УДК 004.896

МУЛЬТИМОДЕЛЬНЫЙ ПОДХОД К ПОСТРОЕНИЮ СИСТЕМ ПРОАКТИВНОГО УПРАВЛЕНИЯ ВЫЧИСЛИТЕЛЬНЫМИ РЕСУРСАМИ

В.В. КРАСНОПРОШИН, А.А. СТАРОВОЙТОВ

Белорусский государственный университет, Республика Беларусь

Поступила в редакцию 05 апреля 2025

Аннотация. В работе изложены принципы построения и реализации проактивной мультимодельной системы управления критичных ИТ-сервисов в условиях неопределенности профиля внешней нагрузки и ограничений по ресурсам.

Ключевые слова: принятие решений, информационная система, проактивное управление, неопределенность внешней нагрузки, нейронные сети, мультимодельная система.

Введение

Поддержка критичных ИТ-сервисов и систем (банковских, телекоммуникационных, промышленных и др.) в рабочем состоянии (с гарантированным объемом вычислительных ресурсов) является актуальной проблемой современного цифрового общества.

Неопределенность внешней нагрузки, отказы вычислительного оборудования приводят к сбоям в работе и деградации производительности критичных ИТ-систем в целом. В результате теряется оперативность обработки информации и проведения банковских и других операций, что в свою очередь может иметь серьезные последствия (финансовые потери, крупные аварии и т.п.).

С наступлением эры облачных вычислений появилась возможность автоматически адаптировать объем вычислительных ресурсов критичных ИТ-систем под текущую нагрузку. Появились автономные решения, способные оперативно управлять масштабированием критических ИТ-сервисов без участия человека.

Автоматизация позволяет системам в реальном времени реагировать на изменение нагрузки, обеспечивая стабильное качество работы в соответствии с требуемым уровнем обслуживания. Автоматическое масштабирование напрямую влияет на бизнес-ценность ИТ-сервисов, поскольку определяет, как эксплуатационные расходы, так и качество обслуживания клиентов. Таким образом, развитие автоматизированных методов управления ресурсами является важным направлением для обеспечения надежности и экономической эффективности критически важных систем.

В общем случае масштабирование является задачей автоматизированного управления вычислительными ресурсами при изменяющейся, высокодинамичной, сложной нагрузке. Идеальный механизм автоматического масштабирования способен минимизировать как затраты, так и нарушения качества предоставления сервиса.

Несмотря на активные исследования в этой области, существует достаточно сильное отличие между методами масштабирования, предлагаемыми в научных статьях и используемыми в крупных промышленных инсталляциях [1].

Промышленные решения по автоматическому масштабированию Microsoft Azure [2], Amazon Web Services [3], Google Cloud Engine [4], Kubernetes Horizontal Pod Autoscaler (HPA) [5] и др. построены на относительно простых подходах (с возможностью заранее определять пороговые значения), предполагающих линейную зависимость между ресурсами и целевыми метриками.

Исследовательские решения [6] менее адаптированы к промышленному использованию, часто являются более сложными. Они, как правило, имеют большое число параметров настройки

и специфичные особенности, связанные с функционированием конкретного типа ИТ-системы (альтернативный набор метрик, заранее заданные пороги для метрик, специфичные профили нагрузки, характерные для данных систем и др.).

В работе предлагается вариант развития метода динамической локальной аппроксимации [7], основанного на мультимодельном подходе (DLANNLIB).

Мультимодельный метод прогнозирования

С оперативным принятием корректных решений, связаны следующие ключевые аспекты:

1. Адекватность прогноза основных параметров системы.
2. Скорость подготовки прогноза.
3. Заблаговременность прогноза.

Используемые в настоящее время подходы (позволяющие делать предсказания по большим выборкам исторических данных) могут в процессе работы терять качество прогнозирования. На начальном этапе работы они требуют подготовки обучающей выборки данных за большой период наблюдений. Используются достаточно сложные модели нейросетей (например, слои LSTM с памятью и др.), которые на данном наборе данных требуют длительного времени обучения и высокопроизводительных ресурсов.

В результате исследований стало понятно, что модель, заранее обученная на большой выборке, может иметь недостаточную обобщающую способность, и не сможет в ряде случаев давать адекватный прогноз для нового характера нагрузки. Кроме того, все критичные системы имеют совершенно разные профили нагрузки и поэтому модель, обученная на данных одной системы, может не подойти для другой. Пример исторических данных (средняя утилизация CPU по вычислительным модулям за 4 ч. работы системы) представлен на рис. 1.

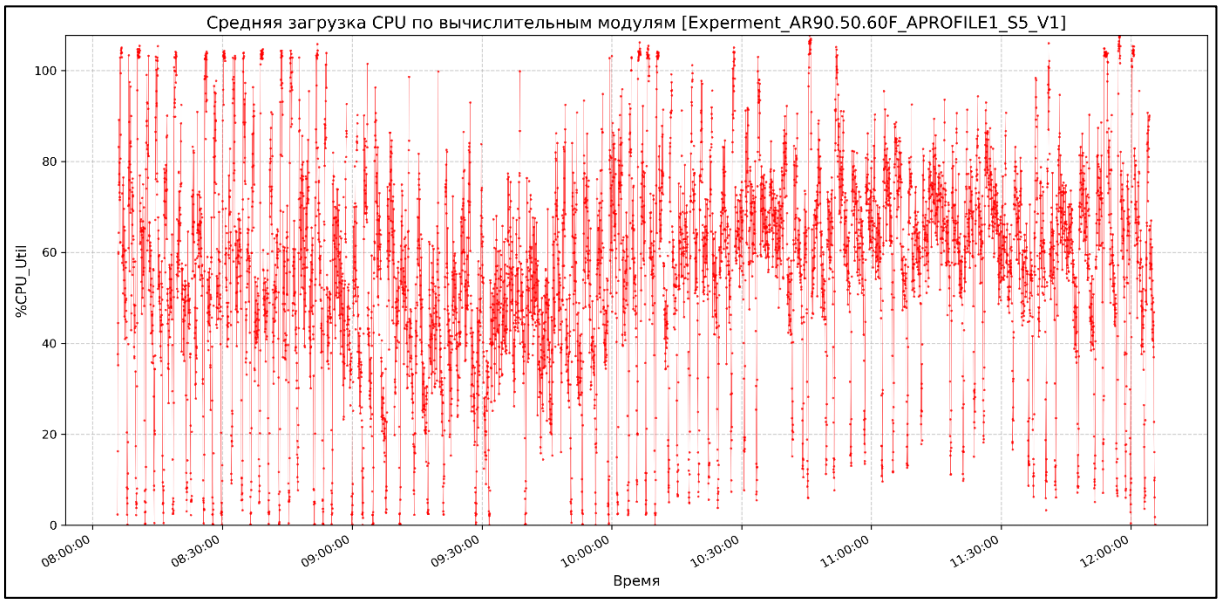


Рис. 1. Пример исторических данных работы системы
(средняя утилизация CPU по вычислительным модулям за 4 ч.)

В итоге появилась идея о том, что не так важно, что происходило с системой в течение длительного промежутка времени. Важно уметь делать адекватные прогнозы по небольшому количеству данных, накапливаемых в реальном времени. Обучение модели на этих данных и последующие предсказания должны выполняться достаточно быстро, параллельно с накоплением новых данных.

Фактически идея сводится к тому, что в процессе работы системы профиль нагрузки может быть разделен на небольшие области. И для каждой области за время ее существования может быть построена модель, описывающая ее поведение и позволяющая сделать набор прогнозов, по результатам которых система перейдет в новую область. Таким образом, будет накапливаться набор моделей, которые будут иметь лучшую точность в области своей компетенции. По

сравнению с глобальной моделью, которая обучалась на данных, подготовленных за более длительный период времени. Иллюстрация идеи представлена на рис. 2.

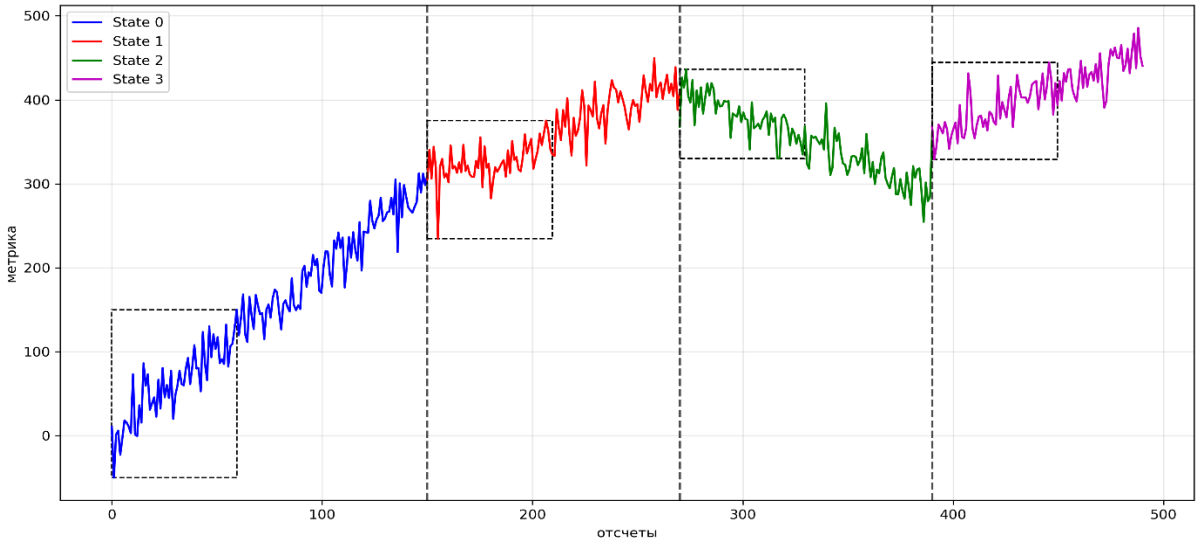


Рис. 2. Разделение профиля нагрузки на состояния и выбор данных внутри состояния для подготовки локальной модели

В рассматриваемом случае состояние критичной системы связано с объемом вычислительных ресурсов, который требуется для нормального функционирования системы для конкретной нагрузки. Состояние определяется количеством используемых вычислительных модулей. Данные для построения модели берутся из локальной временной области, соответствующей конкретному состоянию системы. В качестве метрики используется среднее значение %CPU по набору вычислительных модулей.

Локальная временная область требует отдельного рассмотрения. Время жизни конкретного состояния зависит от характера нагрузки. Переход в другое состояние определяется уровнем измеряемой метрики (уровнем перехода). Задаются максимальный уровень средней утилизации %CPU по вычислительным модулям, при котором система автоматически переходит в новое состояние с большим количеством ресурсов, и минимальный уровень, при котором система переходит в состояние с меньшим количеством ресурсов.

Локальные временные области для различных состояний могут иметь различное количество отсчетов. В это количество должны входить отсчеты, необходимые для обучения модели. На самообучение также требуется определенное время.

Построение прогноза в локальной временной области приводит к следующей задаче:

Пусть задана критическая система, которая может находиться в конечном множестве состояний $S = \{S_1, S_2, ..., S_n\}$, которые определяются воздействием на систему и уровнями переходов. В каждом состоянии S_i система генерирует дискретный сигнал в виде временного ряда $X_t^{S_i}$ (или набора рядов). Необходимо по части этого сигнала $\tilde{X}_t^{S_i}$ построить предиктор в виде нейронной сети, который прогнозирует переход системы в новое состояние S_{i+1}.

Для решения данной задачи был разработан метод динамической локальной аппроксимации нейросетевыми моделями (Dynamic Local Approximation by Neural Network models – DLANN).

Суть метода заключается в следующем: в процессе работы системы для каждого ее состояния строится простая нейросетевая модель (с одним скрытым и одним выходным слоем), которая учится на части данных локальной временной области. Предполагается, что модель, обученная на части данных, будет адекватно делать прогнозы для всей локальной области (идея восходит к принципу Парето). В процессе обучения для каждой модели сохраняется критерий качества обучения – ошибка на валидации. После обучения, на основе новых поступающих данных, формируются прогнозы среднего значения %CPU по набору вычислительных модулей

с заданной заблаговременностью. Эти прогнозы сравниваются с уровнями переходов, и при их достижении вырабатывается управляющее решение. Далее процесс повторяется.

Для упрощения параметры, которые определяют размер временной области для обучения и заблаговременность, определяются заранее и могут быть связаны со спецификой конкретной системы. Возможен автоматический подбор параметров в зависимости от характеристик сигнала. Используются нейросети с одинаковой архитектурой. Возможен вариант с автоматическим подбором архитектуры (например, изменение количества нейронов в скрытом слое или добавление скрытых слоев нейронов), в зависимости от сложности сигнала, которую можно оценивать какой-либо метрикой или их комбинацией (дисперсией, энтропией, различными размерностями и т.д.).

Допустимы ситуации, когда из-за резкой смены характера нагрузки, невозможно собрать достаточное количество данных для обучения модели и подготовить предсказания для принятия управляющего решения. В этом случае управляющее решение вырабатывается на основе текущего среднего значения %CPU по вычислительным модулям, которое сравнивается с уровнями переходов (реактивный тип).

В результате объединения работы двух предикторов получаем комбинированную систему управления, в которой реализуется комбинация реактивности и проактивности.

На основе метода DLANN был разработан мультимодельный метод, использующий библиотеку моделей (Dynamic Local Approximation by Neural Network models with Library – DLANNLIB). Метод позволяет выполнять адаптацию к новым паттернам нагрузки, связанной с неопределенностью, за счет аппроксимации данных, получаемых в режиме реального времени с помощью моделей нейронных сетей. Эти модели вместе с метаданными (метрики по качеству обучения, характеристики данных) запоминаются в процессе работы в библиотеке моделей и в дальнейшем используются для прогнозирования при переходе системы в новые состояния.

В описанном выше процессе происходит создание различных нейросетевых моделей, связанных с определенным состоянием, которое определяется количеством вычислительных модулей. В каждой модели происходит запоминание характеристик конкретного состояния системы.

Набор моделей сохраняется и накапливается в виде библиотеки. Для состояния может быть несколько моделей. Для каждой модели сохраняется метрика по качеству обучения (ошибка на валидации) и оценка сложности сигнала. Библиотеку моделей можно использовать для прогнозов до обучения новой модели в различных вариантах (указаны не все):

А. Использование модели для предыдущего состояния. Самый простой способ выбора модели. Не требуется рассчитывать никакие метрики текущего сигнала. Если для предыдущего состояния модель не была построена, то предиктор по моделям из библиотеки не используется.

Б. Выбор лучшей модели для состояния по ошибке валидации. При переходе системы в одно из известных состояний происходит выбор модели из библиотеки, для которой ошибка на валидации имеет наименьшее значение.

В. Создание ансамбля из набора моделей, которые соответствуют одному состоянию. В этом случае создается ансамбль (композиция без обучения) из прогнозов, существующих N моделей для одного состояния, с весами, связанными с ошибками валидации

$$\hat{y}_{predict} = \sum_{i=1}^{N} \varepsilon_i \hat{y}_{model_i}, \quad \varepsilon_i = \frac{E_i^{-1}}{\sum_{i=1}^{N} E_i^{-1}}. \tag{1}$$

Г. Создание ансамбля из набора моделей, которые соответствуют разным состояниям. В этом случае создается ансамбль (композиция без обучения) существующих моделей для разных состояний, с весами, связанными со сложностью сигнала.

Все эти варианты можно использовать во время обучения новой модели, в виде дополнительного предиктора, когда модель для текущего сигнала еще не готова. На рис. 3 схематически показаны области данных, которые необходимы для выполнения прогнозов с помощью библиотеки моделей и для создания новой модели для состояния.

Использование библиотеки моделей повышает оперативность прогнозирования, т.к. позволяет получить управляющее решение до готовности модели, которая обучится на данных

для конкретного состояния. Это шаг по оперативности в сторону реактивного типа, но с возможностью использования знаний, которые были накоплены системой в течение предыдущих состояний.

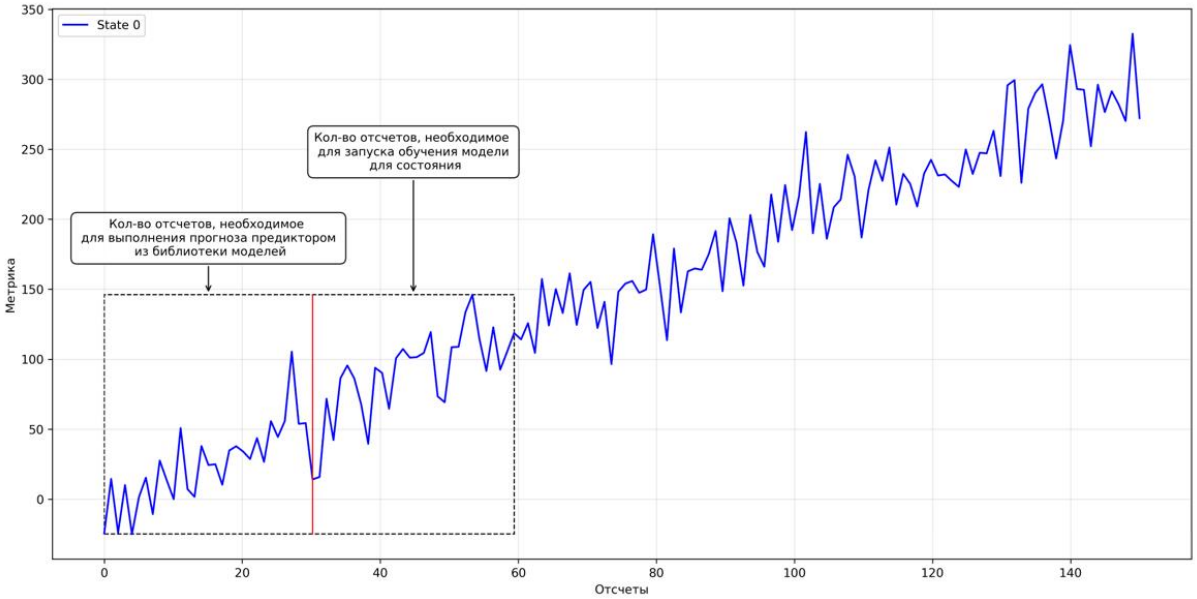


Рис. 3. Разделение профиля нагрузки на состояния и выбор данных внутри состояния для подготовки локальной модели

Создание предиктора из набора моделей можно рассматривать как самостоятельную задачу. Как было показано выше, таких вариантов может быть достаточно много. В том числе можно использовать отдельную нейросеть, которая будет производить классификацию паттернов и выбирать подходящие модели для создания предиктора. Со временем библиотека моделей увеличивается, можно предусмотреть механизм забывания, оставляя лучшие модели.

Оценка качества управления

Для оценки качества управления рассмотрим задачу, где целевая функция отражает компромисс между оптимизацией затрат на вычислительные ресурсы и выполнением требований к уровню обслуживания SLO. Переменные задачи: $\Omega(t)$ – динамически изменяющаяся внешняя нагрузка на систему с неопределенностью, $R_k(t)$ – количество вычислительных ресурсов, выделенных системе на *k*-том шаге (ресурсы системы меняются дискретно, считаем, что *k*-му шагу изменения соответствует состояние S_k), $Cost\left(R_k(t)\right)$ – стоимость использования ресурсов на *k*-том шаге, $SLO_k(t)$ – метрика уровня обслуживания на k-том шаге, $Violation\left(SLO_k(t)\right)$ – штраф за нарушение уровня обслуживания SLO. $Cost\left(R_k(t)\right)$ и $Violation\left(SLO_k(t)\right)$ нормированы в интервале [0,1] суммарно по всем состояниям S_k. Целевая функция, которую нужно минимизировать

$$\underset{A}{argmin}\left(\lambda\sum_{k=1}^{N}Cost\left(R_k\left[A\left(\Omega(t)\right)\right]\right)+(1-\lambda)\sum_{k=1}^{N}Violation\left(SLO_k\left[A\left(\Omega(t)\right)\right]\right)\right), \quad (2)$$

при ограничении на скорость изменения ресурсов $\left|R_k - R_{k-1}\right| \le \gamma$, где γ – максимальное допустимое изменение ресурсов.

В итоге задача свелась к минимизации целевой функции, которая достигается выбором оптимального алгоритма *A*, работа которого порождает *N* состояний системы, при внешней нагрузке $\Omega(t)$ с неопределенностью. Точное решение данной задачи мы построить не можем,

поэтому предлагается строить эвристический алгоритм, оценивающий целевую функцию. Из множества алгоритмов выбираем тот, который доставляет минимум целевой функции.

Нагрузочная система позволяет получить усредненную статистику по времени отклика на один запрос в интервале усреднения 5 с. Введем бинарный индикатор штрафа для системы в случае превышения среднего времени отклика за 5 с в следующем виде

$$B_i = \begin{cases} 1 & if\ R_i^5 > R_{\max} \\ 0 & if\ R_i^5 \le R_{\max} \end{cases}, \tag{3}$$

где R_i^5 – среднее время отклика за 5 с, $R_{\max}$ – заданное максимальное время отклика.

Пусть $N_{\Delta t_j}^5$ – общее число 5 с интервалов за время жизни Δt_j состояния S_j, тогда суммарное число штрафов за Δt_j, суммарное число штрафов по состояниям и нормированный штраф

$$V^{norm} = \frac{V}{N_{\max}},\quad V = \sum_{j=1}^{N} V_{\Delta t_j},\quad V_{\Delta t_j} = \sum_{j=1}^{N_{\Delta t_j}^5} B_j, \tag{4}$$

где $N_{\max}$ – общее число 5 с интервалов во всех состояниях S_j: $N_{\max} = \sum_{j=1}^{N} N_{\Delta t_j}^5$.

Оценим стоимость ресурсов состояния S_j за время жизни Δt_j как $C_j \Delta t_j$, где C_j – число вычислительных модулей состояния S_j. Суммарная стоимость ресурсов и нормированная по всем состояниям

$$C = \sum_{j=1}^{N} C_j \Delta t_j,\quad C^{norm} = \frac{C}{C_{\max} \Delta t}, \tag{5}$$

где C_{max} – максимальное число вычислительных модулей и Δt – суммарное время жизни всех состояний S_j: $\Delta t = \sum_{j=1}^{N} \Delta t_j$. В итоге, с учетом (2) – (5), получаем следующую целевую функцию для оценки качества управления

$$F = \lambda \left[\frac{\sum_{j=1}^{N} C_j \Delta t_j}{C_{\max} \sum_{j=1}^{N} \Delta t_j} \right] + (1-\lambda) \left[\frac{\sum_{j=1}^{N} \sum_{j=1}^{N_{\Delta t_j}^5} B_j}{\sum_{j=1}^{N} N_{\Delta t_j}^5} \right]. \tag{6}$$

Среднее время отклика мало информативно, используем 95 процентиль времени отклика.

Модельная система

Модельная система представляет собой набор вычислительных модулей, на которых функционируют инстансы прикладного программного обеспечения, между которыми осуществляется балансировка внешних запросов. Система поддерживает механизм масштабирования. Для генерации запросов используется нагрузочная система, позволяющая задавать нагрузку и получать статистику по обработанным запросам.

Кратко опишем основные блоки системы. В качестве вычислительных модулей используется Docker Compose [8]. Для балансировщика нагрузки по контейнерам с инстансами прикладного web сервиса используется HAProxy [9]. Web приложение реализоововано на популярном фреймворке echo.labstack [10], на языке Go [11]. Инстансы сервиса реализуют функционал в виде REST API. Для генерации нагрузки используется Locust [12], написанный на Python [13]. На рис. 4 представлена схема системы.

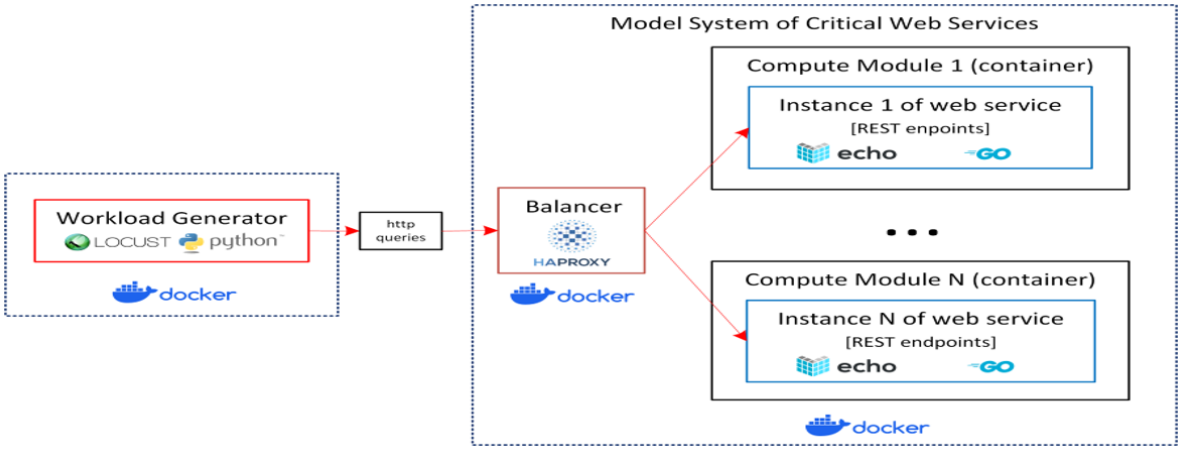
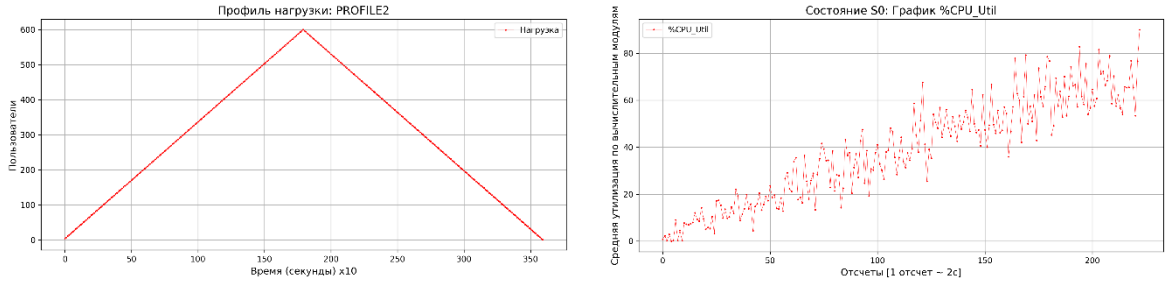


Рис. 4. Схема модельной системы

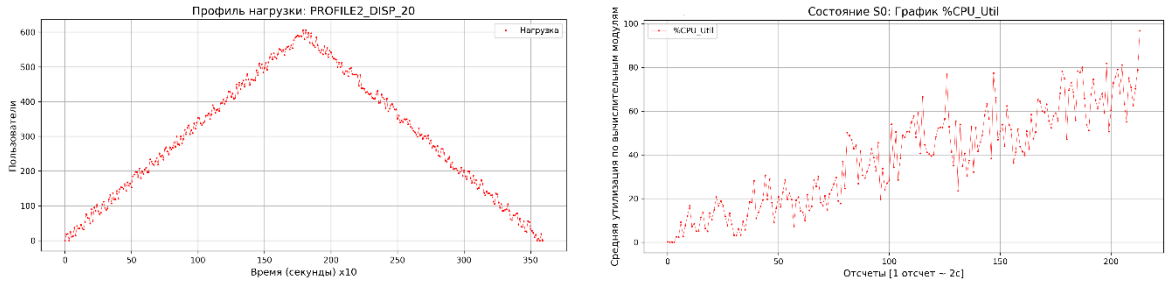
Профили нагрузки

Для каждого профиля создается свой тестовый класс на Python, который реализует механизм поэтапного управления нагрузкой. Для генерации синтетических профилей в Locust написан скрипт на Python, который на вход получает описание этапов в виде списка словарей, в котором определяются длительности этапов, целевые функции, ширина дисперсии, функция для изменения дисперсии и др. Для генерации профилей на основе реальных данных реализован скрипт на Python, который позволяет формировать структуру этапов для запуска нагрузки в Locust. На рис. 5 и 6 представлены профили пользовательской нагрузки "треугольник" и "треугольник с шумом" и графики целевой метрики для состояния S_0.



а *б*

Рис. 5. Профиль нагрузки "треугольник": *а* – изменение числа пользователей в течении 1 ч; *б* – график %CPU_Util для состояния S_0 длительностью ~ 7 мин



а *б*

Рис. 6. Профиль нагрузки "треугольник с шумом": *а* – изменение числа пользователей в течении 1 ч; *б* – график %CPU_Util для состояния S_0 длительностью ~ 7 мин

Профиль нагрузки на рис. 7, основанный на реальных данных [14], описывает число пользовательских запросов в единицу времени (RPS – Requests per Second) к различным ручкам web сервиса Космического центра НАСА им. Кеннеди во Флориде (NASA Kennedy Space Center)

за 24 часа 6 июля 1995 г. Данные представлены в общий доступ в репозитории Internet Traffic Archive на сайте Национальной лаборатории им. Лоуренса в Беркли (Lawrence Berkeley National Laboratory) [15].

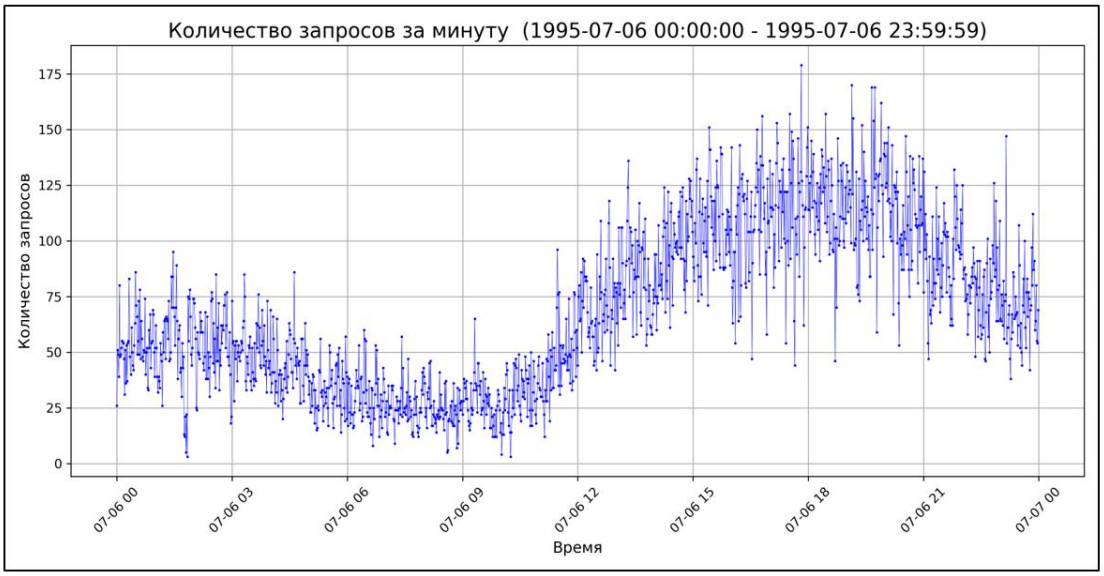
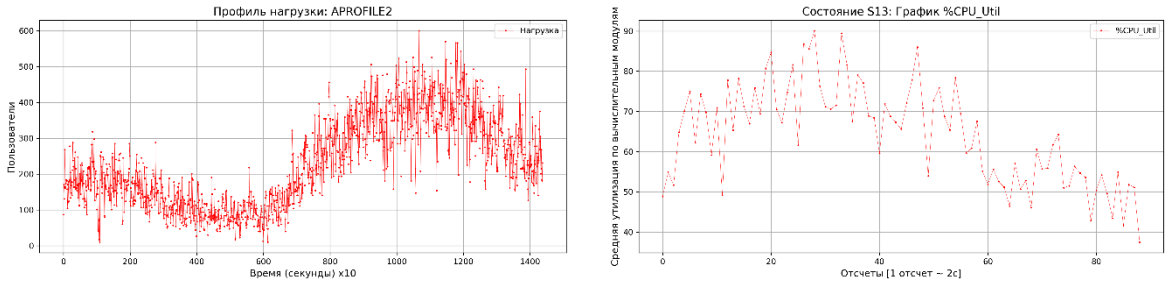


Рис. 7. Запросы пользователей к web-серверу
Космического центра Кеннеди НАСА во Флориде 6 июля 1995 г.

Исходный профиль был сжат с 24 часов до 4 часов, отмасштабирован до 600 пользователей, преобразован в нагрузочный профиль для Locust. На рис. 8 представлен получившийся профиль пользовательской нагрузки "НАСА_06.07.1995_4Н" и график целевой метрики для состояния S_{13}.



а *б*

Рис. 8. Профиль нагрузки "НАСА_06.07.1995_4Н": *а* – изменение числа пользователей в течении 1 ч; *б* – график %CPU_Util для состояния S_{13} длительностью ~ 3 мин

Комбинированная система управления

Для управления ресурсами модельной системы был разработан агент, который в реальном времени получает данные утилизации вычислительных модулей и принимает решение о масштабировании управляемой системы. Агент использует комбинацию реактивного и проактивного управления. При этом для каждого состояния управляемой системы автоматически формируется выборка данных, на которой обучается нейросетевая модель. Данная модель сохраняется в библиотеку моделей, выполняются предсказания параметров утилизации ресурсов разными предикторами (на основе обученной модели и библиотеки моделей).

Агент сравнивает текущие данные средней нагрузки по вычислительным модулям и результаты прогноза для конкретного состояния системы и принимает решение об изменении состояния. Архитектурно система содержит 6 основных модулей, которые представлены на рис. 9.

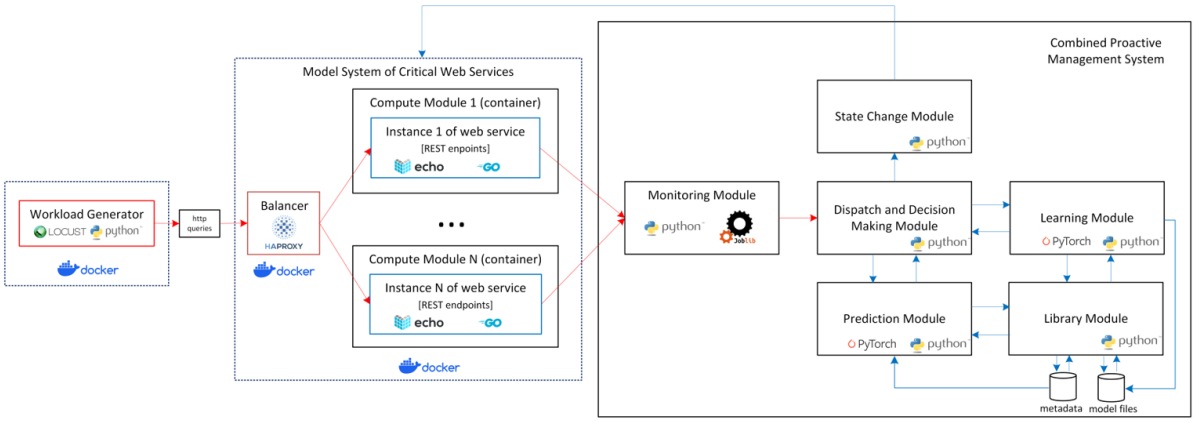


Рис. 9. Структурная схема комбинированной системы управления

Модуль мониторинга, отвечает за сбор метрик производительности вычислительных модулей системы и добавление/удаление новых метрик (при изменении состояния системы). Для сбора метрик (каждые 2 с) используется библиотека joblib [16]. Модуль изменения состояния, отвечает за отправку управляющих команд и контроль корректности изменения состояния. Модуль диспетчеризации и принятия решений является центральным. Он и остальные модули реализованы на Python.

Модули обучения и прогнозирования используют библиотеку PyTorch [17]. Модуль обучения создает модели нейросетей по наборам данных для различных состояний управляемой системы и сохраняет модели в библиотеку моделей. Каждая модель кодируется индексом состояния. Модуль прогнозирования по индексу загружает модели из библиотеки нейросетевых моделей и выполняет прогнозы утилизации с определенной заблаговременностью для различных состояний управляемой системы. Для хранения метаданных моделей используется БД SQLite [18]. Взаимодействие между процессами реализовано с помощью библиотеки multiprocessing [19]. Схема взаимодействия процессов комбинированной системы управления представлена на рис. 10.

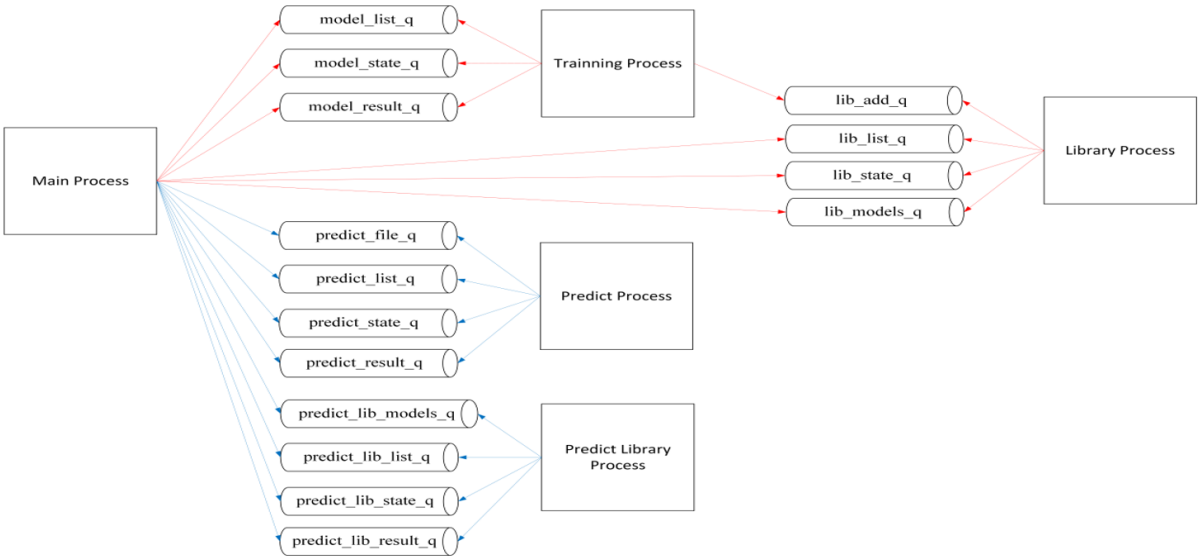


Рис. 10. Схема взаимодействия процессов комбинированной системы управления

Описание экспериментальной среды и результаты тестов

Модельная система, сервис балансировки, нагрузочная система функционируют в Docker-контейнерах на виртуальной машине под управлением ОС семейства Linux – Ubuntu 22.04.1 LTS. Система управления развернута непосредственно в ОС. Тесты производились на различном

физическом оборудовании. Финальные тесты проводились на виртуальной машине с параметрами: vCPU – 8 ядер, ОЗУ – 16 ГБ.

Качество работы комбинированной системы управления автоматическим масштабированием оцениваем с помощью ранее описанной методики. Используем метрику качества F с регулируемым весом λ для уточнения желаемого соотношения стоимости и штрафа. В качестве порогового значения для штрафующей функции по времени отклика используем 24 мс. Для расчета максимальной стоимости используем $C_{\max} = 6$. В экспериментах используются 3 профиля нагрузки, в табл. 1 представлено краткое описание.

Табл. 1. **Описание профилей для нескольких типов нагрузки**

Профиль нагрузки	Описание профиля	Время, ч
PROFILE2	Профиль "треугольник" без дисперсии. Возрастающая линейная нагрузка от 1 до 600 пользователей в течение 30 мин и убывающая линейная нагрузка от 600 до 0 пользователей в течение 30 мин	1
PROFILE2_DISP20	Профиль "треугольник" с случайным шумом. Возрастающая линейная нагрузка со случайным шумом от 1 до 600 пользователей в течение 30 мин и убывающая линейная нагрузка со случайным шумом от 600 до 0 пользователей в течение 30 мин	1
APROFILE1	Профиль HACA_06.07.1995_4H. Основан на реальной нагрузке, описывающей запросы пользователей к web-серверу Космического центра им. Кеннеди НАСА во Флориде 6 июля 1995 г. Исходный профиль был сжат с 24 часов до 4 часов и отмасштабирован до 600 пользователей (при этом исходное количество отсчетов было сохранено) и преобразован в нагрузочный профиль для Locust	4

Каждый профиль тестировался на 3-х различных алгоритмах. В названии алгоритма закодированы основные параметры работы. Расшифровка кода представлена в табл. 2. В каждой группе тестов, соответствующих определенному профилю, тест с реактивным алгоритмом рассматривается как базисный. Работа других алгоритмов сравнивается по метрикам качества с базисным алгоритмом.

Табл. 2. **Расшифровка кода алгоритмов**

Код алгоритма	Тип	Порог добавления, %CPU_Util	Порог удаления, %CPU_Util	Период стабилизации, с
AR90.50.60	Реактивный	90	50	60
AR90.50.60F	Гибридный (Реактивный + DLANN)	90	50	60
AR90.50.60F NNL_LM	Гибридный (Реактивный + DLANN + DLANNLIB самая простая реализация Last Model)	90	50	60

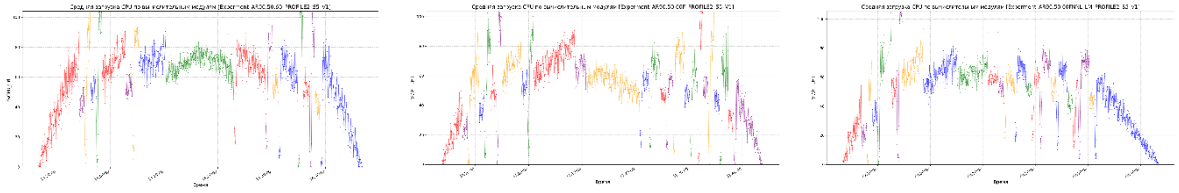
В табл. 3 представлены результаты экспериментов работы автоматизированной системы управления для 3-х различных профилей нагрузки. Метрики качества рассчитывались после проведения экспериментов, исходя из полученной статистики нагрузочной системы и логов работы системы управления.

Табл. 3. **Результаты экспериментов с 3 типами нагрузки**

Профиль нагрузки	Название алгоритма	Нормализованные значения		Целевая функция		
		C^{norm}	V^{norm}	$2F_{1/2}$	$F_{1/3}$	$F_{2/3}$
PROFILE2	AR90.50.60	0.387	0.727	1.115	0.501	0.351
	AR90.50.60F	0.434	0.515	0.949	0.461	0.367
	AR90.50.60FNNL_LM	0.463	0.467	0.930	0.465	0.377
PROFILE2_DISP20	AR90.50.60	0.382	0.793	1.176	0.519	0.350
	AR90.50.60F	0.431	0.544	0.975	0.469	0.366
	AR90.50.60FNNL_LM	0.420	0.595	1.015	0.478	0.362
APROFILE1	AR90.50.60	0.341	0.732	1.073	0.471	0.336
	AR90.50.60F	0.349	0.699	1.048	0.466	0.339

	AR90.50.60FNNL_LM	0.356	0.668	1.024	0.460	0.341

Рис. 12–14 используются в качестве иллюстрации различий работы алгоритмов для профилей нагрузки PROFILE2, PROFILE2_DISP20, APROFILE1. Цветом показаны данные для различных состояний.



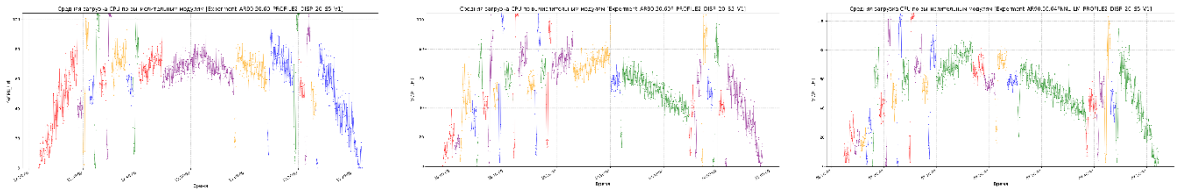
а *б* *в*

Рис. 12. Иллюстрация работы алгоритмов для профиля PROFILE2: *а* – AR90.50.60; *б* – AR90.50.60F; *в* – AR90.50.60FNNL_LM

Анализ результатов для профиля PROFILE2 показывает, что по сумме нормированных метрик стоимости и штрафов $2F_{1/2}$ самым эффективным оказался алгоритм с библиотекой моделей DLANNLIB. Он примерно на 20 % лучше реактивного и на 2 % лучше алгоритма DLANN.

Если сравнивать по отдельным метрикам, то самым экономным по ресурсам, как и ожидалось, является реактивный алгоритм, который экономнее DLANN на 12% и DLANNLIB на 20 %. По штрафам для времени отклика самым лучшим оказался DLANNLIB, который почти на 57 % лучше реактивного алгоритма и на 10 % лучше DLANN.

Для целевой функции $F_{1/3}$ с большим весом в сторону штрафов, лучшими почти с одинаковыми результатами оказались DLANN и DLANNLIB, которые примерно на 8 % эффективнее реактивного.



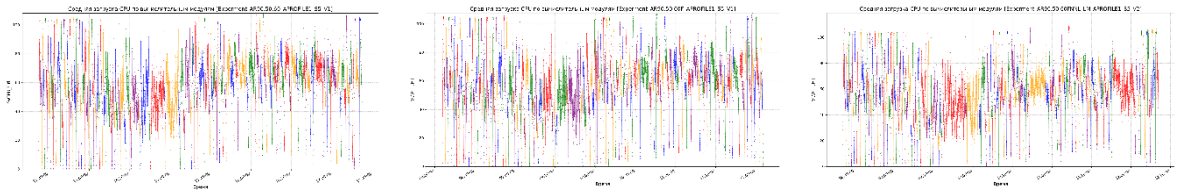
а *б* *в*

Рис. 13. Иллюстрация работы алгоритмов для профиля PROFILE2_DISP20: *а* – AR90.50.60; *б* – AR90.50.60F; *в* – AR90.50.60FNNL_LM

Анализ результатов для профиля PROFILE2_DISP20 показывает, что по целевой функции $2F_{1/2}$ самым эффективным оказался DLANN. Он примерно на 21% лучше реактивного и на 4 % лучше DLANNLIB.

Наиболее экономичным по ресурсам, является реактивный алгоритм, который оказался экономичнее DLANNLIB на 10 % и DLANN на 13 %. По штрафам по времени отклика самым лучшим оказался DLANN, который почти на 46 % лучше реактивного алгоритма и на 9% лучше DLANN.

Для целевой функции $F_{1/3}$, лучшими почти с одинаковыми результатами оказались алгоритмы DLANN и DLANNLIB, которые примерно на 10% оказались эффективнее реактивного.



а *б* *в*

Рис. 14. Иллюстрация работы алгоритмов для профиля APROFILE1: *а* – AR90.50.60; *б* – AR90.50.60F; *в* – AR90.50.60FNNL_LM

Анализ результатов для профиля APROFILE1 показывает, что по целевой функции $2F_{1/2}$ самым эффективным оказался алгоритм DLANNLIB. Он примерно на 5 % лучше реактивного и на 2 % лучше алгоритм DLANN. Наиболее экономным по ресурсам, является реактивный алгоритм, который экономнее DLANNLIB на 4 % и DLANN на 2 %. По штрафам по времени отклика самым лучшим оказался DLANNLIB, который почти на 10 % лучше реактивного алгоритма и на 5 % лучше DLANN.

Для целевой функции $F_{1/3}$ лучшими (почти с одинаковыми результатами) оказались алгоритмы DLANNLIB и DLANN, которые примерно на 2 % эффективнее реактивного.

Полученные результаты показывает, что алгоритмы, работающие хорошо на синтетических данных, серьезно снижают свою эффективность на сложно устроенном профиле, основанном на реальной нагрузке (с 20 % до 5 % по целевой функции $2F_{1/2}$). Но вместе с тем по времени отклика остаются примерно на 10 % эффективнее реактивного алгоритма.

Базовый реактивный алгоритм сохраняет свое преимущество по экономии ресурсов. При этом алгоритм DLANNLIB в самой простой конфигурации (с использованием модели для предыдущего состояния) демонстрирует лучшие показатели по большинству метрик качества на нагрузке, приближенной к реальной.

Заключение

В статье рассмотрена проблема, связанная с управлением вычислительными ресурсами критичного ИТ-сервиса в условиях неопределенности внешней нагрузки.

Предложен оригинальный мультимодельный подход (с использованием библиотеки моделей) к прогнозированию управляющего решения, повышающий адаптационные свойства и устойчивость управляемой системы. Проведены экспериментальные исследования, подтверждающие работоспособность указанного подхода и используемых технологий.

Результаты экспериментов показывают, что алгоритмы, хорошо работающие на синтетических нагрузках, серьезно снижают свою эффективность на сложно устроенном профиле, основанном на реальной нагрузке. Но при этом остаются на 10 % эффективнее реактивного алгоритма по времени отклика.

По совокупности показателей при работе с нерегулярной и сложной нагрузкой комбинированные алгоритмы оказались эффективнее реактивных.

MULTI-MODEL APPROACH TO BUILDING PROACTIVE COMPUTATIONAL RESOURCE MANAGEMENT SYSTEMS

V.V. KRASNOPROSHIN, A.A. STAROVOITOV

Abstract. The paper outlines the principles of designing and implementing a proactive multi-model management system for critical IT services under conditions of uncertain external load profiles and resource constraints.

Keywords: local decision-making, information system, proactive management, external load uncertainty, neural networks, multi-model system.

Список литературы

1. Straesser M., Grohmann J., von Kistowski J., Eismann S., Bauer A., Kounev S. Why Is It Not Solved Yet?: Challenges for Production-Ready Autoscaling // In ICPE '22: ACM/SPEC International Conference on Performance Engineering, Bejing, China, April 9 - 13, 2022, P. 105–115, ACM, 2022.
2. Microsoft Azure. Get started with autoscale in Azure [Электронный ресурс]. – Режим доступа: https://learn.microsoft.com/en-us/azure/azure-monitor/autoscale/autoscale-get-started. Дата доступа: 05.04.2025
3. Amazon Web Services. Predictive Scaling for EC2 [Электронный ресурс]. – Режим доступа: https://aws.amazon.com/en/blogs/aws/new-predictive-scaling-for-ec2-powered-by-machine-learning/. Дата доступа: 05.04.2025.
4. Google Cloud Docs. Autoscaling groups of instances [Электронный ресурс]. – Режим доступа: https://cloud.google.com/compute/docs/autoscaler/. Дата доступа: 05.04.2025.
5. The Kubernetes Authors. Horizontal Pod Autoscaler [Электронный ресурс]. – Режим доступа: https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale/. Дата доступа: 05.04.2025.
6. Singh P., Gupta P., Jyoti K., Nayyar A. Research on auto-scaling of web applications in cloud: survey, trends and future directions // Scalable Computing: Practice and Experience. 2019. Vol. 20, № 2. P. 399–432.
7. Starovoytov A. A., Krasnoproshin V. V. Technology for making real-time decisions based on neural network forecasting // Pattern Recognition and Information Processing (PRIP'2023): Proceedings of the 16th International Conference, October 17–19, 2023, Minsk, Belarus. Minsk, 2023. P. 58–63.
8. Docker Compose Overview [Электронный ресурс]. – Режим доступа: https://docs.docker.com/compose/. Дата доступа: 05.04.2025.
9. HAProxy [Электронный ресурс]. – Режим доступа: https://www.haproxy.org/. Дата доступа: 05.04.2025.
10. Echo LabStack High performance extensible minimalist Go web framework [Электронный ресурс]. – Режим доступа: https://echo.labstack.com/. Дата доступа: 05.04.2025.
11. Go an open-source programming language supported by Google [Электронный ресурс]. – Режим доступа: https://go.dev/. Дата доступа: 05.04.2025.
12. Locust [Электронный ресурс]. – Режим доступа: https://locust.io/. Дата доступа: 05.04.2025.
13. Python [Электронный ресурс]. – Режим доступа: https://www.python.org/. Дата доступа: 05.04.2025.
14. Aslanpour M. S., Ghobaei-Arani M., Toosi A. Auto-scaling Web Applications in Clouds: A Cost-Aware Approach // Journal of Network and Computer Applications. 2017. Vol. 95. DOI: https://doi.org/10.1016/j.jnca.2017.07.012. Дата доступа: 05.04.2025.
15. Lawrence Berkeley National Laboratory. Internet Traffic Archive. NASA-HTTP [Электронный ресурс]. – Режим доступа: https://ita.ee.lbl.gov/html/contrib/NASA-HTTP.html/. Дата доступа: 05.04.2025.
16. Joblib Python library [Электронный ресурс]. – Режим доступа: https://joblib.readthedocs.io/en/latest/parallel.html/. Дата доступа: 05.04.2025.
17. Pytorch [Электронный ресурс]. – Режим доступа: https://pytorch.org/. Дата доступа: 05.04.2025.
18. SQLite [Электронный ресурс]. – Режим доступа: https://www.sqlite.org/. Дата доступа: 05.04.2025.
19. Multiprocessing Python library [Электронный ресурс]. – Режим доступа: https://docs.python.org/3/library/multiprocessing.html/. Дата доступа: 05.04.2025.