Математическое моделирование и методы внешней балансировки нагрузки в кластерах серверов

Р.В. Сущеня

Московский государственный технологический университет «Станкин»

Аннотация: Статья посвящена разработке и экспериментальной проверке внешнего механизма балансировки нагрузки для кластеров серверов, обслуживающих распределённую образовательную сеть. Предложен гибридный подход, сочетающий классические методы (Round Robin, Least Connections) с эволюционным поиском на базе генетического алгоритма. На модельном уровне показано, как задача распределения сеансов пользователей формулируется через минимизацию максимальной загрузки и ограничение задержек. Разработанное решение реализовано целиком на отечественном технологическом стеке — кластерах серверов платформы «1С: Предприятие», контейнерах Docker и интеграционной шине «1С: Шина». Экспериментальное исследование демонстрирует, что новая логика распределения повышает устойчивость системы при колебаниях трафика, снижает задержки для пользователей и рациональнее использует резервные ресурсы, при этом не создавая заметной дополнительной нагрузки на управляющие узлы. Работа подтверждает практическую пригодность эволюционных методов в задачах онлайн-балансировки.
Ключевые слова: балансировка нагрузки, кластеры серверов, генетический алгоритм, имитационное моделирование, 1С: Шина.

Введение

Балансировка нагрузки – это крайне важный процесс, который отвечает за распределение входящих задач и запросов между различными узлами вычислительной сети. Корректное распределение работы между серверами предотвращает как перегрузку отдельных узлов, так и их простой, обеспечивая высокую эффективность использования ресурсов и надежность системы [1, 2]. В контексте кластерных систем задача балансировки приобретает особую актуальность: современные онлайн-сервисы и облачные платформы предъявляют динамично изменяющиеся нагрузки, требующие гибкого перераспределения ресурсов в режиме реального времени.

Особое внимание уделяется внешней балансировке нагрузки между кластерами серверов – то есть распределению работы не только внутри одного кластера, но и между географически или функционально разнесенными кластерами. Такой подход используется, например,

крупнейшими облачными платформами: запросы пользователей могут перенаправляться на разные дата-центры через механизм DNS-балансировки, что позволяет масштабировать систему по регионам и избегать перегрузки одного центра [3].

Актуальность исследования обусловлена потребностью в формальных методах, позволяющих оптимально решать задачи распределения нагрузки в условиях повышенных требований к производительности и надежности. Классическая проблема балансировки нагрузки в кластерах относится к NP-трудным задачам оптимизации [4]. Следовательно, необходимы специальные методы – точные (математическое программирование, графовые алгоритмы) и приближенные (эвристики, метаэвристики, машинное обучение) – для поиска квазиоптимальных решений за разумное время [5, 6].

Цель работы – разработать и проанализировать строгие математические модели и методы внешней балансировки нагрузки между кластерными системами серверов. Для достижения этой цели в статье решаются следующие задачи:

1. обзор современных методов решения;
2. формулировка основных вариантов задач балансировки;
3. разработка концепции имитационного моделирования для оценки предложенных решений;
4. обсуждение результатов и методов верификации модели.

Современные подходы к балансировке нагрузки серверных кластеров

Современная научная литература предлагает множество подходов к балансировке нагрузки в кластерных и облачных системах – от традиционных алгоритмов до гибридных интеллектуальных методов. Задача балансировки напрямую связана с обеспечением качества сервисов (QoS), показателей производительности и соблюдения соглашений об уровне

обслуживания в облачных инфраструктурах [7]. Далее рассмотрены основные группы методов, выделенные в литературе.

Классические подходы включают статические алгоритмы (например, Round Robin (RR), статическое хеширование по IP) и динамические алгоритмы (например, наименьшего числа соединений – Least Connections (LC), наименьшего времени отклика – Least Response Time, адаптивные методы). Статические алгоритмы не учитывают текущую обстановку и распределяют запросы по фиксированному правилу; они просты и часто используются для небольших однородных кластеров [8, 9]. Динамические алгоритмы способны реагировать на изменение нагрузки – мониторят состояние серверов и перераспределяют трафик к менее загруженным узлам, добиваясь более равномерной загрузки и сниженного времени отклика [10]. Приоритетное взвешенное круглое чередование (Priority Weighted Round Robin) – пример улучшенного статического алгоритма, в котором серверам назначаются веса по приоритету, что позволяет учитывать различия в производительности [11]. Аппаратные балансировщики обеспечивают высокую пропускную способность за счет специализированных интегральных схем, однако их недостаток – высокая стоимость и негибкость (сложно менять заложенный алгоритм) [12].

Балансировка на уровне глобальной инфраструктуры. Для распределения нагрузки между удаленными кластерами (например, между дата-центрами) часто применяются методы на уровне DNS и сети. DNS-балансировка (балансировка через систему доменных имен) распределяет запросы пользователей на разные IP-адреса (разные центры обработки данных) на основе географического положения или текущей загрузки центров [3]. Другой подход – использование иерархии балансировщиков: глобальный (GeoDNS или Anycast) балансирует между кластерами, а локальные балансировщики распределяют нагрузку внутри каждого

кластера. В литературе подчеркивается, что для международных облачных платформ многоуровневая система балансировки является ключом к обеспечению масштабируемости и отказоустойчивости [13, 14].

Графовые модели и постановка балансировки как задачи на графах. Распределение задач по серверам можно моделировать двудольным графом, где одна доля – набор задач (или пользователей), другая – серверы (или кластеры). Был предложен динамический алгоритм балансировки на основе взвешенного двудольного графа: в их методе оценивается вычислительное время каждого блока задачи, на основе чего формируется двудольный граф “задачи–узлы” с весами, равными времени вычисления [15]. Затем с помощью модифицированного алгоритма Венгера находится оптимальное соответствие задач и узлов, минимизирующее разброс нагрузки, после чего задачи распределяются по вычислительным узлам в реальном времени [16]. Данный подход продемонстрировал заметное выравнивание загрузки в вычислительном кластере цифрового двойника энергосистемы [17].

Робастные методы и адаптация к изменяющейся нагрузке. Классические оптимизационные модели предполагают детерминированные параметры (объемы нагрузки, скорость серверов и пр.). В реальных же системах нагрузка может значительно меняться со временем, а данные о будущих поступлениях задач носят вероятностный или вовсе неизвестный характер. Робастная оптимизация позволяет учесть неопределенность, обеспечивая гарантированно удовлетворительное решение в худшем случае. Плата за робастность – возможно менее оптимальное распределение для усредненного случая, но выигрыш – в устойчивости решения.

Машинное обучение и интеллектуальные методы. С 2022 года заметен всплеск работ, в которых балансировка нагрузки решается методами искусственного интеллекта. Используются как обучение с учителем/без

учителя (для прогнозирования нагрузки или классификации состояния системы), так и обучение с подкреплением (для выработки оптимальной стратегии распределения). Например, глубокое обучение привлекается для прогнозирования объема запросов. CNN-BiLSTM модель для прогнозирования нагрузки и обеспечения проактивного распределения ресурсов [16].

Метаэвристические алгоритмы. Для решения недетерминированных полиномиальных (non-deterministic polynomial, NP) трудных задач балансировки широко применяются эвристики и метаэвристики – генетические алгоритмы (genetic algorithms, GA), имитация отжига (simulated annealing, SA), алгоритмы роя частиц (particle swarm optimization, PSO) и др. Большинство работ используют гибридные алгоритмы, комбинирующие преимущества разных методов. Например, был предложен гибрид алгоритмов Water Wave Optimization (WWO) и Ant Colony Optimization (ACO), который превзошел по эффективности каждую из методик по отдельности, снизив среднее время расписания задач на 11% и энергопотребление на 12% по сравнению с классическими методами [18]. Другой пример – сочетание алгоритма PSO с генетическим алгоритмом: предложен гибридный DPSO-GA для прогнозирования нагрузки и распределения виртуальных машин, демонстрирующий лучшие показатели по времени отклика и утилизации ресурсов по сравнению с единичными методами [19].

Существуют также оригинальные метаэвристики, созданные специально под задачи облачного распределения. Так, предложен Intercrossed Chimp and Bald Eagle Algorithm (IC&BA) – гибрид двух биоинспирированных алгоритмов (шимпанзе и орлан) для оптимальной балансировки в облаке [20]. Также было показано, что алгоритм PSO часто превосходит другие по

улучшению таких показателей, как минимизация времени выполнения и степень дисбаланса нагрузки [21].

Формализация задачи балансировки нагрузки

На основе обзора можно выделить несколько формальных постановок задачи внешней балансировки нагрузки между кластерами. В данном разделе представлены математические модели для ключевых вариантов задачи.

Графовая модель балансировки. Эта модель представляет систему в виде двудольного графа или поточной сети, что позволяет применять алгоритмы теории графов. Если каждая задача должна быть назначена ровно одному кластеру, и кластеры могут обрабатывать любое число задач (нет явного ограничения на число задач у кластера), то задача сводится к нахождению совершенного (полного) совпадения минимальной стоимости. Это классическая задача назначения, решаемая за полиномиальное время алгоритмом Венгера за $O(n^3)$. Однако такое решение не гарантирует равномерности нагрузки: его цель – минимизировать суммарное время, что может приводить к перекосу (более мощный кластер получит гораздо больше задач, чем слабый, хотя закончит их быстрее – но временная синхронизация не учитывается).

Min-Cut подход: если кластеры географически распределены и связь между ними ограничена, балансировку можно рассматривать с точки зрения разбиения графа. Задача: отделить часть вершин (серверов) с перегрузкой от других так, чтобы минимизировать "стоимость" переноса некоторой доли задач через разрез.

Графовые модели особенно полезны при наличии дополнительных связей: совместимости задач. В работах предлагается модель двудольного графа совместимости: типы задач связаны только с поддерживающими их серверами, и исследуется, как неполнота двудольного графа влияет на распределение нагрузки [22, 23].

Модель включения кластеров с ограничениями. В некоторых случаях возникает задача не только распределять нагрузку, но и выбирать, какие кластеры задействовать. При малой нагрузке достаточно держать включенным один кластер, при росте нагрузки можно подключать дополнительные.

Практический смысл модели: она важна для сценариев авто-масштабирования. В облачных платформах часто реализованы механизмы, автоматически запускающие новые серверы (или контейнеры, или даже целые кластеры) при росте нагрузки. Если нагрузка упала, лишние узлы отключаются, экономя ресурсы.

Модель можно усложнить, учитывая время включения кластеров (при реальном запуске сервера есть задержка, т.н. холодный старт), или ограничение на число переключений (чтобы не включать-выключать слишком часто).

Методы решения задач балансировки

Графовый алгоритм максимального потока (Max-Flow): классический алгоритм Форда–Фалкерсона и его улучшения (Эдмондса–Карпа, Диница и др.) позволяют найти максимальный поток из источника в сток за время $O(VE^2)$ или лучше. В контексте балансировки максимальный поток может найти, сколько задач максимум можно пропустить при данных ограничениях пропускной способности кластеров. Если максимальный поток $< n$ (количества задач), значит, некоторые задачи некуда девать – балансировка невыполнима без отказов. Если $= n$, то есть хотя бы один способ распределить все задачи. Такие алгоритмы используют, когда нужно быстро просчитать оптимальную маршрутизацию запросов в сетях доставки контента или распределенных системах: граф строится по текущей

топологии и задержкам, и вычисляется поток (распределение запросов) минимальной задержки.

Деревья распределения нагрузки: под этим термином могут подразумеваться различные структуры. Один из вариантов – построение иерархического дерева балансировки: корень – главный балансировщик, листья – конечные серверы, а внутренние узлы – промежуточные балансировщики, которые делят нагрузку на подгруппы серверов. Оптимизация такой структуры сводится к определению долей нагрузки, передаваемых по каждой ветви дерева. Алгоритмы типа желчного дерева (load distribution tree, CDN) используются в сетях доставки и некоторых распределенных кластерах: они организуют маршрутизацию запросов от корневого узла к серверным узлам по дереву, стараясь минимизировать время до ответа. Выбор оптимальной структуры дерева и распределения потоков – нетривиальная оптимизационная задача, обычно решаемая эвристически или на основе эволюционных алгоритмов.

В целом, графовые алгоритмы полезны для балансировки, когда можно свести задачу к транспортной (потоки, разрезы). Их преимущество – полиномиаальная вычислительная сложность и возможность использования богатого арсенала теории графов. Недостаток – необходимость упростить модель (часто не удается выразить все нюансы, например, разную стоимость миграции разных задач).

Генетический алгоритм: стохастический оптимизационный метод, основанный на принципах естественного отбора и генетической эволюции. Он особенно эффективен в задачах комбинаторной природы с большим пространством допустимых решений и сложной структурой ограничений.

Решение кодируется в виде особи (индивида) — вектора $x = (x_1, x_2, \dots, x_n)$, где каждая компонента $x_i \in \{1,2, \dots, m\}$ указывает, на какой кластер назначена задача i, а m — количество доступных кластеров.

Рассмотрим основные блоки схемы генетического алгоритма, приведенной на рис.1.

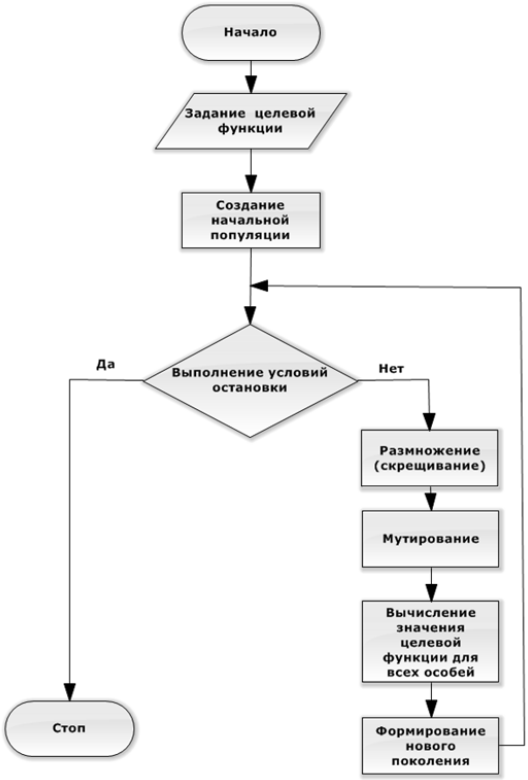

Рис. 1. – Блок-схема алгоритма генетического алгоритма

1. **Задание целевой функции**: определяется функция приспособленности $f(x)$, отражающая эффективность текущего распределения задач по кластерам. Один из наиболее распространённых вариантов — минимизация максимальной нагрузки на кластеры:

$$f(x) = -\max_{j\in\{1,\dots,m\}}\left(\sum_{i:x_i=j} w_i\right),$$

где $x = (x_1, x_2, \dots, x_n)$ — вектор, обозначающий, какой кластер x_i назначен для задачи i;

w_i — вес (нагрузка) задачи i;

m — количество кластеров.

2. **Создание начальной популяции**: формируется стартовая популяция $P^{(0)}$ из N особей (векторов решений):

$$P^{(0)} = \{x^{(1)}, x^{(2)}, \dots, x^{(N)}\}, \quad x^{(k)} \in \{1, \dots, m\}^n.$$

Каждое $x^{(k)}$ — отдельное распределение задач по кластерам.

3. **Выполнение условий остановки**: проверяется, выполнено ли хотя бы одно из условий:

- достигнуто максимальное число поколений T;
- достигнута требуемая точность: $f(x^*) \geq f_{\text{target}}$;
- за K поколений не произошло прироста: $\Delta f < \varepsilon$.

Если условие выполнено — переходим к завершению. Если нет, выполняем основные генетические операторы.

4. **Размножение (скрещивание)**: для каждой пары родителей x^A, x^B формируется новая особь x^C, представляющая собой возможное распределение задач по кластерам. Скрещивание — ключевая операция, позволяющая комбинировать признаки (гены) лучших решений и искать более удачные комбинации. С вероятностью p_c применяется одноточечный кроссовер:

$$x_i^C = \begin{cases} x_i^A, & i \leq r \\ x_i^B, & i > r \end{cases}, r \in \{1, \dots, n-1\},$$

где x_i^C — значение i-го гена (задачи) в новой особи.

x_i^A, x_i^B — соответствующие гены родительских решений x^A и x^B.

r — случайно выбранная точка разрыва между 1 и $n-1$, где n — общее количество задач.

Все задачи с индексами $i \leq r$ наследуются от первого родителя, а с $i > r$ — от второго.

Или равномерный кроссовер:

$$x_i^C = \begin{cases} x_i^A, \theta_i \leq 0.5 \\ x_i^B, \theta_i > 0.5 \end{cases}, \theta_i \sim U(0,1),$$

где θ_i — случайная величина, сгенерированная для каждого гена индивидуально из равномерного распределения $U(0,1)$.

Если $\theta_i \leq 0.5$, то ген копируется от x_i^A, иначе — от x_i^B.

Такой тип скрещивания обеспечивает более равномерное смешение родительских признаков.

5. **Мутирование**: Каждый ген x_i у потомка подвергается мутации с вероятностью p_m:

$$x_i \coloneqq \mathrm{Random}(\{1, \dots, m\} \setminus \{x_i\}).$$

Мутации увеличивают разнообразие и предотвращают застревание в локальном оптимуме.

6. **Вычисление значения целевой функции для всех особей**: на этом этапе каждая особь (распределение задач по кластерам) из текущей популяции оценивается с помощью целевой функции. Функция приспособленности $f(x)$ отражает максимальную суммарную нагрузку на один из кластеров, то есть ищет так называемое бутылочное горлышко в распределении. Чтобы сделать задачу удобной для оптимизации с помощью генетического алгоритма (который по своей природе ищет максимум), используется отрицательное значение пикового кластера:

$$f(x) = -\max_{j \in \{1,\dots,m\}} \left(\sum_{i:x_i=j} w_i\right),$$

где $x = (x_1, x_2, \dots, x_n)$ — распределение n задач по кластерам;

$x_i \in 1,2, \dots, m$ — кластер, на который назначена задача i;

w_i — вычислительная нагрузка задачи i;

m — общее количество кластеров;

$\sum_{i:x_i=j} w_i$ — суммарная нагрузка на кластер j;

$\max\limits_{j \in 1,\dots,m}$ — оператор, выбирающий наибольшую из всех нагрузок;

$f(x)$ — значение функции приспособленности для особи x (чем больше, тем лучше).

7. **Формирование нового поколения**: после выполнения операций скрещивания и мутации, новая популяция формируется с учётом сохранения наиболее приспособленных особей. Оставшаяся часть популяции

заполняется потомками, отобранными на основании значений функции приспособленности.

$$P^{(t+1)} = \text{BestSubset}\left(P^{(t)}, e\right) \cup \text{SelectedOffspring}\left(P^{(t)}\right),$$

где $P^{(t+1)}$ — популяция следующего поколения;

$P^{(t)}$ — популяция на текущем поколении t;

$\text{BestSubset}\left(P^{(t)}, e\right)$ — подмножество из e наиболее приспособленных особей (с максимальными значениями $f(x)$), сохранённых без изменений;

$e \in N$ — количество сохраняемых лучших решений;

$\text{SelectedOffspring}\left(P^{(t)}\right)$ — множество потомков, отобранных на основе их приспособленности $f(x)$ после применения операторов скрещивания и мутации.

8. **Остановка алгоритма**: на этом этапе из последней популяции выбирается наиболее приспособленное решение — особь с наибольшим значением функции приспособленности $f(x)$:

$$x^* = \arg\max_{x \in P^{(T)}} f(x),$$

где x^* — финальное решение, соответствующее оптимальному распределению задач по кластерам;

$P^{(T)}$ — финальная популяция после T поколений;

$f(x)$ — функция приспособленности, вычисляемая согласно шагу 6;

arg max — аргумент, на котором достигается максимум $f(x)$.

Процесс повторяется, пока не достигнут критерий остановки (фиксированное число поколений или отсутствие улучшения). Генетический алгоритм обладает высокой степенью параллелизма, поскольку оценка приспособленности отдельных особей может выполняться независимо. Он также допускает использование сложных и неявных целевых функций. Вместе с тем, основными его недостатками являются необходимость настройки большого количества параметров и сравнительно низкая

эффективность при решении задач малой размерности, где предпочтительнее применение точных методов оптимизации.

Имитационное моделирование

Предлагаемый внешний балансировщик был апробирован на инфраструктуре учебного центра 1С, объединяющей свыше 10 серверов-узлов. Все входящие запросы к авторизации в приложениях, генерации сложной аналитической отчетности, получения видеолекций и иных обучающих материалов сначала попадают во внешний программный балансировщик, который в режиме реального времени выбирает оптимальный кластер серверов/сервер для выдачи требуемых данных. Балансировщик получает телеметрию (загруженность ЦПУ, загруженность ОЗУ, активные сессии, загруженность канала передачи данных, загруженность дисковых устройств и т. д.) от узлов через «1С:Шину» и принимает решение на основе весовых коэффициентов серверов, которые пересчитываются каждые 5 минут, а при всплесках трафика — мгновенно. Принцип функционирования интеграционной шины «1С:Шина» показан на рисунке 2.

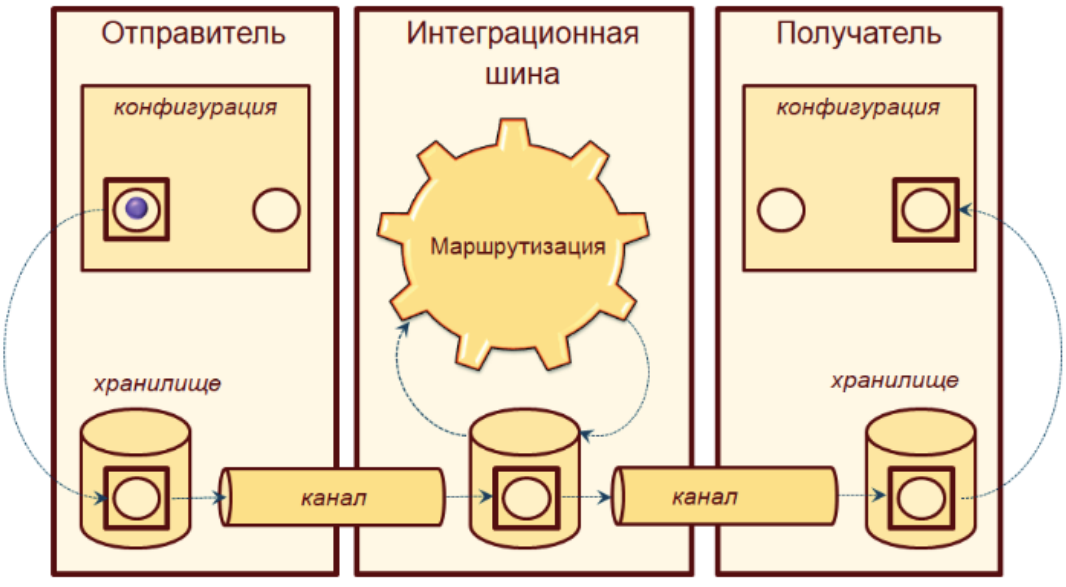

Рис. 2. – Принцип функционирования интеграционной шины «1С:Шина»

1С:Шина выполняет функции маршрутизатора данных между кластерами серверов, внешним балансировщиком и аналитическим центром для мониторинга нагрузки на узлы сети.

На рис.3 схематично представлена интеграция 1С:Шины в схему взаимодействия с внешним балансировщиком.

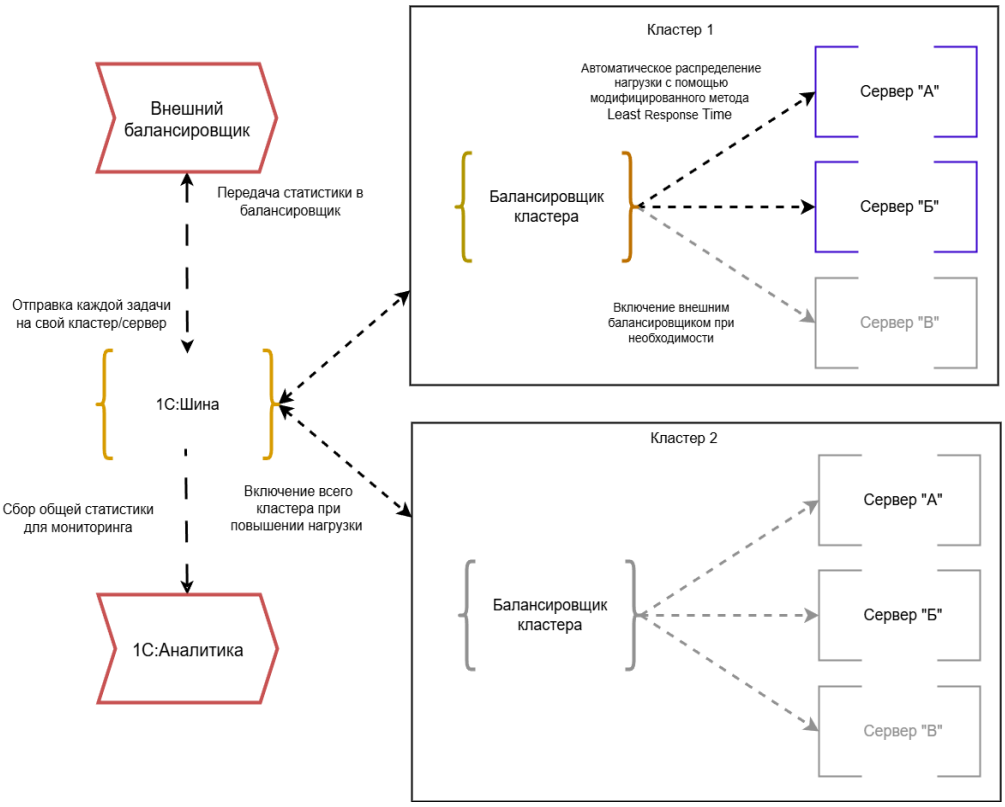

Рис. 3. – Схема интеграции 1С:Шины и сети кластеров серверов 1С

Для проведения эксперимента использовалась инфраструктура, имеющая 7 физических узлов. Для масштабирования сети до 150 узлов были использованы контейнеры Docker, которые помогают повысить масштабируемость. Их техническая конфигурация приведена в таблице 1.

Таблица №1

Конфигурация сервера с интеграционной шиной

Параметр	**Значение**
Операционная система	Linux (Ubuntu 24.0)
Оперативная память	8 GB RAM
Центральный процессор	4 ядра по 2.6 ГГц

Чтобы проверить, как генетический балансировщик ведёт себя при росте нагрузки, измеряли длительность его оптимизационного цикла и средний объём поколений до выхода на стабильное решение. Результаты представлены на рис.4.

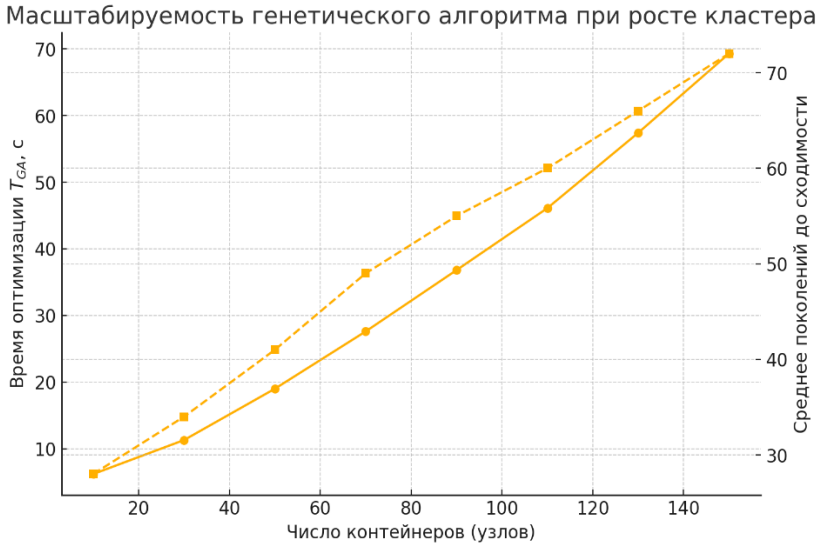

Рис. 4. – Масштабируемость предложенного генетического алгоритма балансировки

При 15-кратном увеличении числа узлов (с 10 до 150), время расчетов выросло примерно в 11 раз (с 6,2 с до 69,3). При практической эксплуатации закладывание времени в 5 минут является обоснованным даже для 200 узлов. Время в 5 минут берется из расчета, что требуются временные затраты на сбор статистики с кластеров (время сбора зависит от количества), время работы самого алгоритма, а также запас для резкого повышения входящего трафика запросов. Также на основе данных о среднем количестве поколений до сходимости можно утверждать, что алгоритм не показывает никаких аномалий, при увеличении количества узлов и способен быстро найти решение, до истечения условного время перерасчета в 5 минут.

Для сравнения эффективности алгоритмы было проведено несколько итераций эксперимента. Эмулировался входной поток 160 запросов в секунду, длительностью в 30 минут. Для сравнения были использованы

алгоритмы RR и LC с обновлением количества активных соединений каждые 30 секунд и рассматриваемый генетический алгоритм с пересчетом весов каждый 90 секунд. Результаты представлены в таблице 2.

Таблица №2

Результаты сравнительного анализа эффективности при стандартной нагрузке

Алгоритм балансировки	**P95*, с**	**SLAv**, %**	**Среднее число миграций / 90 с**	**Время на расчеты, % ЦПУ**
Round Robin	2.68	9.7	0	0
Least Connections	2.25	5.1	5	0.8
Ген. алгоритм	1.84	2.1	37	4.5

* - 95-й перцентиль времени ответа сервера. 95 % всех запросов завершаются быстрее этого значения, 5 % — медленнее.
** - доля запросов, превысивших договорное время обслуживания (в эксперименте равное 3 секундам).

На основе данных можно сделать следующие выводы:

- При переходе от RR к LC мы видим сокращение P95 на ~16%, показатель SLAv падает на ~ 45%. При этом стоимость перехода становится равной всего 0.8% процессорного времени. Такой подход может подойти для малых и средних сетей, где количество узлов не превышает 100 узлов.
- Генетический алгоритм позволяет снизить P95 относительно LC на 18%, а SLAv опустить до 2%. Ценой подобного повышения качества обслуживания запросов является повышение затрат процессорного времени до 4–5%, но при этом остается безопасное количество миграций. Переход на данный алгоритм логичен в высоконагруженных и критичных к задержкам сетях, когда важны обеспечение качества сервиса (QoS) и устойчивость при аварийных всплесках.

Для дальнейших проверок устойчивости алгоритма был проведен эксперимент, в котором поток запросов стал неравномерным. В течение

первых четырёх минут система обслуживала поток интенсивностью 160 запросов в секунду; затем, на три минуты, интенсивность была увеличена на 60 % до 256 запросов в секунду, после чего в оставшиеся три минуты поток снизили до 200 запросов в секунду. Результаты эксперимента приведены в таблице 3.

Таблица №3

Результаты сравнительного анализа эффективности при повышенной нагрузке

Алгоритм балансировки	**P95*, с**	**SLAv**, %**	**Среднее число миграций / 90 с**	**Время на расчеты, % ЦПУ**
Round Robin	4.01	22.3	0	0
Least Connections	2.78	11.4	10	1.1
Ген. алгоритм	2.26	4.8	52	5.8

* - 95-й перцентиль времени ответа сервера. 95 % всех запросов завершаются быстрее этого значения, 5 % — медленнее.
** - доля запросов, превысивших договорное время обслуживания (в эксперименте равное 3 секундам).

На основе данных можно сделать следующие выводы:

- Даже при приросте нагрузки на 60% самым качественным остается генетический алгоритм, при пиковой нагрузке повышая P95 всего до 2.26 секунд, оставаясь в запасе на 40% относительно RR.
- Генетический алгоритм демонстрирует линейное повышение количества миграций и затрачиваемого времени ЦПУ.
- RR – подходит только при низкой нагрузке, так как в реальной ситуации SLAv достигает 25% от всех запросов.
- LC – компромисс для средних сетей, обеспечивающий куда более высокий QoS относительно RR.
- Генетический алгоритм – устойчивый к резким колебаниям трафика, дающий максимальный выигрыш по качеству обслуживания и требующий умеренные 5–6% ЦПУ. Рекомендуется его использовать там, где

стоимость времени на вычисления меньше, чем стоимость превышения SLAv.

Заключение

В представленной работе разработан и исследован гибридный подход к распределению вычислительной нагрузки, объединяющий традиционные детерминированные стратегии балансировки с эволюционным поиском на основе генетического алгоритма. Практическая реализуемость методики подтверждена опытным прототипом, полностью развёрнутым на отечественной платформе (кластер серверов 1С + контейнерная виртуализация).

Интеграция эволюционной составляющей обеспечила качественное совершенствование механизма диспетчеризации: вместо равномерного «кругового» распределения запросы теперь перенаправляются с учётом текущей загрузки узлов, профиля поступающих задач и целевых нормативов по задержке обслуживания. В результате система демонстрирует повышенную устойчивость к кратковременным повышенным нагрузкам, а потребность в избыточном резервировании мощностей снижается.

Таким образом, полученные результаты подтверждают целесообразность использования эволюционных алгоритмов в качестве надстройки над классическими схемами, не требующей значительных инфраструктурных затрат, а также обеспечивающей улучшение эксплуатационных характеристик распределённых вычислительных комплексов.

Литература

1. Nagelli A., Yadav N. K. Efficiency unveiled: Comparative analysis of load balancing algorithms in cloud environments // International Journal of Information Technology and Management. 2024. V. 18. URL: doi.org/10.29070/pb64mp50.

2. Пономаренко Д.Н. Обзор алгоритмов балансировки нагрузки в облачных вычислительных системах // Молодой ученый. 2024. № 12 (511). С. 21-30.

3. Harjanti T. W., Setiyani H., Trianto J. Load balancing analysis using round-robin and least-connection algorithms for Server Service Response Time //Applied Technology and Computing Science Journal. 2022. V. 5. №2. pp. 119-128.

4. Katangur A., Akkaladevi S., Vivekanandhan S. Priority weighted round robin algorithm for load balancing in the cloud // 2022 IEEE 7th international conference on smart cloud (SmartCloud). IEEE, 2022. pp. 230-235.

5. Hou W., Meng L., Ke X., Zhong L. Dynamic load balancing algorithm based on optimal matching of weighted bipartite graph // IEEE Access. 2022. V. 10. pp. 127225-127236.

6. Tang X., Ding Y., Lei J., Yang H., Song Y. Dynamic load balancing method based on optimal complete matching of weighted bipartite graph for simulation tasks in multi-energy system digital twin applications // Energy Reports. 2022. V. 8. pp. 1423-1431.

7. Mayanda D., Amaliah A.R., Raharja M.R.A., Nurbojatmiko N. Load Balancing Techniques for Server Clustering in Cloud Environment: Systematic Literature Review // Journal of Renewable Energy, Electrical, and Computer Engineering. 2024. V. 4. №2. pp. 173-179.

8. Devi N., Dalal S., Solanki K., Dalal S., Lilhore U.K., Simaiya S., Nuristani N. A systematic literature review for load balancing and task scheduling techniques in cloud computing //Artificial Intelligence Review. 2024. V. 57. №10. pp. 276.

9. Ranjini P.S., Hemamalini T., Senthilvel A.N. Enhancing load balancing efficiency in distributed systems through linear programming techniques in WSCLB // International Journal of Computational and Experimental Science and Engineering. 2023. V. 9. URL: doi.org/10.22399/ijcesen.1311.

10. Khan A.R. Dynamic load balancing in cloud computing: optimized RL-based clustering with multi-objective optimized task scheduling // Processes. 2024. V. 12. №3. URL: doi.org/10.3390/pr12030519.

11. Zhou J., Lilhore U.K., Hai T., Simaiya S. C omparative analysis of metaheuristic load balancing algorithms for efficient load balancing in cloud computing // Journal of cloud computing. 2023. V. 12. №1. URL: doi.org/10.1186/s13677-023-00453-3.

12. Diwaker C., Miglani N. A Two-Decade Analysis of Load Balancing in Cloud Computing: Implications for Educational System and Future Directions // Journal of Computer Assisted Learning. 2025. V. 41. №3. URL: doi.org/10.1111/jcal.70042.

13. Jena U.K., Das P.K., Kabat M.R. Hybridization of meta-heuristic algorithm for load balancing in cloud computing environment // Journal of King Saud University-Computer and Information Sciences. 2022. V. 34. №6. pp. 2332-2342.

14. Latchoumi T.P., Parthiban L. Quasi oppositional dragonfly algorithm for load balancing in cloud computing environment // Wireless Personal Communications. 2022. V. 122. №3. pp. 2639-2656.

15. Adil M., Song H., Ali J., Jan M.A., Attique M., Abbas S., Farouk A. Enhanced-AODV: A robust three phase priority-based traffic load balancing scheme for Internet of Things // IEEE Internet of Things Journal. 2021. V. 9. №16. pp. 14426-14437.

16. Kathole A.B., Singh V.K., Goyal A., Kant S., Savyanavar A.S., Ubale S.A., Jain P., Islam M.T. Novel load balancing mechanism for cloud networks

using dilated and attention-based federated learning with Coati Optimization // Scientific Reports. 2025. V. 15. №. 1. URL: doi.org/10.1038/s41598-025-99559-8.

17. Annie Prnima Princess G., Radhamani A.S. A hybrid meta-heuristic for optimal load balancing in cloud computing // Journal of grid computing. 2021. V. 19. №2. URL: doi.org/10.1007/s10723-021-09560-4.

18. Lilhore U.K., Simaiya S., Prajapati Y.N., Rai A.K., Ghith E.S., Tlija M., Lamoudan T., Abdelhamid A.A. A multi-objective approach to load balancing in cloud environments integrating ACO and WWO techniques // Scientific Reports. 2025. V. 15. №1. URL: doi.org/10.1038/s41598-025-96364-1.

19. Ben Halima A., Yzzogh H., Benaboud H. Predictive Load Balancing in Cloud Computing: A Comparative Study // Proceedings of the 7th International Conference on Networking, Intelligent Systems and Security. 2024. C. 1-6.

20. Geetha P., Vivekanandan S.J., Yogitha R., Jeyalakshmi M.S. Optimal load balancing in cloud: Introduction to hybrid optimization algorithm //Expert Systems with Applications. 2024. V. 237. URL: doi.org/10.1016/j.eswa.2023.121450.

21. Zhou J., Lilhore U.K., Hai T., Simaiya S. Comparative analysis of metaheuristic load balancing algorithms for efficient load balancing in cloud computing // Journal of cloud computing. 2023. V. 12. №1. URL: doi.org/10.1186/s13677-023-00453-3.

22. Rutten D., Mukherjee D. Load balancing under strict compatibility constraints // Mathematics of Operations Research. 2023. V. 48. №. 1. pp. 227-256.

23. Goldsztajn D., Borst S.C., Van Leeuwaarden J.S.H. Server saturation in skewed networks // Proceedings of the ACM on Measurement and Analysis of Computing Systems. 2024. V. 8. №2. pp. 1-37.

References

1. Nagelli A., Yadav N. K. International Journal of Information Technology and Management. 2024. V. 18. URL: doi.org/10.29070/pb64mp50.

2. Ponomarenko D.N. Molodoj uchenyj. 2024. № 12 (511). pp. 21-30.

3. Harjanti T. W., Setiyani H., Trianto J. Applied Technology and Computing Science Journal. 2022. V. 5. №2. pp. 119-128.

4. Katangur A., Akkaladevi S., 2022 IEEE 7th international conference on smart cloud (SmartCloud). IEEE, 2022. pp. 230-235.

5. Hou W., Meng L., Ke X., Zhong L. IEEE Access. 2022. V. 10. pp. 127225-127236.

6. Tang X., Ding Y., Lei J., Yang H., Song Y. Energy Reports. 2022. V. 8. pp. 1423-1431.

7. Mayanda D., Amaliah A.R., Raharja M.R.A., Nurbojatmiko N. Journal of Renewable Energy, Electrical, and Computer Engineering. 2024. V. 4. №2. pp. 173-179.

8. Devi N., Dalal S., Solanki K., Dalal S., Lilhore U.K., Simaiya S., Nuristani N. Artificial Intelligence Review. 2024. V. 57. №10. pp. 276.

9. Ranjini P.S., Hemamalini T., Senthilvel A.N. International Journal of Computational and Experimental Science and Engineering. 2023. V. 9. URL: doi.org/10.22399/ijcesen.1311.

10. Khan A.R. Processes. 2024. V. 12. №3. URL: doi.org/10.3390/pr12030519.

11. Zhou J., Lilhore U.K., Hai T., Simaiya S. Journal of cloud computing. 2023. V. 12. №1. URL: doi.org/10.1186/s13677-023-00453-3.

12. Diwaker C., Miglani N. Journal of Computer Assisted Learning. 2025. V. 41. №3. URL: doi.org/10.1111/jcal.70042.

13. Jena U.K., Das P.K., Kabat M.R. Journal of King Saud University-Computer and Information Sciences. 2022. V. 34. №6. pp. 2332-2342.

14. Latchoumi T.P., Parthiban L. Wireless Personal Communications. 2022. V. 122. №3. pp. 2639-2656.

15. Adil M., Song H., Ali J., Jan M.A., Attique M., Abbas S., Farouk A. IEEE Internet of Things Journal. 2021. V. 9. №16. pp. 14426-14437.

16. Kathole A.B., Singh V.K., Goyal A., Kant S., Savyanavar A.S., Ubale S.A., Jain P., Islam M.T. Scientific Reports. 2025. V. 15. №. 1. URL: doi.org/10.1038/s41598-025-99559-8.

17. Annie Prnima Princess G., Radhamani A.S. Journal of grid computing. 2021. V. 19. №2. URL: doi.org/10.1007/s10723-021-09560-4.

18. Lilhore U.K., Simaiya S., Prajapati Y.N., Rai A.K., Ghith E.S., Tlija M., Lamoudan T., Abdelhamid A.A. Scientific Reports. 2025. V. 15. №1. URL: doi.org/10.1038/s41598-025-96364-1.

19. Ben Halima A., Yzzogh H., Benaboud H. Proceedings of the 7th International Conference on Networking, Intelligent Systems and Security. 2024. C. 1-6.

20. Geetha P., Vivekanandan S.J., Yogitha R., Jeyalakshmi M.S. Expert Systems with Applications. 2024. V. 237. URL: doi.org/10.1016/j.eswa.2023.121450.

21. Zhou J., Lilhore U.K., Hai T., Simaiya S. Journal of cloud computing. 2023. V. 12. №1. URL: doi.org/10.1186/s13677-023-00453-3.

22. Rutten D., Mukherjee D. Mathematics of Operations Research. 2023. V. 48. №. 1. pp. 227-256.

23. Goldsztajn D., Borst S.C., Van Leeuwaarden J.S.H. Proceedings of the ACM on Measurement and Analysis of Computing Systems. 2024. V. 8. №2. pp. 1-37.

Дата поступления: 3.06.2025

Дата публикации: 25.07.2025