкое распространение в управляющих приложениях и приложениях направленных на обработку сигналов [16]. Схема персептрона с элементами задержки первого порядка, используемого для прогнозирования временного ряда, изображена на рисунке 1.2. Как можно заметить, в матрице весов скрытого слоя W_1 теперь хранятся еще и веса каждой из рекуррентных связей.

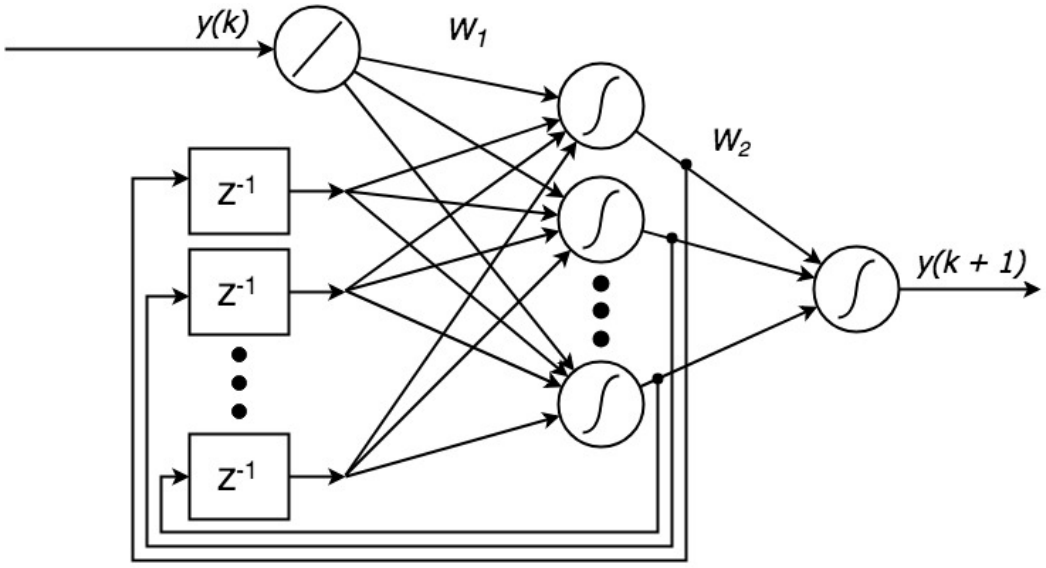

Рисунок 1.2 — Схема рекуррентного многослойного персептрона

1.3 Сравнение моделей прогнозирования

Так как выбранный в результате сравнительного анализа метод будет использоваться для предупреждения избыточной нагрузки, то ко всем моделям в данном разделе предъявлялось требование: ошибка предсказания пиков избыточной нагрузки (то есть перегруженного состояния системы, при отсутствие перегрузки на момент расчета прогноза) должна составлять не более 5 %.

1.3.1 Подготовка данных для сравнения

Для сбора данных был построен тестовый стенд, включающий в себя две вычислительные системы, находящихся в одной локальной сети. Первая под управлением операционной системы Ubuntu 16.04.3 LTS, вторая под управлением операционной системы macOS High Sierra 10.13.3. Аппаратное обеспечение второй вычислительной системы:

- *центральный процессор* Intel Core i5-4258U3,
- *оперативная память* 8 GB 1600 MHz DDR3,
- *внешняя память* SSD 128 GB.

На вычислительных ресурсах второй системы был развернут разрабатываемый сервис агрегатор, а так же система управления базами данных для его работы. Сервис принимал HTTP-запросы, осуществлял необходимые для их выполнения действия и выдавал результаты. Кроме того, сервис в режиме реального времени осуществлял сбор статистических данных и сохранял их в файле в формате tsv (tab separated values – значения, разделённые табуляцией). Схема тетового стенда изображена на рисунке 1.3.

Так как сервис разрабатывается на Java, сбор информации о загрузке процессора осуществлялся при помощи компонента Java платформы com.sun.management. Данный компонент позволяет получить информацию о загрузке процессора от операционной системы [17]. При помощи компонента java.lang.Runtime сервисом производился сбор информации об объёме памяти доступной виртуальной машине Java [18].

Рисунок 1.3 — Схема тестового стенда

Каждые 300 мс сервисом производился сбор семнадцати статистических характеристик, которые возможно разделить на пять типов:

1. **Количество пришедших запросов с момента предыдущего замера.**
   У разрабатываемого сервиса насчитывается шесть видов запросов пользователей, для каждого выделено по одному отдельному счетчику.

2. **Количество загрузчиков внешних ресурсов.**
   При поисковом запросе производится запуск процесса загрузки для необходимых внешних ресурсов. По одной количественной характеристике выделено для каждого внешнего ресурса, всего две.

3. **Среднее время обработки запроса с предыдущего замера по текущий момент времени.**
   Аналогично первому типу – шесть видов запросов.

4. **Загрузка процессора.**
   Две отдельные характеристики: загрузка процессора, оказываемая процессом и общая загрузка процессора.

5. **Объем доступной памяти.**
   Одна характеристика: объем памяти доступной виртуальной машине Java.

Характеристики первого и второго типа предоставляют информацию о том, на что именно расходуются вычислительные ресурсы системы. По общей тенден-

ции данных временных рядов может быть предсказан факт снижения или увеличения нагрузки системы. Характеристики третьего и четвёртого типов широко используются в нагрузочном тестировании как основные показатели производительности системы [19] и потому могут быть использованы при прогнозировании её перегрузки. На основании значений пятой характеристики может быть спрогнозирован запуск сборщика мусора виртуальной машиной Java, данное действие оказывает значительное влияние на производительность системы [20]. Графики со статистическими данными изображены на рисунке 1.4.

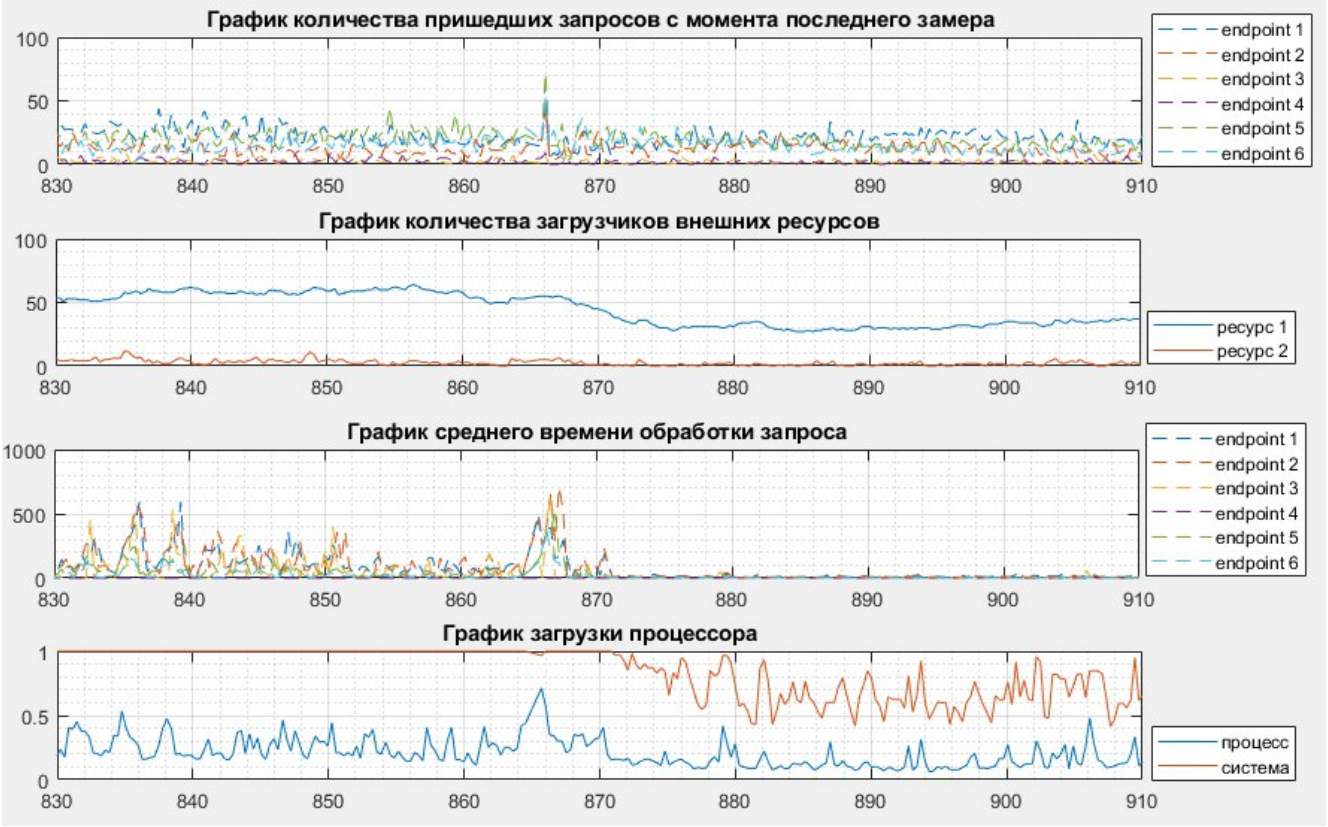

Рисунок 1.4 — Статистические данные сервиса-агрегатора

Генерация HTTP-запросов осуществлялась средствами Apache JMeter. Был составлен тестовый план, потоки которого имитировали работу браузера пользователя, производя следующие действия:

1. Запрос начальной информации, необходимой для отображения стартовой страницы.
2. Отправка запросов, имитирующих «живой поиск» по словарям агрегатора.
3. Запрос на поиск со случайными параметрами.

4. Отправка периодичных запросов в ожидании завершения поиска.
5. Отправка запросов на получение результатов поиска.

Временные интервалы между запросами устанавливались согласно правилам:

- среднее время позиционирования курсора и нажатия на клавишу «мыши» составляет 0.4 с. [21],
- среднее время ввода символа при помощи клавиатуры составляет 0.25 с. [21],
- при ожидании завершения поиска период отправки запросов составляет 1 с.

Варьирование нагрузки производилось путем изменения числа одновременно исполняемых потоков. Было произведено стресс-тестирование с целью определения числа потоков, приводящего систему к перегрузкам. Считалось, что система подвержена избыточной нагрузке, если среднее время выполнения любого из шести видов запросов составляло больше 90 мс. Во время тестирования каждые 2 часа происходило увеличение активных потоков на 10, затем рассчитывался процент времени перегрузки системы (например при 470 активных потоках система находилась в состоянии перегрузки 6.31 % времени). Всего на тестирование было затрачено около трёх суток.

Был написан awk-скрипт, генерирующий расписание активных потоков для плагина Apache JMeter – Ultimate Thread Groups. При генерации соблюдалось ограничение на число активных потоков: во время стрессового тестирования было установлено, что при 560 активных потоках система более 95 % (98.14 %) времени находилась в состоянии перегрузки, поэтому большее количество потоков не использовалось. Следует отметить, что все расписание может быть условно разделено на фазы трех типов: возрастание числа активных потоков, удержание числа потоков на постоянном уровне и уменьшение числа потоков. Посредством переключения фаз производится перевод системы из неперегруженного состояния в перегруженное и обратно. В начале главы упоминалось, что данному поведению системы уделяется особое внимание при прогнозировании.

Пример расписания активных потоков изображен на рисунке 1.5, где по оси X – время прошедшее с начала теста, а по оси Y – количество потоков.

Рисунок 1.5 — График количества активных потоков в расписании тестового плана

1.3.2 Сравнение моделей прогнозирования ARIMA

Ранее в главе был дан краткий обзор моделей прогнозирования ARIMA, и как можно отметить, один из основных недостатков данных моделей – это необходимость повторного расчета коэффициентов при получении новых данных, что может негативно сказаться на производительности системы (так как значимая доля её процессорного времени будет уделяться данной проблеме) [2]. Поэтому порядок моделей и количество обучающих данных были ограничены.

Для расчета ARIMA-моделей была использована Java-библиотека с открытым исходным кодом – Workday/timeseries-forecast. На одних и тех же входных данных по загрузке процессора, собранных за один час работы системы (процесс сбора данных описан в предыдущем разделе), были построены модели с порядками $p \leq 11$, $d \leq 3$ и $q \leq 11$. Количество обучающих данных n варьировалось от 10 до 4000.

Оценка точности модели производилась по следующему алгоритму:

- рассчитывалась ARIMA модель с порядками p, d и q на последовательных значениях загрузки с i-го по $i + n - 1$

- рассчитывался прогноз f следующего шага – $i + n$
- происходила нормализация спрогнозированного значения по следующему правилу: если $f < 0$, то $f_n = 0$; если $f > 1$, то $f_n = 1$
- фиксировалась ошибка – разница между фактическим значением временного ряда и нормализованным значением f_n
- выполненные действия повторялись при $i = i + 1$

После завершения работы данного алгоритма вычислялась средняя квадратическая ошибка (root mean square error RMSE, данный способ оценки точности часто используется для сравнения моделей пронозирования [22, 23, 5]) модели порядков p, d, q с количеством обучающих данных равным n. Результаты, отсортированные в порядке возрастания ошибки, представлены в таблицах 1.1–1.4. Результаты прогнозов наиболее точных моделей AR, MA, ARMA и ARIMA изображены на рисунке 1.6.

Таблица 1.1 — Сравнение моделей MA

Порядок модели MA (q)	Количество обучающих данных (n)	Средняя квадратическая ошибка
1	15	0.116628
1	16	0.116784
1	11	0.116977
1	13	0.117005
1	18	0.117108
1	20	0.117462
1	19	0.117549
1	14	0.117555
1	17	0.117649
1	22	0.117656

Таблица 1.2 — Сравнение моделей AR

Порядок модели AR (p)	Количество обучающих данных (n)	Средняя квадратическая ошибка
9	800	0.103868
8	800	0.103885
8	700	0.103948
9	700	0.103956
10	800	0.103992
10	700	0.104136
7	800	0.104277
9	900	0.104283
8	900	0.104333
7	700	0.104349

Таблица 1.3 — Сравнение моделей ARMA

Порядки модели ARMA (p, q)	Количество обучающих данных (n)	Средняя квадратическая ошибка
8, 1	800	0.104518
9, 1	800	0.104555
10, 1	700	0.104639
9, 1	700	0.104718
8, 1	700	0.104766
10, 1	900	0.104776
10, 1	800	0.104785
9, 2	700	0.104795
9, 3	800	0.104845
9, 1	900	0.104894

Таблица 1.4 — Сравнение моделей ARIMA

Порядки модели ARIMA (p, d, q)	Количество обучающих данных (n)	Средняя квадратическая ошибка
0, 1, 3	700	0.103599
7, 1, 0	700	0.103604
8, 1, 0	700	0.103613
8, 1, 0	800	0.103665
7, 1, 0	800	0.103714
7, 1, 0	600	0.103752
0, 1, 3	800	0.103774
8, 1, 0	600	0.103777
10, 1, 0	800	0.103798
9, 1, 0	700	0.103804

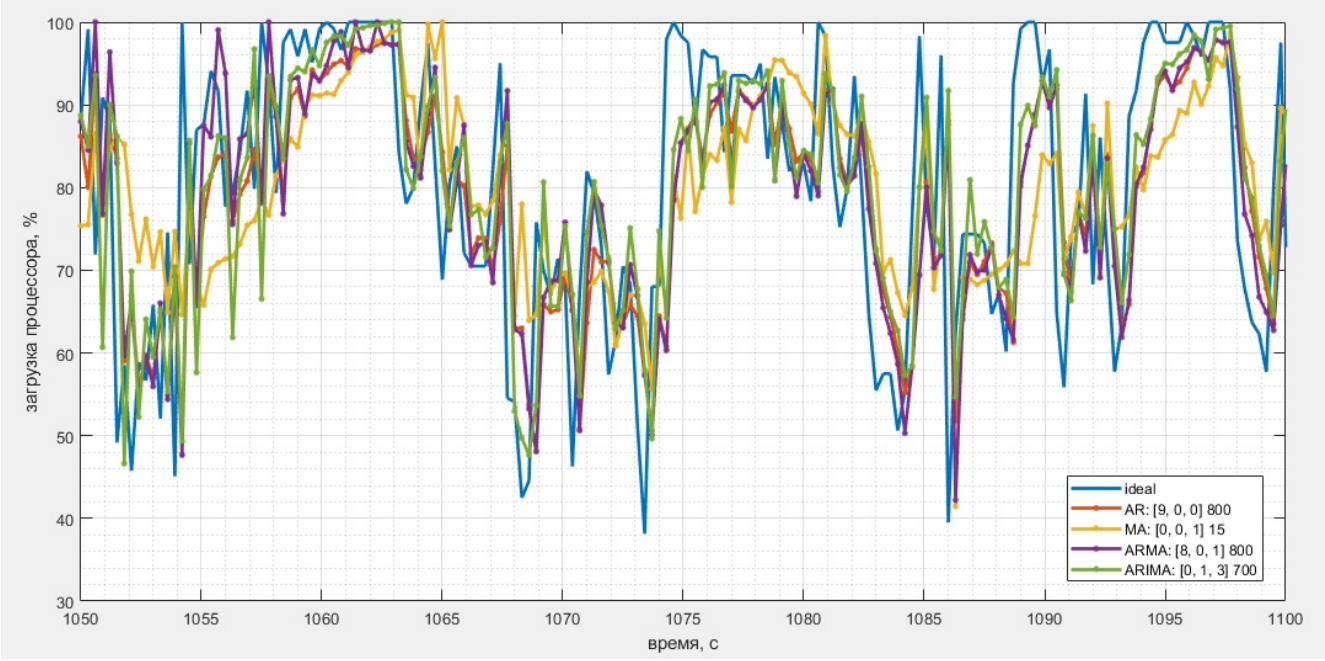

Рисунок 1.6 — Прогноз загрузки процессора

На рисунке 1.6 можно отметить, что резкие скачки нагрузки не прогнозируются ни одной из моделей. Кроме того наиболее точные модели вычисляют повторение исходного ряда, но смещенного по времени. По сути значения рас-

считанные любой из вышеперечисленных моделей нельзя считать прогнозом. Совокупность данных фактов не соответствует основному требованию прогнозирования пиков избыточной нагрузки, приведённому в начале раздела.

1.3.3 Сравнение нейросетевых моделей

Ранее в главе были представлены два типа нейронных сетей, используемых для прогнозирования временных рядов. Следует выделить их особенности [15]:

1. При использовании многослойного персептрона, в отличие от рекуррентного, необходимо заранее знать порядок линии задержек на входе. Однако, в пределах данной работы это не является проблемой, так как порядок может быть выбран во время экспериментов, и больше не меняться в ходе работы системы.

2. Рекуррентные сети имеют бóльшую точность при многошаговом прогнозировании, но это так же не имеет значения в данной работе.

3. Обучение рекуррентных сетей является более трудоемкой задачей, в виду наличия у них дополнительных степеней свободы [16, 24].

Из этого следует, что при выборе между двумя типами нейронных сетей наибольший интерес (на основании теоретических сведений) представляет первый.

Но наличие рекуррентных связей является не единственной структурной конфигурацией, приводящим к изменению точности прогнозирования. Влияние оказывают и другие параметры, такие как: количество слоёв, количество нейронов в скрытых слоях, а так же активационные функции нейронов. Более того, при изменении алгоритма обучения можно добиться различной сходимости, а следовательно – и точности [24].

Для составления и обучения нейронной сети был использован Java-фреймворк с открытым исходным кодом – Encog. В качестве обучающего был взят тот же участок данных, что использовался при построении моделей ARIMA в предыдущем разделе. На вход нейронной сети подавались значения семнадцати динамических характеристик в момент времени t, а также $n - 1$ предыдущих

значений каждой из характеристик для сети с линией задержек порядка n (более подробное описание данных характеристик представлено в подразделе 1.3.1). Если при $t + 1$ система находилась в состоянии перегрузки, то на выходе сети ожидалось значение 1. В обратном случае – 0.

Всего было обучено около десяти тысяч нейронных сетей различных конфигураций. При этом количество скрытых слоёв варьировалось от одного до трёх, а количество нейронов в скрытых слоях от одного до количества нейронов предыдущего слоя. Кроме того, изменялись функции активации скрытого и выходного слоёв (всего использовалось 3 типа функций: сигмоидальная, гиперболического тангенса и логарифмическая), а так же порядок линии задержек от одного до десяти для нейронной сети с линией задержек. При обучении использовался метод Resilient Propagation (упругого распространения), как показавший наибольшую точность при решении подобных задач [24]. Обучение продолжалось до двадцати тысяч эпох. Однако при обучении рекуррентной нейронной сети не удалось добиться схождения.

Суммарно на обучение нейронных сетей было затрачено около трёх недель. В виде тестовых данных был взят еще один участок статистических данных длиной один час. Точность оценивалась по факту значения ошибки – числа неверно спрогнозированных бинарных значений относительно общего числа прогнозов на обучающих и тестовых данных. Десять конфигураций сетей отсортированные в порядке возрастания ошибки на обучающих данных и соответствующие требованиям точности предсказания пиков избыточной нагрузки представлены в таблице 1.5.

1.4 Результаты сравнительного анализа

В результате сравнительно анализа были сделаны следующие выводы:

1. Модели AR, MA, ARMA, ARIMA не подходят для прогнозирования нагрузки процессора, в силу недостаточной точности на данном виде временного ряда.

2. Ошибка прогнозирования пиков избыточной нагрузки части моделей прогнозирования, составленных на базе многослойного пересептрона, меньше

Таблица 1.5 — Сравнение нейросетевых моделей

Конфигурация нейронной сети			Ошибка на обучающей выборке, %	Ошибка на тестовых данных, %	Ошибка предсказания пиков избыточной нагрузки на тестовых данных, %
Порядок линии задержек	Количество нейронов в 1-м, 2-м и 3-м слоях	Функция активации			
8	80, 48, 16	log	0.08	2.53	1.86
7	63, 35, 14	log	0.13	2.22	2.05
7	70, 28, 14	log	0.14	2.31	2.05
8	72, 40, 8	log	0.14	2.42	2.79
6	60, 24, 6	log	0.14	2.79	2.61
7	70, 35, 7	log	0.15	2.43	2.42
6	60, 24, 18	log	0.21	2.28	3.35
6	54, 24, 18	log	0.27	2.11	4.10
6	60, 30, 6	log	0.29	2.16	4.84
7	56, 35, 7	log	0.35	2.62	4.84

5 % на обучающей выборке, что соответствует требованиям. Однако при разработке алгоритма распределения нагрузки необходимо расширить обучающую выборку.

Так же были составлены следующие рекомендации по выбору моделей прогнозирования:

- При необходимости прогнозирования поведения рядов, имеющих плавный и непрерывистый характер, наиболее подходящим выбором будут модели класса ARIMA.
- Если при этом ряд стационарен, то будет достаточно класса ARMA.
- При выборе модели класса ARIMA в первую очередь необходимо проверить точность моделей низших порядков.
- Если необходимо автоматизировать процесс выбора модели поведения на основании значений одномерного или многомерного временного ряда, то в

таком случае подойдут модели составленные на базе многослойного персептрона.

- Если требуется модель, вычисляющая многошаговый прогноз, то наиболее подходящими являются модели, составленные на базе рекуррентного многослойного персептрона.

2 РАЗРАБОТКА АЛГОРИТМА РАСПРЕДЕЛЕНИЯ НЕСТАЦИОНАРНОЙ НАГРУЗКИ

2.1 Необходимые функциональные возможности

Для разрабатываемого алгоритма были определены необходимые функциональные возможности, а именно:

1. Формирование решения о необходимости приостановки запуска новых загрузчиков внешних ресурсов и передачи пользователям данных из кэша, на основании предоставленных алгоритму статистических данных. При таком подходе система не будет тратить свои вычислительные ресурсы на загрузку новых данных из внешних ресурсов, что увеличит её производительность и уменьшит среднее время ответа.
2. Возможность продолжения обучения лежащей в основе алгоритма нейронной сети. За расширением обучающей выборки и продолжением обучения нейронной сети следует увеличение точности определения перегрузок вычислительной системы, что значительно улучшает эффективность работы алгоритма.

2.2 Требования к алгоритму

Так как алгоритм должен быть встроен в разрабатываемый сервис, были сформированы следующие требования:

1. Практическая реализация разработанного решения должна быть осуществлена на языке Java SE 8, так как серверная часть разрабатываемого сервиса уже реализована на языке Java версии 8 и это обеспечит простую встраиваемость.
2. Алгоритм должен быть представлен в виде компонента Spring Framework, чтобы обеспечить единообразность кода.
3. Обученная нейронная сеть, а так же её обучающая выборка должны храниться на диске, в силу того, что операции чтения или записи этих данных

при стабильной работе сервиса будут происходить не больше двух – трёх раз в сутки.

4. Все настройки алгоритма должны задаваться при помощи файлов свойств Spring Framework (по умолчанию application.properties или application.yml), так как все конфигурационные параметра указываются именно таким способом.

2.3 Разработка алгоритма

После внедрения алгоритма распределения запросов поведение разрабатываемого сервиса изменится, в силу следующих причин:

1. Каждые 300 мс. вместе с получением новых данных будет рассчитываться прогноз, а точнее производиться перемножение весовых коэффициентов каждого слоя нейронной сети на значения полученные из функций активации предыдущих слоев. Это займет часть процессорного времени вычислительной системы.

2. Если нейронная сеть спрогнозирует перегрузку системы, то результаты поиска не будут загружаться из внешних ресурсов и пользователи сразу же получат результаты найденные в кэше. В таком случае нагрузка системы снизится.

Поэтому данные необходимые для обучения нейронной сети могут быть получены только при её функционировании в составе сервиса. Тогда на вход нейронной сети с линией задержек порядка n так же может подаваться восемнадцатый временной ряд – предыдущие n бинарных значений с выхода нейронной сети. Анализ данного ряда позволит определить снижение нагрузки, которое может оказать влияние на прогноз поведения системы.

Для решения проблемы недостаточности обучающей выборки нейронной сети, описанной в конце подраздела 1.3.3, был разработан алгоритм аккумулирования обучающей выборки, который позволил не перегружать обучающую выборку большим объемом данных, и в то же время постепенно увеличивать

точность прогнозирования посредством её дополнения и повторного обучения нейронной сети. Алгоритм представлен ниже:

1. Формирование тестовой выборки из собранных статистических данных за прошедшие n дней.
2. Расчет прогноза нейронной сетью на тестовой выборке.
3. Дополнение обучающей выборки, сохранённой после предыдущего запуска данного алгоритма, при ошибочном прогнозе (либо пустой обучающей выборки, если алгоритм запущен впервые).
4. Обучение нейронной сети на дополненной обучающей выборке.
5. Фиксирование в памяти обученной нейронной сети, её обучающей выборки и точности на тестовой выборке.
6. Если не израсходовано время отведённое на отработку алгоритма, переход к пункту 2.
7. Сохранение нейронной сети с наибольшей точностью, а так же её обучающей выборки.

Данный алгоритм может запускаться в определенные часы ночью, пока сервис простаивает.

После встраивания обоих алгоритмов в разрабатываемый сервис, было произведено дополнительное сравнение пяти конфигураций нейронных сетей с наибольшей точностью из подраздела 1.3.3. В связи с добавлением восемнадцатой характеристики, количество нейронов скрытых слоёв было увеличено пропорционально увеличению числа нейронов входного слоя. Для сравнения использовался модифицированный тестовый стенд из подраздела 1.3.1, представленный на рисунке 2.1, где для работы распределителя запросов пользователей использовался разработанный алгоритм распределения запросов. При сравнении был использован тот же тестовый план для Apache JMeter, что и в подразделе 1.3.1. С целью уменьшения времени затрачиваемого на сбор данных и обучение нейронных сетей, было решено сократить время сбора данных с дневного интервала до четырёх часов. Чтобы проверить обучаемые сети на сходимость,

обучение продолжалось до тех пор, пока ошибка на обучающей выборке не опускалась до величины меньшей чем 0.5 %. Сбор данных и последующее за ним обучение проводились пять раз.

Рисунок 2.1 — Схема тестового стенда

Первая нейронная сеть была обучена тремя способами:

1. Тестовая выборка состояла из статистических данных собранных за последние четыре часа работы сервиса, или один временной промежуток сбора данных. Ошибка прогнозирования полученной в результате обучения сети составила 0.67 %, а суммарное время обучения — 35 часов 12 минут.

2. Тестовые данные были собраны за восемь часов работы сервиса, или два временных промежутка сбора данных. Ошибка прогнозирования: 0.54 %, суммарное время обучения: 28 часов 27 минут.

3. Три временных промежутка сбора данных. Ошибка прогнозирования: 0.48 %, суммарное время обучения: 24 часа.

В результату сравнения трёх способов обучения, для обучения остальных конфигураций нейронных сетей был выбран второй способ, так как он дает значительный прирост скорости обучения и точности. Процент ошибки уменьшился на 0.13 %, а время обучения на 6 часов 45 минут, (или на 19.4 % и 19.2 % относительно исходной величины).

На рисунке 2.2 можно увидеть графики обучения данной сети вторым способом. В легенде указано на каком промежутке данных производится расчёт ошибки прогнозирования (как описано в подразделе 1.3.1: сбор статистических данных производится раз в 300 мс., поэтому участок $(1 - 48000)$ соответствует четырёх часовому периоду). Одна итерация – это выполнения пунктов 2–5 аккумулирующего алгоритма. Сплошными линиями на графике обозначен процент ошибки на тестовой выборке; пунктирными – на данных, ранее побывавших в тестовой выборке.

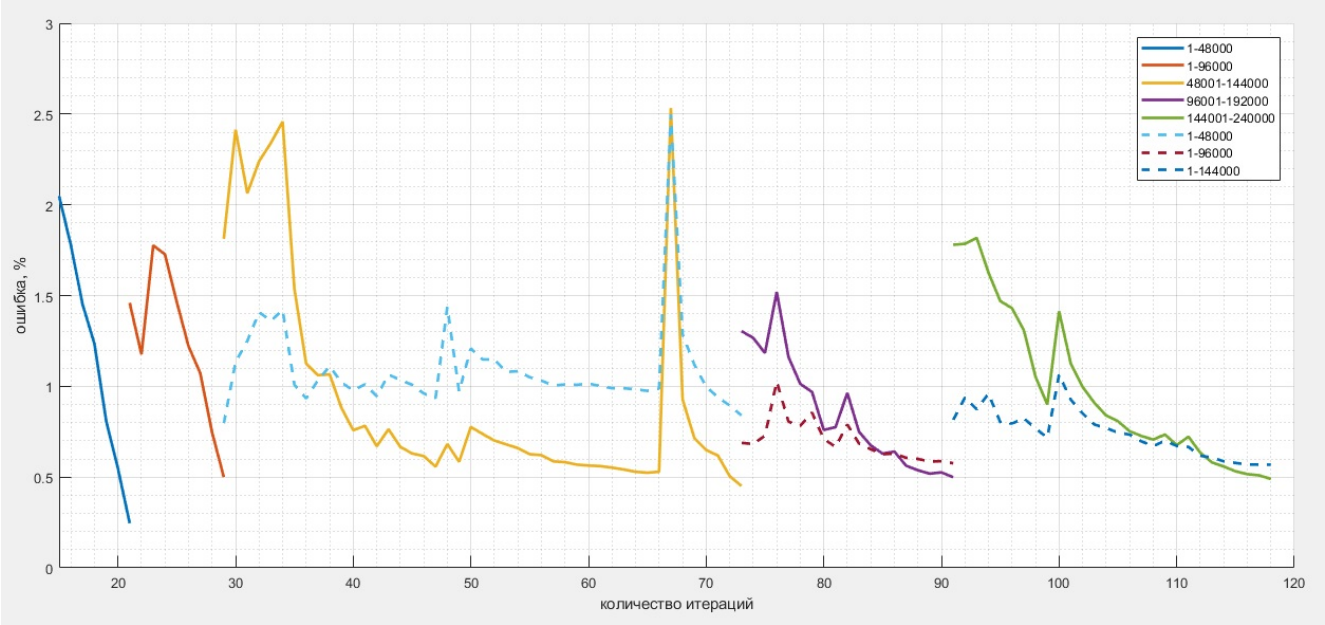

Рисунок 2.2 — Графики итерационного обучения нейронной сети при помощи разработанного аккумулирующего алгоритма

Из графиков видно, что процесс обучения нейронной сети сходится, а процент ошибки на предшествующих тестовых данных к концу обучения не превысил 1 %. При наличии ограничения на время обучения, процент ошибки будет постепенно уменьшаться за счет выбора наиболее точной сети.

Всего на сбор данных и обучение нейронных сетей было затрачено около двенадцати дней. Быстрее остальных была обучена сеть с линией задержек шестого порядка, 64 нейронами в первом скрытом слое, 26 во втором и 19 в третьем. Её обучение заняло 25 часов 36 минут, а размер обучающей выборки составил 19542 (примерно 98 минут, или 8.1 % от общего количества данных). Данная нейронная сеть была задействована в конечном варианте разработанного алгоритма распределения нестационарной нагрузки.

2.4 Практическая реализация разработанного алгоритма

В итоге разработанный алгоритм реализует 3 функции:

- *double getError();*
  Возвращает процент ошибки нейронной сети на тестовых данных, рассчитанный во время последнего обучения.

- *boolean isOverloaded();*
  Принимает на вход массив из восемнадцати статистических характеристик и пяти задержанных во времени значений для каждой. Возвращает true, если система нуждается в уменьшении нагрузки.

- *void startTraining();*
  Принимает на вход тестовую выборку, на которой производится обучение сети согласно алгоритму аккумуляции, представленному в предыдущем разделе. Время обучения ограничивается параметром, описанным ниже.

Для составления, сохранения в файл, загрузки из файла и обучения нейронной сети используется Encog Framework.

При помощи файлов свойств Spring Framework могут быть заданы следующие параметры разработанного алгоритма:

- forecasting.path-to-stored-data
  При помощи данного параметра задаётся путь к хранимым алгоритмом данным.

- forecasting.learning-limit
  Задаёт ограничение на время обучения сети.

После каждого обучения сеть, её точность и обучающая выборка сохраняются в файлы, расположенные по указанному пути. При запуске сервиса происходит загрузка нейронной сети и точности из файлов. Если файл с нейронной сетью не обнаружен или поврежден – создаётся новая нейронная сеть. Если значение ошибки на тестовых данных неизвестно, то функцией getError() будет возвращено значение NaN.

3 ТЕСТИРОВАНИЕ РАЗРАБОТАННОГО АЛГОРИТМА

3.1 Тестовое окружение

Для сравнения среднего времени ответа модифицированного и исходного сервисов были использованы тестовые стенды (рисунок 1.3 и рисунок 2.1) представленные в подразделе 1.3.1 и разделе 2.3. Так же были сгенерированы два расписания активных потоков, каждый длительностью 24 часа, с максимальным числом потоков равным 560, и 460 (более подробно процесс генерации описан в подразделе 1.3.1). Так как в первом случае число активных потоков доходило до 560, то при отсутствии алгоритма распределения запросов система 37 % времени тестирования находилась в состоянии перегрузки. Во втором случае, в виду меньшего количества активных потоков, система находилась в состоянии перегрузки чуть более 1 % времени. Наличие двух видов тестов позволяет проверить какое влияние оказывает алгоритм распределения на поведение сервиса при наличии и отсутствии перегрузок.

Каждый из двух вариантов тестов проводился по четыре раза на сервисе с распределителем нагрузки и без него. Измерения времени ответа производились средствами Apache JMeter.

3.2 Результаты тестирования

После проведения тестов было рассчитано среднее время ответа для каждого из видов запросов. Так же была рассчитана величина 95-го перцентиля для первого теста и 99-го для второго [25]. Кроме того, для величин среднего времени ответа был так же рассчитан доверительный интервал с уровнем доверия 95 % [25].

Результаты тестирования представлены в таблицах 3.1 и 3.2. Для обоих тестов в отдельном столбце произведено сравнение среднего времени ответа и перцентелей для исходного сервиса и сервиса с разработанным алгоритмом.

Таблица 3.1 — Результаты первого тестирования

	Вид запроса	Исходный сервис	Сервис с алгоритмом распределения	Уменьшение величины, %
Среднее время ответа, мс	1	133.5 ± 4.5	112.6 ± 3.6	15.7
	2	152.9 ± 5.3	120.3 ± 4.3	21.3
	3	122.5 ± 5.7	98.7 ± 4.1	19.4
	4	23.5 ± 2.7	21.1 ± 1.5	10.2
	5	72.3 ± 4.2	57.8 ± 3.3	20.1
	6	70.6 ± 4.4	57.1 ± 3.5	19.1
95-й перцентиль, мс	1	552	438	20.7
	2	612	436	28.8
	3	581	440	24.3
	4	106	91	14.2
	5	278	204	26.6
	6	270	212	21.5

3.3 Выводы по результатам тестирования

По результатам тестирования можно сделать вывод, что в случае появления значительных перегрузок системы, разработанный алгоритм, при встраивании его в сервис-агрегатор, обеспечивает в среднем для всех видов запросов на 17.6 % меньшее время ответа, а так же уменьшение 95-го перцентиля на 22.7 %. Следует отметить, что работа алгоритма оказывает положительное влияние на время ответа для всех видов запросов без исключения.

Если же система не подвержена постоянным перегрузкам, то алгоритм обеспечивает уменьшение величины среднего времени ответа на 1.1 % и 99-го перцентиля на 7.9 %.

Уменьшение данных характеристик в обоих тестах является показателем того, что алгоритм справляется с поставленными задачей предупреждения перегрузок вычислительной системы.

Таблица 3.2 — Результаты второго тестирования

	Вид запроса	Исходный сервис	Сервис с алгоритмом распределения	Уменьшение величины, %
Среднее время ответа, мс	1	38.0 ± 1.1	37.1 ± 1.1	2.3
	2	34.4 ± 1.1	33.8 ± 1.0	1.8
	3	26.1 ± 0.9	25.8 ± 0.8	1.1
	4	12.7 ± 0.5	12.6 ± 0.5	0.8
	5	26.0 ± 1.7	25.3 ± 1.5	2.7
	6	20.3 ± 1.3	19.9 ± 1.2	2.0
99-й перцентиль, мс	1	110	99	10.0
	2	104	95	8.7
	3	88	83	5.7
	4	35	34	2.8
	5	115	103	10.4
	6	101	91	9.9

Стоит отметить, что полученные результаты не являются конечными. При продолжении обучения нейронной сети, лежащей в основе разработанного алгоритма, точность, а с ней и эффективность алгоритма могут быть улучшены.

ЗАКЛЮЧЕНИЕ

По завершению работы были достигнуты следующие результаты.

1. Выполнен сравнительный анализ методов прогнозирования.
2. Сформулированы рекомендации по выбору модели прогнозирования.
3. Предложен алгоритм аккумуляции обучающей выборки нейронной сети.
4. Разработан алгоритм распределения нестационарной нагрузки.
5. Выполнено сравнение исходной, и использующей разработанный алгоритм систем.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Chatfield Chris. Time-series forecasting. CRC Press, 2000.
2. Э. Е. Тихонов. Методы прогнозирования в условиях рынка: учебное пособие. Северо-Кавказский государственный технический университет, 2006.
3. Armstrong J. Scott, Brodie Roderick J. Forecasting for Marketing // Quantitative Methods in Marketing / под ред. Graham J. Hooley, Michael K. Hussey. 2 изд. International Thompson Business Press, 1999. С. 92–119.
4. Brockwell Peter J., Davis Richard A. Introductionto Time Seriesand Forecasting. 2 изд. Springer-Verlag, 2002.
5. Hyndman Rob, Athanasopoulos George. Forecasting: principles and practice. URL: — Режим доступа: https://www.otexts.org/fpp/, свободный. Дата запроса 15.02.2018.
6. Л. А. Попов. Анализ и прогнозирование временных рядов STATGRAPHICS Centurion: Учебное пособие. Российская экономическая академия, 2006.
7. Box George E. P. Time series analysis. John Wiley & Sons, 2008.
8. Kirchgassner Gebhard, Wolters Jurgen. Introduction to Modern Time Series Analysis. Springer-Verlag, 2007.
9. В. Н. Афанасьев, М. М. Юзбашев. Анализ временных рядов и прогнозирование. Финансы и статистика, 2001.
10. Fuller Wayne A. Introduction to Statistical Time Series. 2 изд. John Wiley & Sons, 1996.
11. Л. П. Рунова. Модель авторегрессии и скользящего среднего (ARMA). Издательство ЮФУ, 2013.
12. Hamilton James D. Time series analysis. Princeton University Press, 1994.
13. Станислав Осовский. Нейронные сети для обработки информации. Финансы и статистика, 2002.

14. Haykin Simon. Neural networks, a comprehensive foundation. 2 изд. Prentice Hall, 1999.

15. А. М. Чернодуб. Обучение рекуррентных нейронных сетей методом псевдорегуляризации для многошагового прогнозирования на примере хаотического процесса Маккея-Гласса // Problems of Computer Intellectualization. ITHEA, 2012. С. 141–151.

16. Medsker L. R., Jain L. C. Recurrent neural networks. CRC Press LLC, 2000.

17. Официальная документация Oracle [Электронный ресурс] // Oracle Docs: Interface OperatingSystemMXBean. URL: — Режим доступа: https://docs.oracle.com/javase/7/docs/jre/api/management/extension/com/sun/managem свободный. Дата запроса 14.02.2018.

18. Официальная документация Oracle [Электронный ресурс] // Oracle Docs: class Runtime. URL: — Режим доступа: https://docs.oracle.com/javase/7/docs/api/java/lang/Runtime.html, свободный. Дата запроса 14.02.2018.

19. Molyneaux Ian. The Art of Application Performance Testing. 2 изд. O'Reilly Media, 2015.

20. Java, Performance Companion / Charlie Hunt, Monica Beckwith, Poonam Parhar [и др.]. Pearson Education, 2016.

21. Дмитрий Александрович Кораблев. Методы проектирования эффективных экранных интерфейсов систем электронного документооборота [Текст]: автореф. дис. на соиск. учен. степ. канд. техн. наук (05.13.12) / Дмитрий Александрович Кораблев; Санкт-Петербургский государственный университет информационных технологий, механики и оптики. 2011.

22. Liu L. M. Forecasting and Time Series Analysis Using the SCA Statistical System. Scientific Computing Associates Corp., 1994. Vol. 1.

23. Mills T. C. The Foundations of Modern Time Series Analysis. Palgrave Macmillan, 2011.

24. A Comparative Study on CPU Load Predictions in a Computational Grid using Artificial Neural Network Algorithms / Shaik Naseera, G. K. Rajini, N. Amutha Prabha [и др.] // Indian Journal of Science and Technology. 2015. Т. 8, № 35.

25. И. И. Елисеева, М. М. Юзбашев. Общая теория статистики: Учебник / под ред. И. И. Елисеевой. 5 изд. Финансы и статистика, 2004. С. 656.