DOI 10.37882/2223-2966.2024.01.31

РАЗРАБОТКА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ ДЛЯ СБОРА И АНАЛИЗА ДАННЫХ О ДОСТУПНОСТИ ПРОКСИ СЕРВЕРОВ С ИСПОЛЬЗОВАНИЕМ МЕТОДОВ МАШИННОГО ОБУЧЕНИЯ

DEVELOPMENT OF SOFTWARE FOR COLLECTING AND ANALYZING DATA ON THE AVAILABILITY OF PROXY SERVERS USING MACHINE LEARNING METHODS

A. Rusakov
M. Deryugin
V. Nikonov
M. Simonova

Summary. The article describes the process of creating software for collecting and analyzing data on the availability of proxy servers using machine learning methods. Among the ready-made solutions, there were no options that could fully meet the needs for improving the quality of services provided, so it was decided to start developing its own program that allows, without the active participation of the user, to check and collect information about the availability of proxy servers from public resources, analyze their performance and analyze their availability in the near future. By sending a request to a public web resource through a proxy, it was possible to check the performance based on the response or on the delay of the request. Some machine learning algorithms have been tested for forecasting, as well as exponential smoothing.

Keywords: proxy server, proxy server availability, information security, machine learning.

Русаков Алексей Михайлович
Старший преподаватель, МИРЭА — Российский технологический университет
rusal@bk.ru
Дерюгин Михаил Геннадьевич
Аспирант, МИРЭА — Российский технологический университет
UStStation@yandex.ru
Никонов Вячеслав Викторович
Доцент, МИРЭА — Российский технологический университет
nikonov_v@mirea.ru
Симонова Мария Алексеевна
МИРЭА — Российский технологический университет,
simonova_maria_al@mail.ru

Аннотация. В статье приводится процесс создания программного обеспечения для сбора и анализа данных о доступности прокси серверов с использованием методов машинного обучения. Среди готовых решений не было вариантов, которые могли бы в полной мере удовлетворить потребности в повышении качества предоставляемых услуг, поэтому было решено приступить к разработке собственной программы, позволяющей без активного участия пользователя проверять и собирать информацию о доступности прокси-серверах из публичных ресурсов, анализировать их работоспособность и анализировать их доступность в ближайшем будущем. Используя отправку запроса на публичных веб-ресурс через прокси, можно было проверить работоспособность, опираясь на ответ или на задержку запроса. Для прогнозирования были проверены некоторые алгоритмы машинного обучения, а также задействовано экспоненциальное сглаживание.

Ключевые слова: прокси сервер, доступность прокси сервера, информационная безопасность, машинное обучение.

Благодаря развитию технологий современный человек уже слабо представляет свою жизнь без интернета. Интернет проник во все сферы жизнедеятельности и существенно упрощает процесс взаимодействия между людьми, он позволяет в один клик совершать покупки и буквально управлять своим домом. Благодаря такому широкому охвату многие технологические компании собирают информацию о действиях пользователей в интернете, что с одной стороны позволяет улучшать качество предоставляемых услуг, но с другой не всем может понравиться своего рода «слежка» за действиями в глобальной сети. Одним из основных идентификаторов пользователя интернета является его IP-адрес, который позволяет определить местоположение конечного устройства с высокой точностью. Благодаря этой особенности, возможно, например. Ограничить доступ к какому-либо ресурсу в сети, который был ограничен по каким-либо причинам, что негативно сказывается на конечном пользователе.

Необходимость прокси-сервера заключается в том, что с помощью него возможно подключение к интернет-ресурсу через посредника. Благодаря этому появляется возможность подменить выходной IP адрес клиента или ресурса сети. Это может быть полезно при наложении ограничений со стороны политик компании, например, либо системного администратора или каких-либо других регуляторов. В сети интернет есть множество ресурсов, которые позволяют получить список публичных прокси-серверов, которые могут быть использованы всеми желающими. Но не всегда удаётся получить приемлемое качество услуг от таких прокси-серверов, так как пода-

вляющее большинство этих серверов не обслуживаются или имеют слишком высокое время отклика на запросы. И даже в том случае, если прокси-сервер удовлетворяет требованиям пользователя на данный момент, никто не может гарантировать, что качество предоставляемых услуг прокси-серверов не будет ухудшаться. Такой вариант развития событий возможен, например, из-за непреднамеренного предоставления публичности прокси-серверу владельцем, из-за чего сервер стал доступен из сети интернета. Поэтому актуальным является автоматический подбор быстрых и безопасных прокси-серверов. Существуют следующие способы выхода в Интернет, используя прокси-сервер, рассмотрим достоинства и недостатки каждого:

1. публичные прокси-сервера:
— анонимность пользователя, не требуется раскрытие личной информации;
— широкий выбор источников из списка доступных;
— часть этих источников имеет низкий уровень качества предоставляемых услуг;
2. собственный прокси-сервер:
— предоставление собственных прав доступа;
— необходимость администрирования ресурса;
— трудности с расширением, так как для каждого нового прокси-сервера необходима аренда ресурсов (IP адрес);
3. использование специализированных ресурсов:
— лёгкость в масштабируемости и гибкость решений;
— необходим надёжный поставщик услуг;
— деанонимизация пользователя услугами.

В рамках этой работы мы разработаем программу, включающую в себя преимущества каждого из способов выхода в интернет с помощью прокси-сервера. Способ реализации включает в себя анализ и последующий сбор данных о публичных прокси-серверах с дальнейшим исследованием качества предоставляемых услуг для выхода в глобальную сеть и выбором наилучших по результатам исследования.

Модель функционирования системы сбора и анализа данных о доступности прокси серверов

В качестве источников прокси-серверов будут использоваться ресурсы, постоянно формирующие списки публичных прокси-серверов. Разрабатываемая программа периодически будет проверять списки с веб-ресурсов и записывать в базу информацию о прокси-серверах. Далее, программа постоянно анализирует каждый прокси-сервер на доступность и пишет в базу данных следующие данные:

— время отклика;
— HTTP статус код;
— адрес клиента, который идентифицировал целевой ресурс (внешний адрес прокси-сервера);

Программа состоит из следующих компонентов:

— Компонент сбора — отвечает за сбор информации из публичных источников;
— Компонент проверки — ответственен за анализ доступности прокси-серверов;
— База данных, которая ответственна за хранение данных и синхронизацию работы компонентов;
— Менеджер задач — отвечает за отправку задач в очередь сообщений;
— API — организует последовательный доступ к данным.

Программа функционирует по следующему алгоритму, представленному на рисунке 1:

1. Компонент сбора по расписанию агрегирует данные о прокси-серверах с публичных ресурсов, которые предоставляют данную информацию;

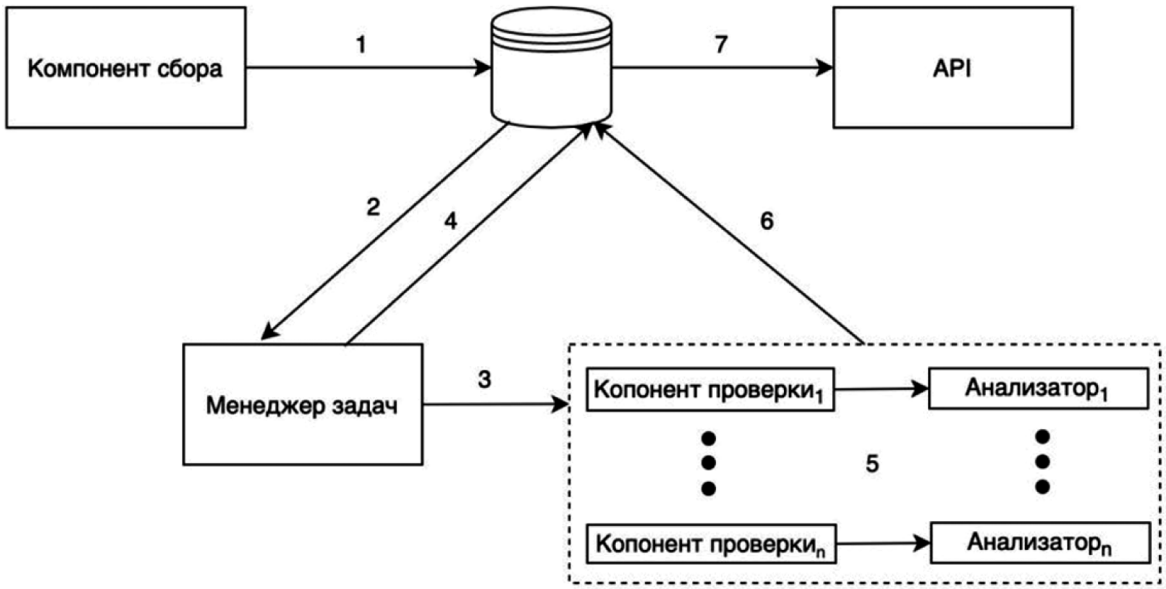

Рис. 1. Архитектура системы сбора и анализа данных о доступности прокси серверов

2. Менеджер задач берёт из базы данных те прокси-сервера, работоспособность которых должна быть протестирована;
3. Менеджер задач посылает задачу на тестирование сформировавшегося списка прокси-серверов;
4. Менеджер задач отмечает в базе данных время, когда конкретный прокси-сервер отправился тестирование;
5. Компонент проверки тестирует каждый полученный прокси-сервер и пересылает полученные данные анализатору;
6. Анализатор составляет прогноз доступности прокси-сервера на ближайшее будущее, после чего записывает результат предсказания в базу данных;
7. API отвечает за предоставление доступа к информации о прокси-серверах, для которых уже готов прогноз доступности.

По результатам анализа доступности прокси-сервера имеют возможность попасть в одну из трех групп:

1. Группа для тестирования — в этой группе прокси-сервера имеют среднее время проверки работоспособности. Данная группа создана для прокси-серверов, оценка производительности которых до сих пор не завершена.
2. Группа активных серверов — проверки для этой группы осуществляются наиболее часто. Те прокси-сервера, что попали в эту группу маркируются как доступные и готовы к отправке конечному пользователю.
3. Группа неактивных серверов — в этой группе прокси-сервера больше на анализируются на работоспособность. Если сервер попал в данную группу, то считается, что он стабильно не рабочий.

Распределение по группам предназначено для более эффективного использования вычислительных ресурсов, сделано это потому, что анализ доступности представляет из себя довольно затратную по времени операцию. Таким образом, те сервера, которые помечены как стабильно нерабочие исключаются из проверки, а наиболее жизнеспособные прокси-сервера проверяются чаще. Ниже, на изображении 2, приведена схема жизненного цикла прокси-сервера в программе:

Количественный анализ данных о доступности прокси серверов

Исследование было проведено на 670000 прокси-серверах из 10 разных публичных источников. Каждые

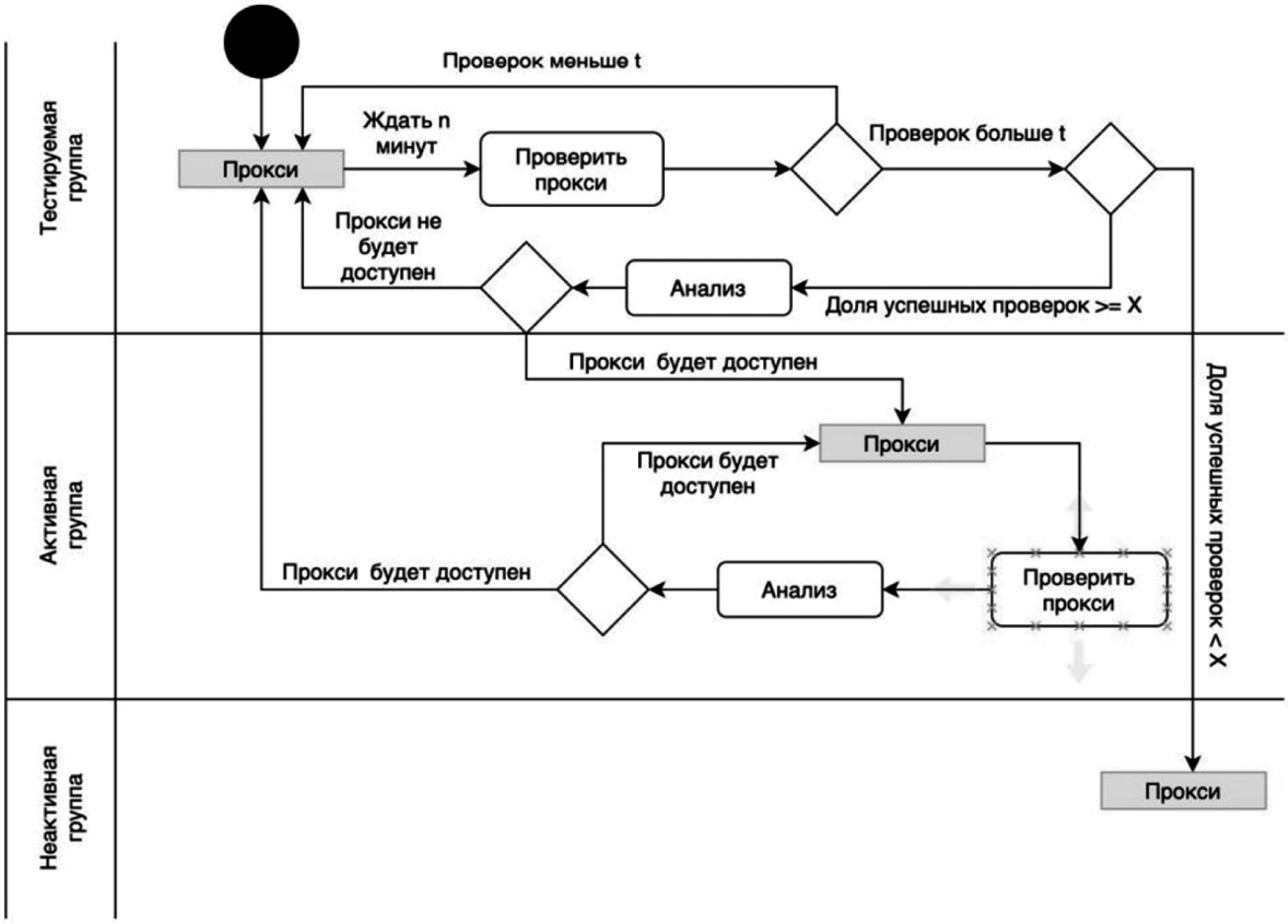

Рис. 2. Жизненный цикл прокси-сервера

2 часа для выбранных прокси-серверов осуществлялся анализ доступности. Прокси-сервер помечался как недоступный, если время ответа было больше 20 секунд, или если HTTP статус-код возвращался отличный от успешного. Одной из основных задач этой работы является возможность предсказывания работоспособности прокси-серверов. Необходимо отобрать такие признаки, которые за наименьшее количество проверок могут спрогнозировать работоспособность сервера в ближайшем времени. Предположим, следующую теорию: прокси-сервера с высоким уровнем работоспособности могут время от времени быть недоступными или отвечать с высокими задержками (больше 20 секунд), однако они не могут проваливать несколько проверок подряд.

Разработка модели оценки доступности

Для определения работоспособности работы алгоритмов машинного обучения и для анализа его применимости в нашей работе, был составлен базовый метод предсказания. Суть работы метода: берется 10 последних проверок и высчитывается число **S**, которое равняется доли успешных среди этих проверок. Чтобы проанализировать, пройдет ли прокси-сервер проверку, которая следует за этими 10 проверками, берется некоторое число **Bound** от 0 до 1 и если **S ≥ Bound**, то мы делаем вывод, что прокси-сервер пройдет проверку работоспособности. В качестве числа **Bound** было взято 11 чисел от 0 до 1.0 с шагом 0.1, и для каждого была произведена классификация описанным ранее способом. Опираясь на результаты данной классификации, был сформирован следующий график, изображённый на рисунке 3:

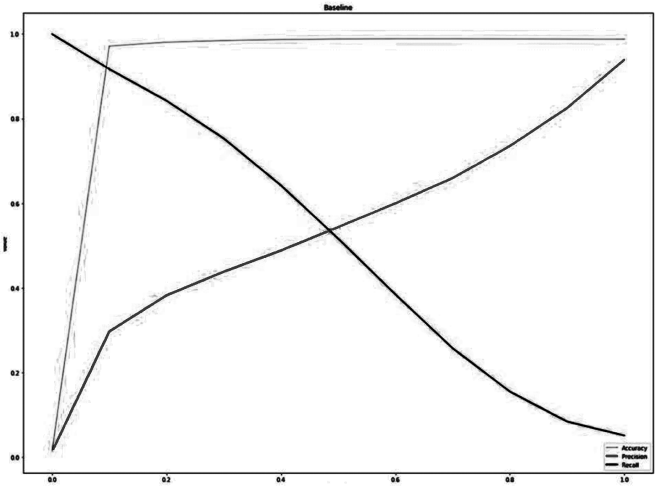

Рис. 3. Распределение количества проверок. Горизонтальная ось — bound. Вертикальная ось — значение

Результаты классификации интерпретируются следующим образом:

1. TP — количество истинно-положительно классифицированных объектов;
2. TN — количество истинно-отрицательно классифицированных объектов;
3. FP — количество ложно-положительно классифицированных объектов;
4. FN — количество ложно-отрицательно классифицированных объектов.

Метрики качества классификации:

$$precision = \frac{TP}{TP + FP}$$

$$recall = \frac{TP}{TP + FN}$$

$$accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

Перед переходом к этапу обучения моделей было разработано несколько наборов описания объектов по признаку:

1. **last_10_bool** — набор состоит из 10 признаков, которые составлены из 10 последних проверок. Значение признака равно 0 если сервер не прошел проверку и 1 если прошел.
2. **first_10_bool** — набор состоит из 10 признаков, которые составлены из 10 первых проверок. Значение признака равно 0 если сервер не прошел проверку и 1 если прошел.
3. **last_20_bool** — набор состоит из 20 признаков, которые составлены из 20 последних проверок. Значение признака равно 0 если сервер не прошел проверку и 1 если прошел.
4. **last_10_time** — набор состоит из 20 признаков, которые составлены из 10 последних проверок. Для первых 10 признаков значение признака равно 0 если сервер не прошел проверку и 1 если прошел, а остальные 10 признаков определяются как время (в секундах), прошедшее с момента каждой из 10 последних проверок до проверки, которую мы пытаемся предсказать.
5. **last_10_succ_rate**— набор состоит из 11 признаков, которые составлены из 10 последних проверок. Для первых 10 признаков значение признака равно 0 если сервер не прошел проверку и 1 если прошел, а 11-ый признак равен доли успешных проверок этого сервера за все время.
6. **last_10_fiar** — набор состоит из 11 признаков, которые составлены из 10 последних проверок. Для первых 10 признаков значение признака равно 0 если сервер не прошел проверку и 1 если прошел, а 11-ый признак равен максимальному количеству непройденных подряд проверок.
7. **last_10_fiar_sr** — набор состоит из 12 признаков, которые составлены из 10 последних проверок. Для первых 10 признаков значение признака равно 0 если сервер не прошел проверку и 1 если прошел, 11-ый признак равен доли успешных про-

верок этого сервера за все время, а 12-ый признак равен максимальному количеству непройденных подряд проверок.

8. **last_10_fiar_sr_good** — составлен по тому же принципу, что набор 7, но в данном наборе присутствуют только те сервера, у которых среди первых 10 проверок доля успешных больше или равна 0.1. Всего таких серверов 29190.

Целевой переменной для всех наборов признаков будет служить результат проверки, которая следует за теми, которые использовались для образования конкретного набора.

Настройка гиперпараметров моделей

Большинство моделей машинного обучения имеют всевозможные гиперпараметры. Гиперпараметры — это такие параметры модели машинного обучения, которые задаются до обучения модели и остаются неизменными в процессе обучения. В зависимости от данных, одни и те же модели с разными наборами гиперпараметров скорее всего получат различные результаты классификации/регрессии.

Существует множество методов оптимизации гиперпараметров, но в этой работе были применены два следующих:

1. Поиск по решетке. Данный метод совершает полный перебор по заданному вручную множеству гиперпараметров. Для каждого набора гиперпараметров проводится оценка модели методом скользящего контроля [1] (кросс-валидация). Результатом работы метода является набор гиперпараметров, который показал самый лучший результат.
2. Случайный поиск. Данный метод схож с методом поиска по решетке, однако вместо полного перебора заданного множества, наборы гиперпараметров составляются случайным образом. Количество тестируемых наборов задается вручную. Данный метод обычно используется для моделей с большим количеством гиперпараметров.

В таблице 1 показаны значения всех гиперпараметров для каждой модели.

Из таблицы видно, что случайный поиск необходим только для модели RandomForest. Происходит это из-за того, что полный перебор значений всех гиперпараметров этой модели занял бы огромное количество времени. Также, в таблице не представлен наивный байесовский классификатор, так как он не имеет настраиваемых гиперпараметров.

Результатом работы алгоритмов оптимизации гиперпараметров стали следующие значения, которые приведены в таблице 2.

Результаты работы классификаторов

В качестве алгоритмов машинного обучения были выбраны следующие модели:

1. SVM (linear kernel) — метод опорных векторов с линейным ядром;
2. Random Forest — случайный лес;
3. SVM (rbf kernel) — метод опорных векторов с радиально-базисным ядром;
4. Naive Bayes — наивный байесовский классификатор;
5. Logistic Regression — логистическая регрессия.

Таблица 1.

Описание гиперпараметров моделей и множество тестируемых значений

Модель	Метод	Гиперпараметры	Значения	Описание
SVC(linear)	Поиск по решетке	C	0.001, 0.01, 0.1, 1, 10	Отвечает за ширину разделяющей полосы
RandomForest	Случайный поиск	n_estimators	100, 200, 400, 600, 800, 1000, 1200, 1400, 1600, 1800, 2000	Количество деревьев
		min_samples_split	2, 5, 10, 20, 50	Минимальное количество объектов для ветвления узла
		min_samples_leaf	1, 2, 4, 8, 16, 32	Минимальное количество объектов в листе
		max_features	auto, sqrt	Количество признаков, которое учитывается при ветвлении
		max_depth	10, 20, 30, 40, 50, 60, 70, 80, 90, 100	Максимальная глубина дерева
		bootstrap	True, False	Использование бэггинга
SVC(rbf)	Поиск по решетке	C	0.001, 0.01, 0.1, 1, 10	Отвечает за ширину разделяющей полосы
		gamma	0.001, 0.01, 0.1, 1, 10	Параметр радиально-базисной функции
LogisticRegression	Поиск по решетке	C	0.001, 0.01, 0.1, 1, 10	Отвечает за степень регуляризации
		penalty	l1, l2, elasticnet	Вид регуляризатора

Таблица 2.

Описание гиперпараметров моделей и множество тестируемых значений

Модель	Метод	Гиперпараметры	Значения
SVC(linear)	Поиск по решетке	C	10
RandomForest	Случайный поиск	n_estimators	1200
		min_samples_split	2
		min_samples_leaf	8
		max_features	sqrt
		max_depth	60
		bootstrap	False
SVC(rbf)	Поиск по решетке	C	10
		gamma	1
LogisticRegression	Поиск по решетке	C	1
		penalty	l1

Для получения результатов работы классификаторов использовалась схема 5-fold cross validation:

1. Вся выборка случайно делится на 5 равных по количеству объектов частей;
2. В качестве тестовой выборки берется первая из 5 частей, а в качестве обучающей все остальные;
3. Модель обучается, тестируется и вычисляются метрики качества;
4. Далее, процедура повторятся еще 4 раза так, чтобы в итоге каждая из пяти частей побывала в качестве тестовой выборки;
5. В качестве результата вычисляются усредненные метрики качества.

В таблице 3 приведены значения ключевых метрик моделей машинного обучения. Здесь также присутствует метрика F1, которая вычисляется по следующей формуле:

$$F1 = 2 * \frac{precision * recall}{precision + recall}$$

Таблица 3.

Метрики моделей машинного обучения

Модель	Набор признаков	Accuracy	Precision	Recall	F1
SVC (linear)	last_10_bool	0.988	0.662	0.306	0.419
RandomForestClassifier		0.988	0.594	0.342	0.434
SVC (rbf)		0.988	0.594	0.347	0.438
Naive Bayes		0.971	0.309	0.914	0.461
LogisticRegression		0.989	0.633	0.385	0.479
SVC (linear)	fist_10_bool	0.988	0.628	0.324	0.428
RandomForestClassifier		0.988	0.594	0.342	0.434
SVC (rbf)		0.988	0.588	0.366	0.451
GaussianNB		0.968	0.289	0.927	0.441
LogisticRegression		0.988	0.610	0.387	0.473
SVC (linear)	last_20_bool	0.989	0.643	0.255	0.365
RandomForestClassifier		0.989	0.617	0.339	0.437
SVC (rbf)		0.988	0.508	0.358	0.420
GaussianNB		0.964	0.248	0.945	0.393
LogisticRegression		0.989	0.598	0.403	0.482
SVC (linear)	last_10_ with_time	0.989	0.640	0.302	0.410
RandomForestClassifier		0.989	0.652	0.360	0.464
SVC (rbf)		0.988	0.557	0.340	0.422
GaussianNB		0.972	0.302	0.920	0.455
LogisticRegression		0.989	0.607	0.383	0.470
SVC (linear)	last_10_ succ_rate	0.990	0.653	0.364	0.468
RandomForestClassifier		0.990	0.631	0.418	0.503
SVC (rbf)		0.989	0.595	0.381	0.464
GaussianNB		0.971	0.291	0.915	0.441
LogisticRegression		0.990	0.629	0.434	0.514
SVC (linear)	last_10_ with_fiar	0.989	0.632	0.297	0.404
RandomForestClassifier		0.989	0.617	0.390	0.478
SVC (rbf)		0.988	0.524	0.397	0.452
GaussianNB		0.972	0.294	0.915	0.445
LogisticRegression		0.989	0.585	0.404	0.478
SVC (linear)	last_10_sr_ fiar	0.990	0.651	0.379	0.479
RandomForestClassifier		0.990	0.659	0.453	0.537
SVC (rbf)		0.988	0.529	0.404	0.458
GaussianNB		0.971	0.290	0.915	0.440
LogisticRegression		0.989	0.597	0.412	0.487
SVC (linear)	last_10_sr_ fiar_good	0.805	0.655	0.424	0.515
RandomForestClassifier		0.810	0.658	0.458	0.540
SVC (rbf)		0.778	0.553	0.456	0.500
GaussianNB		0.750	0.491	0.743	0.591
LogisticRegression		0.800	0.629	0.431	0.512

Если ориентироваться на метрику F1, то лучшим набором признаков получится набор под номером 8

(last_10_sr_fiar_good). Наиболее вероятно это связано с тем, что остальные наборы содержат в себе слишком большое количество серверов, без единой успешной пройденной проверки, из-за чего 95 % результатов являются почти одинаковыми. Можно также заметить, что в отношении точности результатов, лучшей моделью среди всех оказался метод опорных векторов с линейным ядром. Однако, логистическая регрессия во всех экспериментах имеет почти такие же результаты, как и SVM, но она всегда опережает конкурентов по полноте, из-за чего метрика F1 у логистической регрессии оказывается в среднем чуть больше, чем у SVM. К тому же процесс обучения логистической регрессии происходит гораздо быстрее, чем у метода опорных векторов. Стоит так сказать, что хоть наивный байесовский классификатор имеет наибольшее значение F1 в последнем эксперименте, но он также значительно уступает по точности в сравнении с другими представленными моделями, что делает его выбор в качестве основного метода оценки качества прокси-серверов не валидным, так как в рамках решения поставленной задачи точность играет гораздо большую роль, чем полнота. Таким образом, по результатам проведенных проверок и исследований, моделью с наилучшим результатом является логистическая регрессия.

Линейная регрессия

Так как все проверки включают в себя значение времени отклика, то можно попытаться угадать это значение для следующей проверки вместо того, чтобы просто предсказывать доступность прокси-сервера. Так как время отклика — это числовое значение, а не категориальное, то для его предсказания не подходят методы классификации.

Для обучения регрессии был составлен специальный набор признаков. Этот набор состоит из 20 признаков. Первые 10 признаков представляют из себя значения времени отклика последних 10 проверок. Если программа не получила ответ от прокси-сервера во время проверки, то она считает, что время отклика равно 21000, искусственно завышая максимальное время отклика, равное 20000. Остальные 10 признаков определяются как время (в секундах), которое прошло с момента каждой из последних 10 проверок до той проверки, которую мы пытается предсказать.

Результатом линейной регрессии будет являться число L, которое будет равно предполагаемому времени отклика сервера, который мы пытаемся предсказать. Для сравнения регрессии с остальными моделями, как и в случае с базовым методом, мы можем взять некое число **Bound**, и если **L < Bound**, то мы считаем, что сервер пройдет проверку, а если **L ≥ Bound**, то будем считать, что проверка не пройдена. Для визуализации результатов работы регрессии был сформирован следующий график, изображённый на рисунке 4:

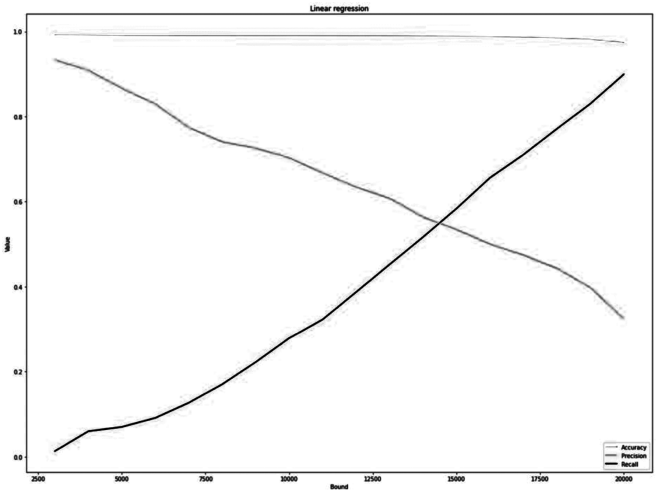

Рис. 4. Результаты линейной регрессии. Горизонтальная ось — Bound. Вертикальная ось — значение

Экспоненциальное сглаживание

Методы машинного обучения моделируют результат для одного прокси-сервера опираясь на данные, полученные от всех прокси-серверов, использованных во время обучения. Иной подход использует метод экспоненциального сглаживания, который во время предсказания оперирует данными только того сервера, поведение которого он пытается предсказать. Второй метод выглядит привлекательнее в случае, когда индивидуальные особенности конкретного прокси-сервера значат больше, чем усредненные значения признаков множества серверов.

На графике ниже, изображённом на рисунке 5, можно заметить, что синяя линия обрывается чуть раньше, чем другие. Связано это с тем, что синяя линия показывает текущие данные о времени отклика, полученные во время анализа. А последние значения остальных линий отображают прогнозируемые значения времени отклика для следующей проверки.

Для составления прогнозов значений будем использовать все имеющиеся проверки прокси-сервера кроме самой последней, так как ее мы будем предсказывать. Для того, чтобы данный метод можно было сравнить с другими, будем считать, что если прогнозируемое значение больше или равно 20000, то сервер не пройдет проверку. После чего производится сравнения результата с реальным и делается вывод о правильности составленного предсказания.

Так как данный метод имеет 2 настраиваемых параметра, то были взяты некоторые значений обоих параметров, составлены всевозможные пары этих значений и с каждой парой были проведены манипуляции, опи-

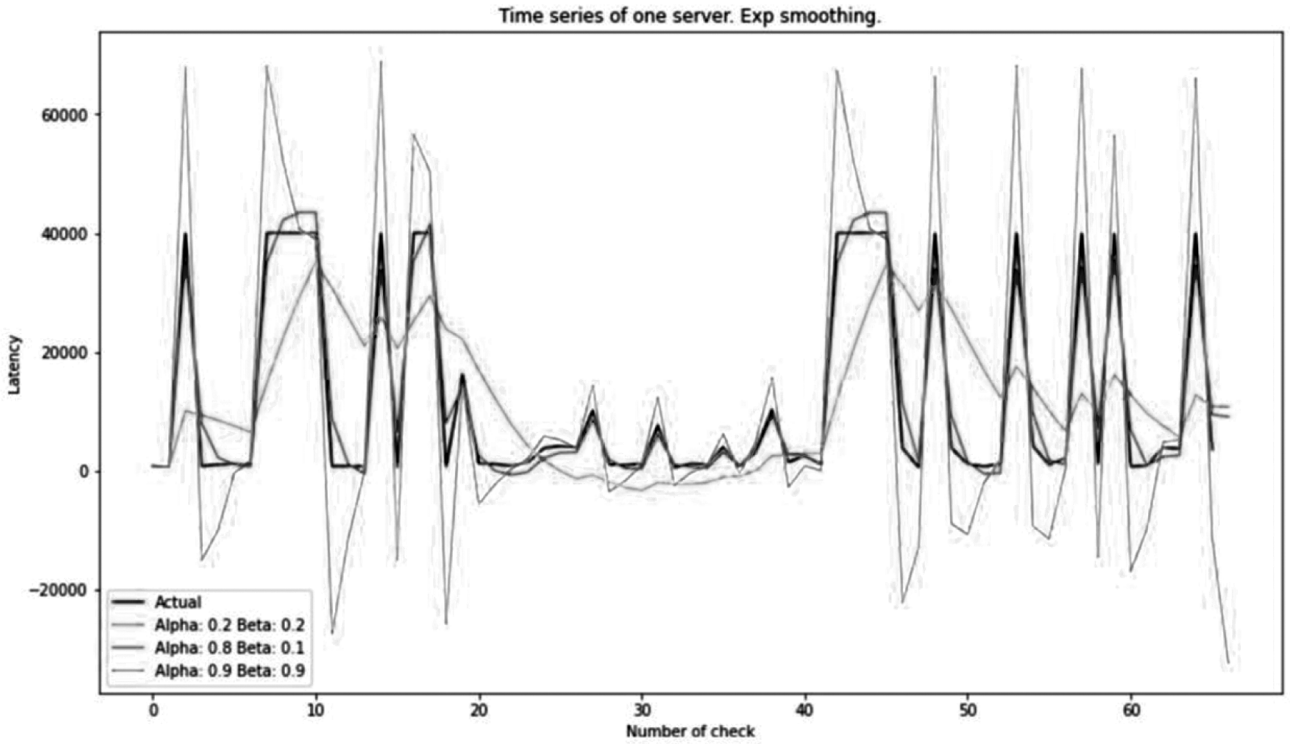

Рис. 5. Двойное экспоненциальное сглаживание с разными параметрами Горизонтальная ось — номер проверки. Вертикальная ось — время отклика.

санные выше. В результате была выбрана та пара, которая показала наилучшие результаты по метрике F1.

В таблице 4 приведены значения ключевых метрик метода двойного экспоненциального сглаживания.

Таблица 4.

Результаты прогнозирования двойного экспоненциального сглаживания

Модель	Accuracy	Precision	Recall	F-measure
Double Exp Smoothing (alpha = 1 beta = 0.2)	0.746	0.839	0.823	0.831

Выводы по сравнению методов прогнозирования

В процессе поиска решений данной задачи проведено большое количество исследований, в рамках которых было установлено, какой метод больше всего подходит для предсказания поведения прокси-серверов. По итогу полученных результатов наилучшим инструментом оказался метод экспоненциального сглаживания. Помимо того, что этот метод показал отличнейшие результаты по сравнению с конкурентами, он также является наиболее простым для использования, так как не требует затрат ресурсов на предварительное обучение.

Общая архитектура программы

Программа состоит из пяти основных модулей, представленных на рисунке 6:

— gtp_scrapy — реализует компонент сбора информации о прокси-серверах из публичных источников;
— gtp_manager — реализует менеджер задач;
— gtp_checker — реализует компонент проверки доступности прокси-серверов;
— gtp_rest — реализует API доступа к данным;
— gtp_common — пакет, в котором содержатся вспомогательные классы, которые необходимы сразу в нескольких компонентах.

Модель хранения данных

Компонент gtp_scrapy

Компонент gtp_scrapy отвечает за сбор данных о прокси-серверах с публичных веб-ресурсов, которые занимаются публикацией этих данных. Как правило, такие ресурсы имеют доход с рекламы, но есть и такие, которые предлагают платный доступ к полному списку их базы прокси-серверов, а бесплатно предоставляют только некоторую часть этого списка.

Компонент gtp_scrapy реализован при помощи бесплатного фреймворка с открытым исходным кодом Scrapy[2]. Упомянутый фреймворк создан для разработки приложений, которым необходимо извлекать информацию с веб-сайтов и хранить ее в различных представлениях.

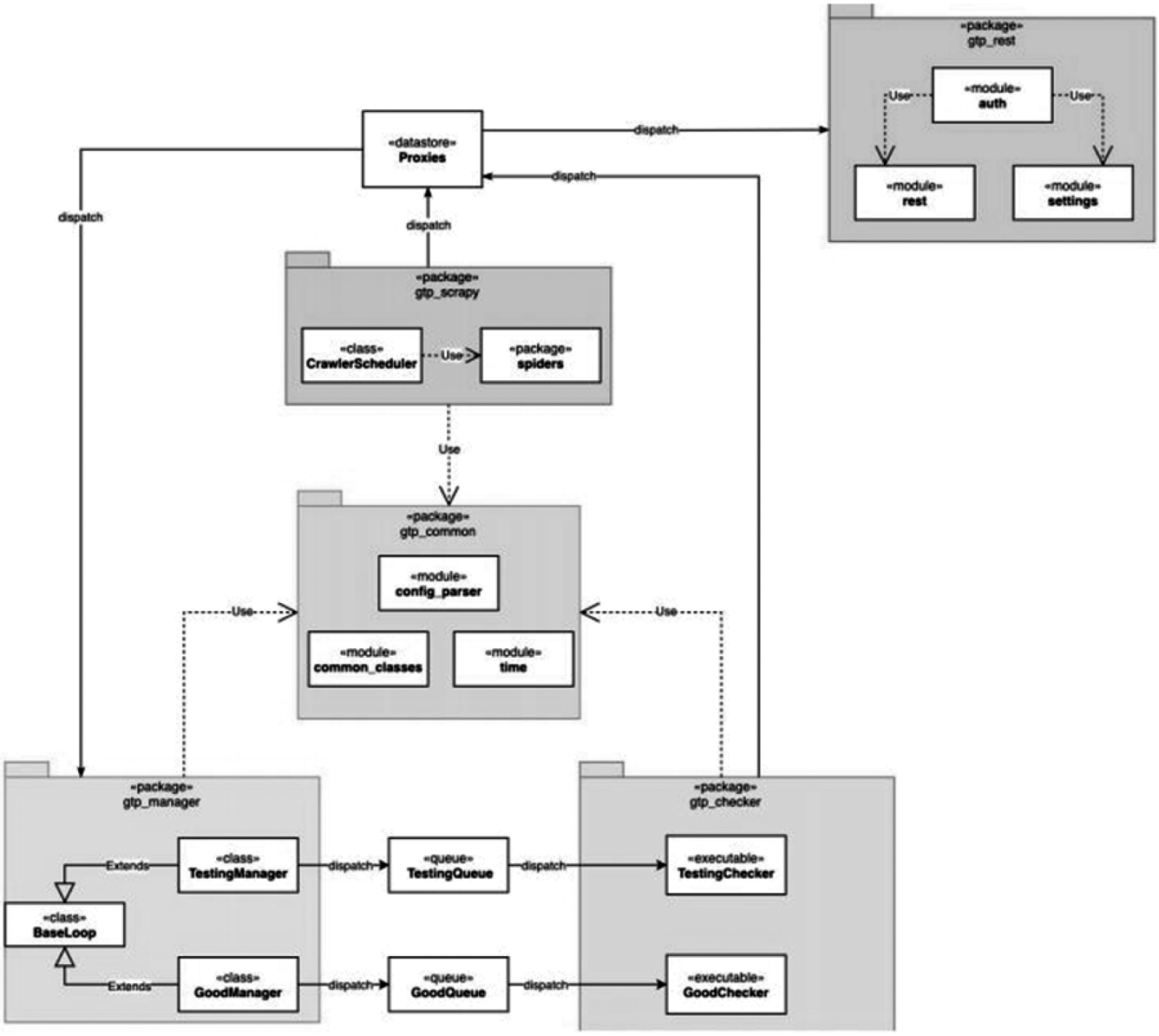

Рис. 6. Архитектура программы сбора и анализа данных о доступности прокси серверов

В этой системе было использовано 10 spider-классов, каждый из которых ответственен за получение данных из 1–2 веб-ресурсов. На вход spider-класс получает загруженную веб-страницу в HTML [3] формате. Извлечение данных происходит в результате обхода дерева тегов веб-страницы и обнаружения тегов, подходящих под шаблоны, заданные в spider-классе. В случаях, если необходимая информация содержится более чем на одной странице веб-ресурса, то на всех полученных веб-страницах данного ресурса производится поиск ссылок, которые соответствуют заранее описанному шаблону. В следующем шаге происходит переход на обнаруженные ссылки и процесс получения данных повторяется. Процесс повторяется, пока все обнаруженные страницы не будут обработаны.

У источников есть собственный интервал обновления списка прокси-серверов, и для организации сбора данных, согласно этому интервалу, был разработан вспомогательный класс, ответственный за расписание сбора информации с источников. Этот класс начинает процесс старта каждого spider'а и записывает время старта в кэш. Следующие инициализации пауков начинаются по прошествии времени, обозначенном в конфигурационном файле. Время проставляется отдельно для каждого spider'а.

Многие ресурсы, которые публикуют информацию о публичных прокси-серверах, пользуются средствами для предотвращения автоматизированного сбора данных об этих серверах. Как правило, защита базируется на использовании JavaScript кода, который производит изменения страницы после того, как она загрузилась. Иногда, необходимые данные находятся внутри JavaScript кода в открытом виде. Тогда необходимо полу-

чить JavaScript код в виде строки и после этого из строки доставать необходимые данные. Помимо этого, были замечены несколько ресурсов, которые определяли номера портов при помощи специальных JavaScript методов, и получить необходимые данные из страницы уже не представлялось возможным предыдущими способами. Для решения этой проблемы была подключена сторонняя библиотека js2py[4], которая позволяет транслировать JavaScript код в Python код. Кроме того, было поставлено ограничение на количество запросов до 1 в секунду во избежание блокировки веб-ресурсом из-за превышения количества запросов.

Компонент gtp_manager

Компонент gtp_manager ответственен за отправку заданий компоненту gtp_checker. С заданным в файле конфигурации промежутком времени gtp_manager запрашивает у базы данных сервера, не стоящих в очереди на проверки и у которых с последней проверки прошло большое количество времени. В базе данных для каждого сервера проставлены поля enqueued, в которое вносится время последней отправки прокси-сервера на проверку, и last_update, в котором проставляется время последней проверки прокси-сервера. Если last_update в базе меньше, чем enqueued, значит сервер уже стоит в очереди на проверку и не нуждается в повторном отправлении.

Для того чтобы эти байты не были интерпретированы как служебная информация протокола передачи сообщений AMQP [5], они кодируются с помощью стандарта кодирования base64[6] и в закодированном виде передаются в очередь сообщений.

Проверка доступности прокси-серверов

Компонент gtp_checker служит для проведения проверок откликов прокси-серверов и внесения данных о результатах проверок в базу данных. Для осуществления проверок применяются следующие методики:

1. К определённому сетевому ресурсу осуществляется запрос через нужный прокси-сервер. Чтобы убедиться, что запрос отправился именно через определённый прокси-сервер, в качестве конечной точки используется специальные ресурсы, которые в ответ отправляют адрес клиента, например, http://ident.me. Время ожидания ответного запроса ограничено 20 секундами.
2. В базу данных вносится следующие данные о результатах проверки:

— HTTP статус код ответа;
— время отклика;
— адрес целевого ресурса;
— адрес клиента, который вернул целевой ресурс;
— время проведения проверки.

Критерием успешности прохождения теста для прокси-сервера служит запись об ответном запросе, пришедшем в течение 20 секунд и, если статус код этого запроса валидный (равен 2xx).

Описание классов, схема которых изображена на рисунке 7:

1. CheckModel — промежуточное представление проверки
— condition — главная информация об объекте, который необходимо обновить;
— check — данные о результатах проверки, которые будут добавлены в базу.
— meta — данные, которые будут обновляться (счетчики проверок и т.д.)
2. BaseProvider — базовый класс для описания взаимодействий с СУБД
— flush_updates — абстрактный метод обновления данных в БД.
3. MongoProvider — реализация BaseProvider для взаимодействия с MongoDB
— collection — инкапсулирует соединение с базой данных;
— flush_updates — реализация обновления данных для MongoDB.
4. CachingUpdater — класс, реализующий функционал буфера для проверок
— cache — буфер, служащий для временного хранения результатов проверок;
— provider — экземпляр одного из наследников класса BaseProvider;
— store_results — хранит массив объектов результатов проверок в буфере кэша.

Архитектура очередей сообщений

Как уже упоминалось ранее, в процессе функционирования программы прокси-сервера могут быть помещены в одну из трёх групп: *тестируемая*, *активная* и *неактивная*. Главным отличием этих групп друг от друга служит то, что проверки для прокси-серверов из разных групп инициализируются с различной периодичностью, и чтобы поддерживать частоту запуска этих проверок, каждая группа (кроме *неактивной*) независимо от других обрабатывается своим менеджером задач. Менеджер задач формирует задачи на проверку прокси-серверов своей группы и отправляет их в свою очередь сообщений. Далее, те процессы компонента проверки, которые настроены на чтение из соответствующей очереди, получают задачи, выполняют их и записывают результаты этих задач в базу данных Схема работы изображена на рисунке 8.

Использование очередей сообщений позволяет решить сразу две задачи:

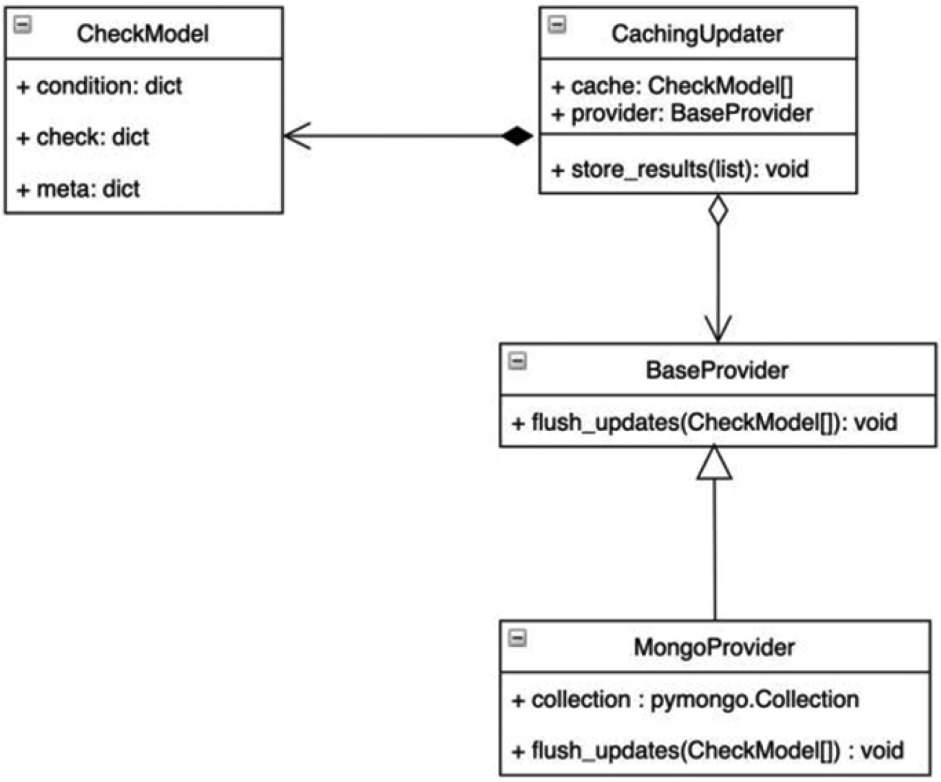

Рис. 7. Диаграмма классов checker'a

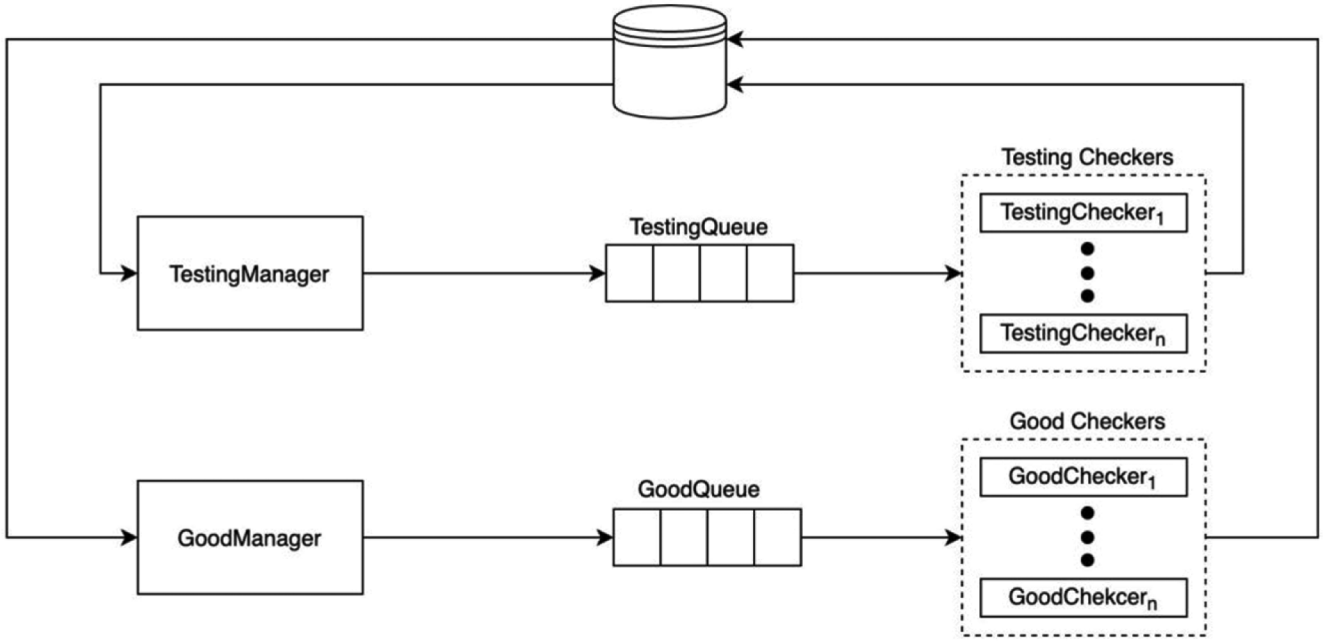

Рис. 8. Схема работы очередей сообщений

— Проверка доступности прокси-серверов из разных групп осуществляется независимо друг от друга, что даёт возможность выполнять проверки в каждой группе с необходимой частотой.
— Так как связь с очередью сообщений происходит посредством сетевого взаимодействия, то в случае необходимости маштабирования можно запустить несколько процессов компонента проверки на разных компьютерах, что значительно увеличит количество выполняемых проверок.

Для организации работы компонентов с очередью сообщений использовался фреймворк Celery [7], а в качестве поставщика очередей сообщений был выбран инструмент RabbitMQ [8].

Компонент gtp_rest

Данный компонент ответственен за доступ к данным с помощью API. Он разработан при помощи асинхронного (событийно-ориентированного) фреймворка twisted[9] в виде REST API сервиса, который принимает HTTP GET запросы и возвращает HTTP-ответ со сведениями о прокси-сервере в формате JSON[10].

После того, как сервис получил запрос от пользователя, программа обрабатывает отправленные пользователем аргументы и формирует новый запрос к базе данных. Получив ответ от базы данных, программа формирует из него JSON-строку и отправляет ее обратно пользователю.

Мониторинг

В разрабатываемой программе используется система мониторинга Prometheus [11]. Prometheus на постоянной основе берёт значения заранее описанных метрик из компонентов программы и хранит их в виде временных рядов в базе данных. Также, в Prometheus есть возможность использовать язык запросов PromQL, который позволяет гибко собирать и обрабатывать хранимые данные. Для удобства визуализации полученных из компонентов метрик развёрнут сервис для интерактивного отображения данных Grafana [12]. Схему работы мониторинга отображена на рисунке 9:

Мониторинг функционирует по следующим принципам:

1. В каждом компоненте определяются необходимые для отображения метрики и логика их изменения;
2. При старте компонента, он также запускает HTTP target-сервер на определенном порту;
3. Каждые 15 секунд Prometheus сервер пуллит из каждого target-сервера все необходимые метрики и записывает их в базу данных в виде временных рядов;
4. Grafana Server по запросу пользователя берёт из Prometheus сервера нужные данные и представляет их в удобном для визуализации виде.

В таблице 5 указаны основные отображаемые метрики для каждого компонента

Примеры отображения данных с помощью сервиса Grafana показаны на рисунках 10 и 11.

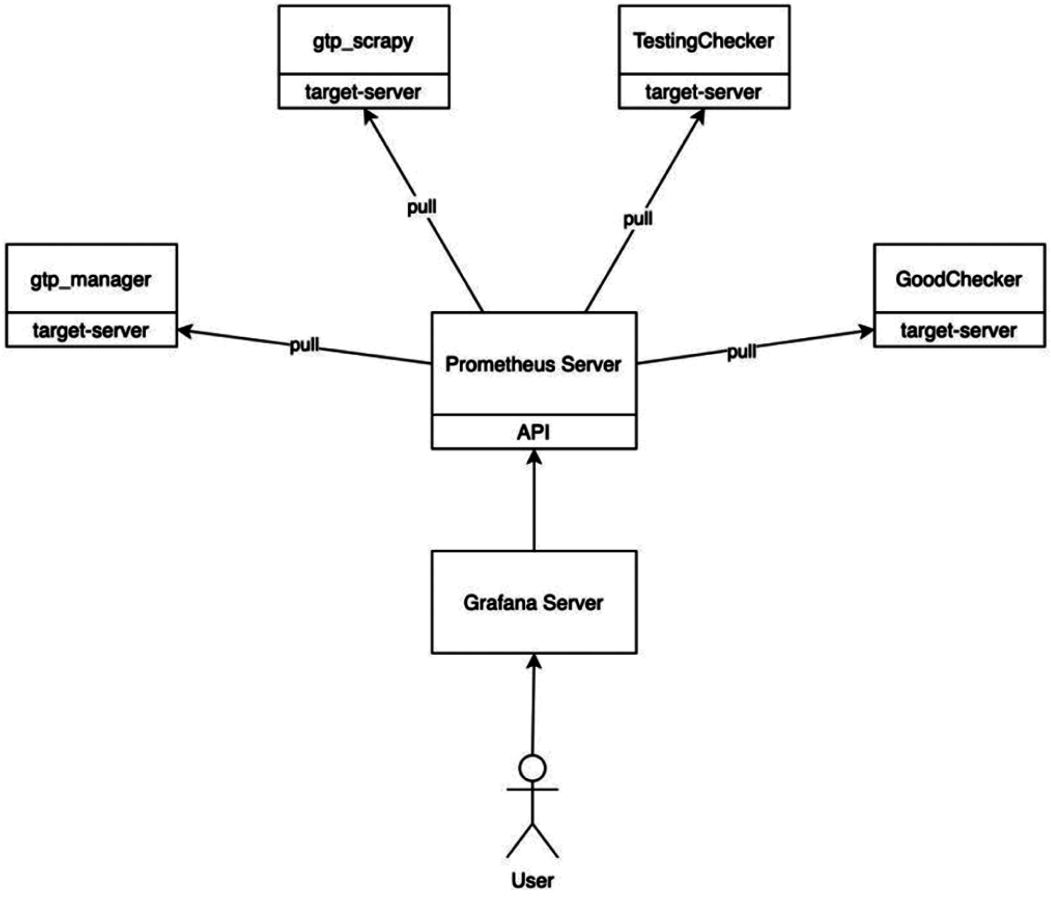

Рис. 9. Схема мониторинга проекта

Таблица 5.

Отображаемые метрики для каждого компонента

Компонент	Метрика
gtp_scrapy	Число обнаруженных прокси-серверов
	Средняя скорость обнаружения Количество новых прокси-серверов Количество дублей Распределение обнаруженных прокси-серверов по источникам
	Число возникших ошибок
gtp_manager	Число отправленных заданий по очередям сообщений
gtp_checker	Число выполненных проверок доступности
	Среднее время получения ответа от прокси-серверов
	Распределение полученных от прокси-серверов ответов по HTTP статус кодам
	Число успешных проверок
	Число проваленных проверок
	Число обнаруженных ошибок

Тестирование работоспособности

В качестве тестирования работоспособности выбранного метода проведем следующее исследование: вытащим через API 10 прокси-серверов с готовыми прогнозами доступности и отправим на каждый прокси-сервер по 10 запросов в течение 10 минут и определим процент успешных запросов и среднее время получения ответа. Далее повторим тот же эксперимент, но уже для 10 прокси-серверов, которые были отобраны случайным образом, и после этого сравним результаты двух тестов. На момент начала исследования программа функционировала на протяжении 16 дней, чуть больше 2 недель, и в базе данных находилось около 685 тысяч прокси-серверов. Результаты отображены в таблице 6.

Таблица 6.

Результаты эксперимента

	Процент успешных запросов	Среднее время отклика
Прокси-сервера из API	84 %	3617 мс
Случайные прокси-сервера	0 %	—

Из таблицы 6 можно сделать вывод, что сервера, выбранные случайном образом не ответили ни разу, на самом деле такой результат был ожидаем, так как во время проведения исследования было установлено, что 98 % публичных прокси-серверов являются неработоспособными. Сервера со спрогнозированной работоспособностью оказались успешными на 84 %, более того, на первый запрос ответили все сервера без исключения.

Рис. 10. Примеры отображения счётчиков и таблиц

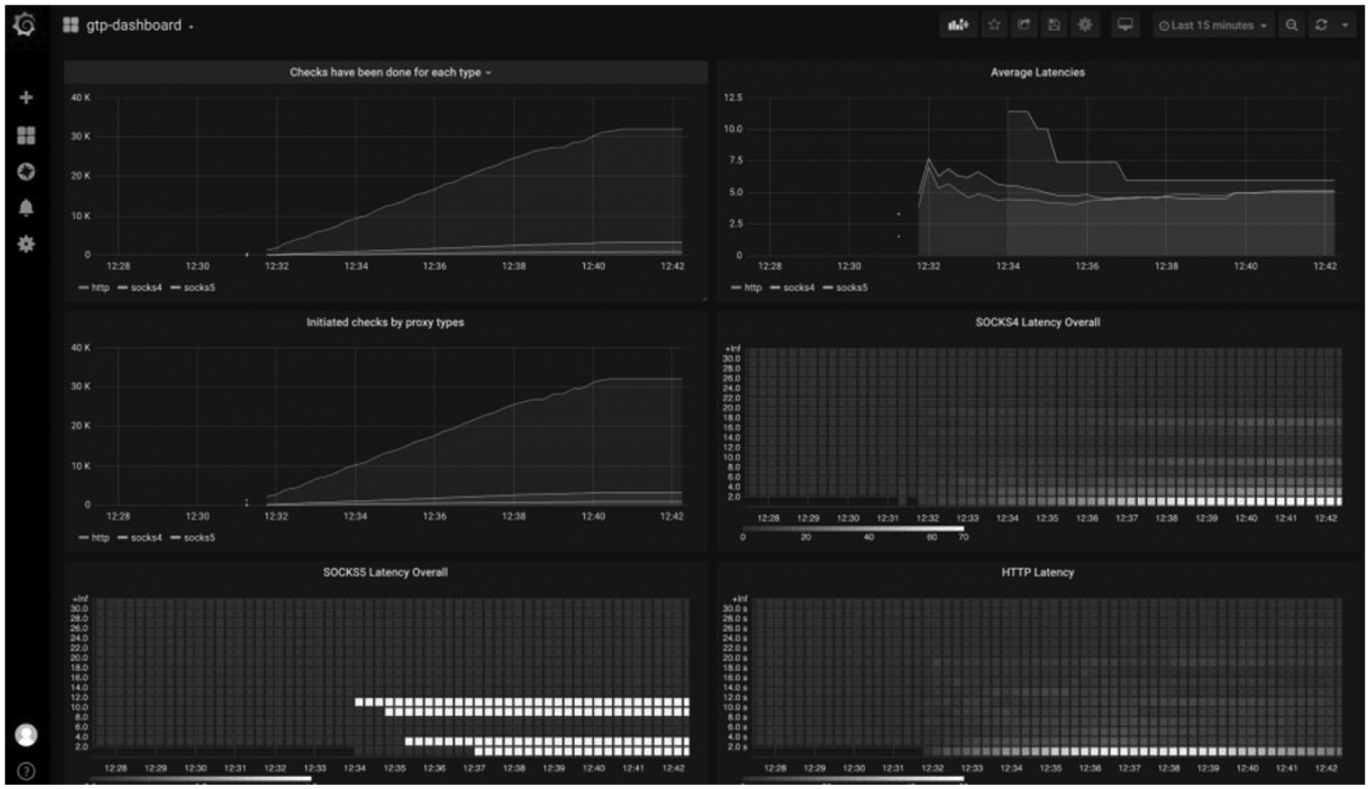

Рис. 11. Примеры отображения графиков и тепловых карт

Заключение

Использование публичных прокси-серверов является наиболее оптимальным среди всех рассмотренных ранее в данной работе, во многом благодаря бесплатного предоставления услуг и возможности сохранения анонимности пользователя, а также из-за простоты реализации. Но низкое качество предоставляемых услуг сильно сказывается на возможности их полноценного использования. Подход к решению данной проблемы требуют глубокого погружения и многочисленных исследований.

Среди готовых решений не было вариантов, которые могли бы в полной мере удовлетворить потребности в повышении качества предоставляемых услуг, поэтому было решено приступить к разработке собственной программы, позволяющей без активного участия пользователя проверять и собирать информацию о доступности прокси-серверах из публичных ресурсов, анализировать их работоспособность и анализировать их доступность в ближайшем будущем. Используя отправку запроса на публичных веб-ресурс через прокси, можно было проверить работоспособность, опираясь на ответ или на задержку запроса. Для прогнозирования были проверены некоторые алгоритмы машинного обучения, а также задействовано экспоненциальное сглаживание.

ЛИТЕРАТУРА

1. Cross Validation — метод оценки аналитической модели [Электронный ресурс] URL: https://scikit-learn.org/stable/modules/cross_validation.html (дата обращения 20.11.2023)
2. Scrapy — фреймворк для веб-краулинга [Электронный ресурс] URL: https://scrapy.org/ (дата обращения 20.11.2023)
3. HTML — язык разметки гипертекста [Электронный ресуср] URL: https://tools.ietf.org/html/rfc1866 (дата обращения 20.11.2023)
4. js2py — библиотека трансляции JavaScript кода в Python код [Электронный ресурс] URL: https://pypi.org/project/Js2Py/ (дата обращения 20.11.2023)
5. AMQP — открытый протокол обмена сообщениями между компонентами системы [Электронный ресурс] URL: http://www.amqp.org/resources/download (дата обращения 20.11.2023)
6. Base64 — стандар кодирования двоичных данных [Электронный ресурс] URL: https://tools.ietf.org/html/rfc4648 (дата обращения 20.11.2023)
7. Celery — фреймворк для работы с очередью сообщений [Электронный ресурс] URL: http://www.celeryproject.org/ (дата обращения 20.11.2023)
8. RabbitMQ — очередь сообщений [Электронный ресурс] URL: https://www.rabbitmq.com/ (дата обращения 20.11.2023)
9. Twisted — событийно-ориентированный сетевой фреймворк [Электронный ресурс] URL: https://twistedmatrix.com/trac/ (дата обращения 20.11.2023)
10. JSON — текстовый формат обмена данными [Электронный ресурс] URL: https://www.json.org/json-ru.html (дата обращения 20.11.2023)
11. Prometheus — система мониторинга [Электронный ресурс] URL: https://prometheus.io/ (дата обращения 20.11.2023)
12. Grafana — система визуализации результатов мониторинга [Электронный ресурс] URL: https://grafana.com/grafana/ (дата обращения 20.11.2023)