ИПМ им.М.В.Келдыша РАН • Электронная библиотека

Препринты ИПМ • Препринт № 37 за 2025 г.

ISSN 2071-2898 (Print)
ISSN 2071-2901 (Online)

В.А. Судаков, Ю.П. Титов, П.М. Иванова, Т.В. Сивакова

Оптимизация следования параметров модели в вычислительном кластере асинхронной модификацией метода муравьиных колоний

Рекомендуемая форма библиографической ссылки: Оптимизация следования параметров модели в вычислительном кластере асинхронной модификацией метода муравьиных колоний / В.А. Судаков [и др.] // Препринты ИПМ им. М.В.Келдыша. 2025. № 37. 18 с. EDN: FQWAMZ
https://library.keldysh.ru/preprint.asp?id=2025-37

О р д е н а Л е н и н а
ИНСТИТУТ ПРИКЛАДНОЙ МАТЕМАТИКИ
имени М.В. Келдыша
Р о с с и й с к о й а к а д е м и и н а у к

В.А. Судаков, Ю.П. Титов, П.М. Иванова, Т.В. Сивакова

Оптимизация следования параметров модели в вычислительном кластере асинхронной модификацией метода муравьиных колоний

Москва — 2025

Судаков В.А., Титов Ю.П., Иванова П.М., Сивакова Т.В.

Оптимизация следования параметров модели в вычислительном кластере асинхронной модификацией метода муравьиных колоний

Работа посвящена параллельной модификации метода муравьиных колоний для задачи параметрической оптимизации с целью сохранения вычислительных ресурсов кластера. Рассматривается подход разделения метода муравьиных колоний на поток параллельного поиска путей и поток отправления значений параметров в модель на отдельной вычислительной машине. Предлагается введение копии графа для непрерывного поиска агентами пути без блокировок на процесс обновления феромонов на графе. Для взаимодействия с вычислительным кластером предлагается протокол взаимодействия с возможностью поддержки многопакетной отправки данных.

Ключевые слова: метод муравьиных колоний, перебор значений параметров, параллельные вычисления, многоэкстремальность, кластер

Vladimir Anatolievich Sudakov, Yuri Pavlovich Titov, Ivanova Polina Mihailovna, Tatiana Vladimirovna Sivakova

Optimization of following model parameters in a computing cluster by asynchronous modification of the ant colony method

The paper is devoted to the parallel modification of the ant colony method for the parametric optimization problem in order to conserve cluster computing resources. The approach of dividing the ant colony method into a stream of parallel pathfinding and a stream of sending parameter values to the model on a separate computer is considered. It is proposed to introduce a copy of the graph for continuous path search by agents without blocking the process of updating pheromones on the graph. For interaction with the computing cluster, an interaction protocol is proposed, with the ability to support multi-packet data sending.

Key words: ant colony method, iteration of parameter values, parallel computing, multiextremeality, cluster

Введение

В настоящее время развитие вычислительной техники позволяет повысить вычислительную сложность задач, при этом время, затрачиваемое на их решение, компенсируется мощностью вычислительной машины. С ростом объемов обрабатываемых данных и увеличением сложности вычислительных задач сложных математических моделей, как правило, выносятся вычисления на отдельные высокопроизводительные кластеры.

В случае решения оптимизационных задач такие модели требуют многократного повторения расчетов на различных наборах входных данных. Ввод наборов параметров по одному приводит к тому, что после получения результатов расчетов модели пользователь должен проанализировать результаты и в случае, если он оказался неподходящим, вручную подкорректировать значения параметров и снова отправить их на вычислительный кластер. Такой подход занимает много времени и требует полного участия пользователя в процессе расчетов.

Современные вычислительные кластеры позволяют выполнять отправку всех возможных наборов значений, что позволило пользователю не проверять результат расчета каждого набора, а просто получить на выходе программы один набор с оптимальным результатом. Однако такой подход затрачивает слишком много ресурсов вычислительной машины и требует большого времени ожидания получения результата. Кроме того, если речь идет о решении задач многоэкстремальной оптимизации, то полученный набор может быть не всегда оптимальным, поскольку такие задачи могут иметь несколько решений.

В связи с этим стоит проблема оптимизации порядка следования параметров сохранения вычислительных ресурсов кластера. На данный момент существуют различные решения данной проблемы. Среди них - случайный поиск, байесовская оптимизация, эволюционные методы и т.д.

Наиболее близкими являются системы, основанные на байесовском оптимизаторе, например, IBM Bayesian Optimization Accelerator (BOA). Решения на основе BOA используют метаэвристики для создания множества гипотез. Данные метаэвристики являются коммерчески закрытыми и не подлежат анализу, в отличие от открытости и легкости анализа метода муравьиных колоний [1]. Кроме того, изменения технических процессов от ручного вычисления и поиска оптимальных структур к автоматизированному поиску этих структур за счет возрастающей вычислительной мощности требуют новых решений.

1. Модели и методы

Параллельный ACO

Задачи анализа или создания расчетных моделей часто подразумевают под собой оптимизацию параметров. Различают одноэкстремальную и

многоэкстремальную оптимизацию. В случае одноэкстремальной оптимизации чаще всего используются такие методы оптимизации, как метод градиентного спуска, который позволяет реализовать наискорейший поиск параметров модели, дающих оптимальный результат. Однако в случае многоэкстремальных задач такие методы позволяют определить только локальный экстремум функции. Если же речь идет о глобальном экстремуме, то используются эвристические методы, которые совмещают в себе стохастический и направленный поиск и обладают высокой сходимостью.

Эвристические методы обладают хорошими возможностями для расширения и модификации. Модификации методов позволяют учитывать дополнительные особенности решаемых задач. Таким образом, модифицированные методы позволяют решать более широкий круг оптимизационных задач и иметь больше практического применения [2].

Метод муравьиных колоний (АСО) относится к роевым эвристическим алгоритмам. Традиционно данный метод используется для решения задачи коммивояжёра и задачи о назначениях [3].

Современные исследования АСО позволяют расширить сферу его применения до параметрической оптимизации, то есть применять его для решения задач поиска оптимальных наборов параметров целевой функции. АСО обладает хорошей сходимостью и, следовательно, находит оптимальные решения в любом случае. Однако из хорошей сходимости следует, что оптимизационный метод может уйти в локальный оптимум, при этом не найти глобальный. Данная проблема может решаться двумя способами: во-первых – хитрые модификации, например, метод градиентного спуска с сохранением итераций, который позволяет проскакивать локальные минимумы, а во-вторых – мультистарт. Под мультистартом подразумевается множественный запуск метода из разных точек [4]. Кроме того, сходимость достаточно медленная, поскольку требует большого количества итераций. Уменьшения количества итераций можно добиться несколькими способами. Один из вариантов – начать из хорошей точки и уже вокруг нее искать глобальный оптимум. Однако такой способ подходит, только если известно поведение целевой функции. Если же поведение целевой функции неизвестно, тогда количество итераций можно сократить за счет параллелизма.

Таким образом, можно перейти к параллельному АСО. В общем случае задачу оптимизации можно разделить на поток подбора параметров методом АСО для целевой функции и поток расчета целевой функции. В таком случае можно рассмотреть 3 случая отношений времени выполнения этих процессов:

1. Время подбора параметров равно времени расчета целевой функции ($t_{пп}=t_{вцф}$). Данный случай считается эталонным.
2. Время подбора параметров больше времени расчета целевой функции ($t_{пп}>t_{вцф}$). В таком необходимо оптимизировать АСО для приближения к случаю 1.

3. Время подбора параметров меньше времени расчета целевой функции ($t_{пп}<t_{вцф}$). Для приближения к равенству времени работы процессов необходимо оптимизировать процесс расчета целевой функции.

Ускорение метода ACO можно осуществить двумя способами: модификацией метода и параллелизмом. Самими известными модификациями метода являются: элитарный ACO, системы муравьиных колоний (ACS) и MIN-MAX ACO.

Элитарный ACO был разработан в 1992 году и предназначен решить проблему медленной сходимости оригинального метода. Данная оптимизация вводит такое понятие, как элитные муравьи: данные агенты не только заносят феромоны в конце итерации на ребра выбранного ими пути, но и также добавляют дополнительные феромоны на ребра лучшего решения, найденного ранее [5]. Таким образом, быстрый рост феромонов на узлах, составляющих оптимальные пути, повышает вероятность выбора данных узлов на последующих итерациях, что позволяет ускорить сходимость метода в области оптимального решения.

Системы муравьиных колоний (ACS) – оптимизация метода муравьиных колоний, разработанная в 1996 году. Основной особенностью ACS является введение понятия эксплуатации. В отличие от обычных агентов, выбирающих путь в графе случайным образом, эксплуатирующие агенты выбирают в графе узел, обладающий наибольшей вероятностью выбора. При этом оптимизация позволяет настроить процентное соотношение агентов, ищущих по вероятностной формуле, и агентов, эксплуатирующих [6].

MIN-MAX оптимизация была разработана также в 1996 году. Данная модификация позволяет повысить производительность алгоритма ACO за счет комбинации использования лучших решений, найденных в ходе поиска, и эффективного механизма для предотвращения застоя в поиске на ранних итерациях. В MIN-MAX оптимизации несколько ключевых особенностей. Во-первых, в конце каждой итерации феромон добавляет только тот агент, который нашел лучшее решение либо в конце текущей итерации, либо за весь процесс выполнения алгоритма. Во-вторых, для избегания стагнации для каждого феромона задается интервал его возможных значений. Кроме того, первоначально феромоны инициализируются максимальным возможным значением, что позволяет ускорить процесс поиска на начальных итерациях [7].

Ускорение расчета целевой функции может быть реализовано двумя способами:

1. Параллельное вычисление целевой функции;
2. Промежуточное хранение данных.

Разделение задачи оптимизации на потоки подбора параметров ACO и расчета целевой функции позволяет разнести эти подзадачи на разные вычислительные машины, образующие кластер. При параллельном вычислении

целевой функции кластер работает в многопоточном режиме, а метод муравьиных колоний – в последовательном (рис. 1). За счет такого подхода время, затрачиваемое на расчет целевой функции, сокращается, что позволяет приблизить его к времени работы АСО.

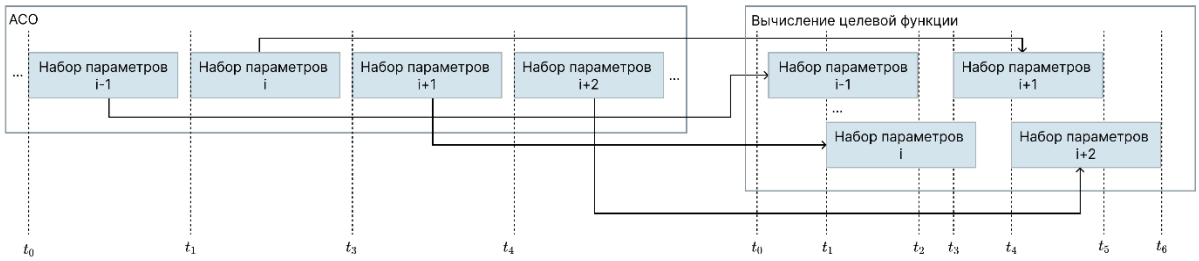


Рис. 1. Схема параллельной работы вычислительного кластера

Ускорение работы целевой функции через промежуточное хранение предполагает, что и АСО, и расчет целевой функции проходят в последовательном режиме. Однако агенты делятся на две группы: одна группа отправляет данные на расчет функции, а другая забирает результаты из промежуточного хранилища, в качестве которого могут использоваться как хэш-таблицы, так и базы данных (рис. 2).

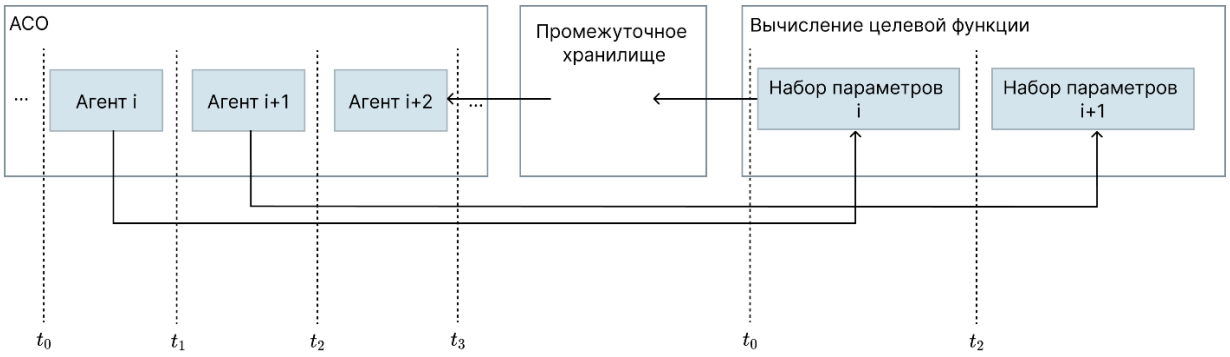


Рис. 2. Схема взаимодействия вычислительного кластера и АСО через промежуточное хранилище

На основании алгоритма синхронного модифицированного метода муравьиных колоний [1] можно построить схему жизненного цикла агента (рис. 3). Он состоит из следующих основных этапов: создание, выбор пути, отправление пути на вычислительный кластер, обработка результатов расчетов, изменение феромонов в графе и отправление данных пользователю и уничтожение. В случае, когда $t_{пп} > t_{вцф}$, чтобы избежать простаивания кластера, необходимо, чтобы объем поступающих данных превышал скорость их обработки на вычислительном кластере. Этого можно добиться, если заставить агентов искать значения параметров в графе одновременно с помощью асинхронной реализации алгоритма [8].

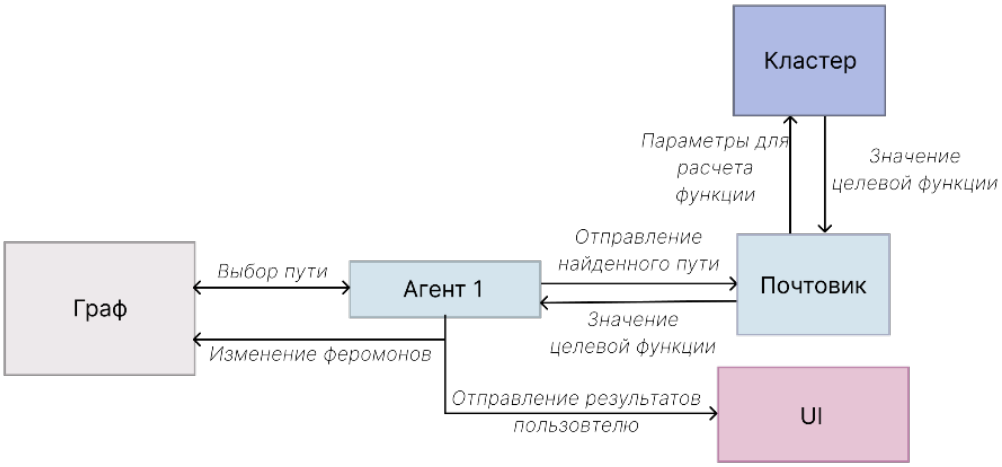


Рис. 3. Схема жизненного цикла агента

Чтобы обеспечить непрерывное отправление данных на кластер, необходимо реализовать 2 потока: первый поток осуществляет параллельный поиск агентами путей в графе и занесение найденных наборов в очередь, при этом каждый набор идентифицируется ID агентом. Второй поток осуществляет непрерывную проверку очереди, и в случае если она не пуста, то он выбирает из нее запись по принципу FIFO и отправляет ее через модуль связи на вычислительный кластер.

После того как модуль связи получает результаты расчета кластера, он отправляет их пользователю в клиентскую часть приложения и запускает процесс пересчета феромонов в копии графа. По сути, происходит решение классической задачи производителя-потребителя, которая требует, чтобы задача выбора путей в графе была мощнее, чем задача отправления данных на вычислительный кластер [9].

Обновление феромонов на графе без блокировки

В традиционном методе муравьиных колоний обновление феромонов на графе осуществляется после прохождения по графу группы агентов, однако такой подход блокирует выполнение на время обновления графа (рис. 4).

Как видно из рисунка, агенты осуществляют чтение графа, которое может осуществляться параллельно без блокировок, однако при записи обновленных феромонов на граф чтение блокируется, что приводит к тому, что следующая группа агентов может начать читать граф только после того, как обновится после прохождения прошлой группы. Данную проблему можно решить, используя механизмы синхронизации без блокировок, такие как RCU (Read-Copy-Update) [10].

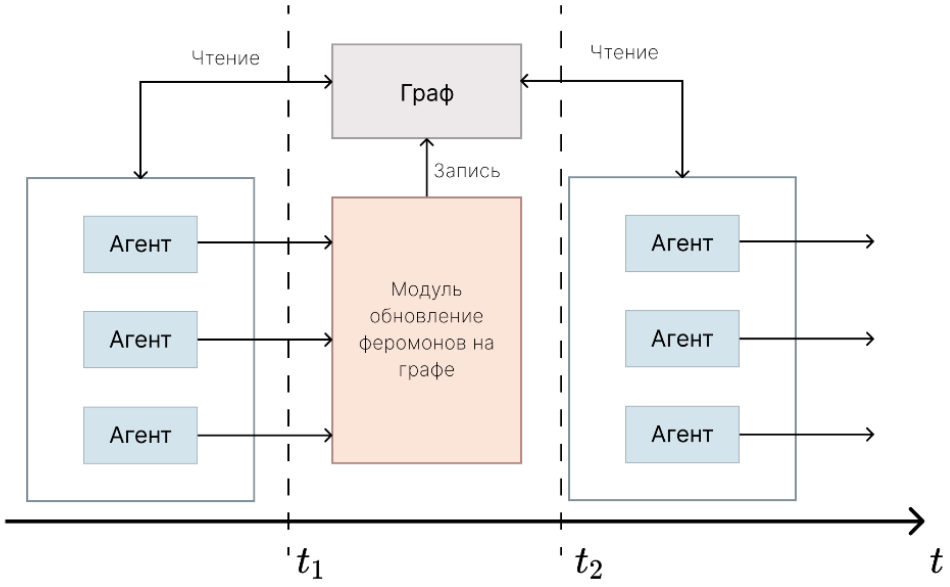


Рис. 4. Схема блокировки при обновлении графов

Согласно алгоритму RCU, вводится копия графа. После того как группа агентов осуществила параллельный поиск, она обновляет не основной граф, а его копию, в то время как следующая группа агентов ищет пути в основном графе. После обновления копии графа основной граф удаляется, и следующая группа агентов ищет феромоны на копии, которая становится основным графом, и обновляет новую копию. Таким образом осуществляется непрерывный цикл чтения, копирования и обновления.

Однако постоянное удаление и пересоздание копий графа приводит к сильным задержкам по времени. Для решения этой проблемы вводится копия графа, при этом основной граф не удаляется, агенты просто переключаются между копией и основным графом, который обновляется в момент, когда агенты читают копию с обновленными феромонами (рис. 5).

Для управления переключениями между графами вводится менеджер обращений, который следит за состояниями копии и основного графа. Если изменения вносились, то агент отправляется на обновленную копию графа, если же нет – на основной. При этом если копия графа отличается от основного, то менеджер запускает копирование изменений в основной граф.

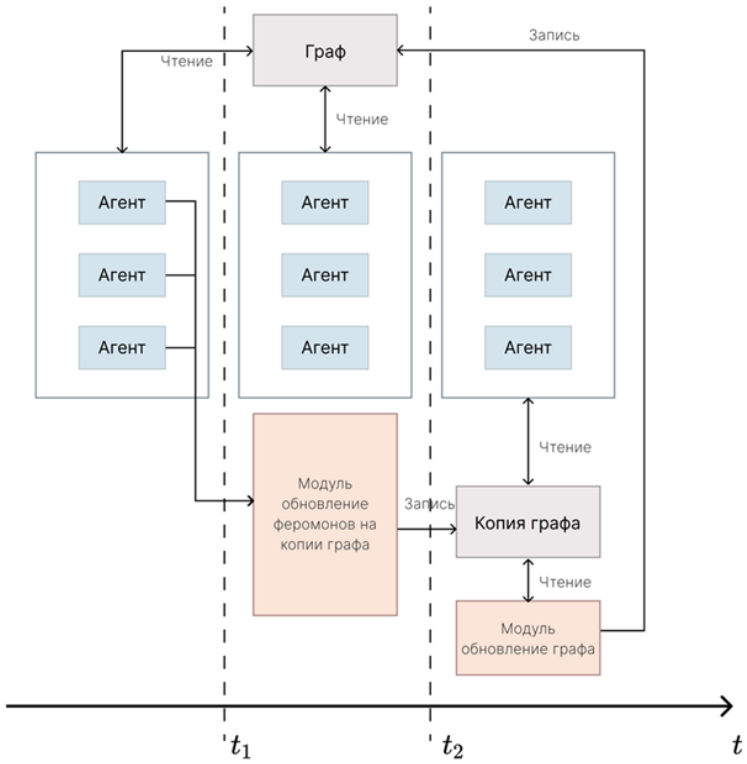


Рис. 5. Схема без блокировки при обновлении графов

Тогда итоговую схему можно представить согласно рисунку 6. Для экономии вычислительных ресурсов набор значений, отправляемых на него, должен быть уникальным. Для этого были введены хэш-таблицы. Когда агент находит очередной путь в графе, он проверяет, есть ли его хэш в таблице (то есть проводился ли расчет модели с заданными параметрами). Если хэш уже присутствует в таблице, то агент ищет новый уникальный набор. Если хэш отсутствует в таблице, то агент добавляет его и отправляет значения параметров в модель для расчета. В зависимости от объема входных данных выбирается способ хранения хэшей. Если объем входных данных не очень большой, то есть количество путей в графе не превышает 1 Гб памяти, то для хранения хэшей используются таблицы, хранящиеся в оперативной памяти. В противном случае подключается база данных. Такой подход позволяет хранить больше хэшей, чем в оперативной памяти, однако этот способ хранения существенно замедляет процесс проверки. Процесс поиска можно ускорить, если хранение будет осуществляться по принципу ключ-значение.

Так как агенты ищут уникальный путь параллельно, то и сверяют хэши путей они тоже параллельно. Хэш-таблица является критической секцией, то есть за один раз к ней может обратиться только 1 агент. После обращения агента секция блокируется, и новые агенты встают в очередь. Если разнести проверку хэша и его занесение в таблицу, то получаем еще одну возможную коллизию. Когда после проверки таблицы агент снова отправляется в очередь, то может получиться ситуация, когда перед ним на запись уже стоит агент с точно таким же хэшем. Для предотвращения этого стоит соединить проверку и

запись, то есть в случае если хэша в таблице нет, то агент должен сразу его записать. Однако из-за того, что хэш-таблицы являются критической секцией, агенты все рано тратят слишком много времени на проверку и запись. Чтобы решить данную проблему, стоит очищать очередь. То есть когда очередной агент отправляется на запись, стартует параллельная проверка очереди агентов. Если в очереди есть агенты с точно таким же хэшем, то они отправляются на перевыбор пути еще до того, как до них дойдет очередь.

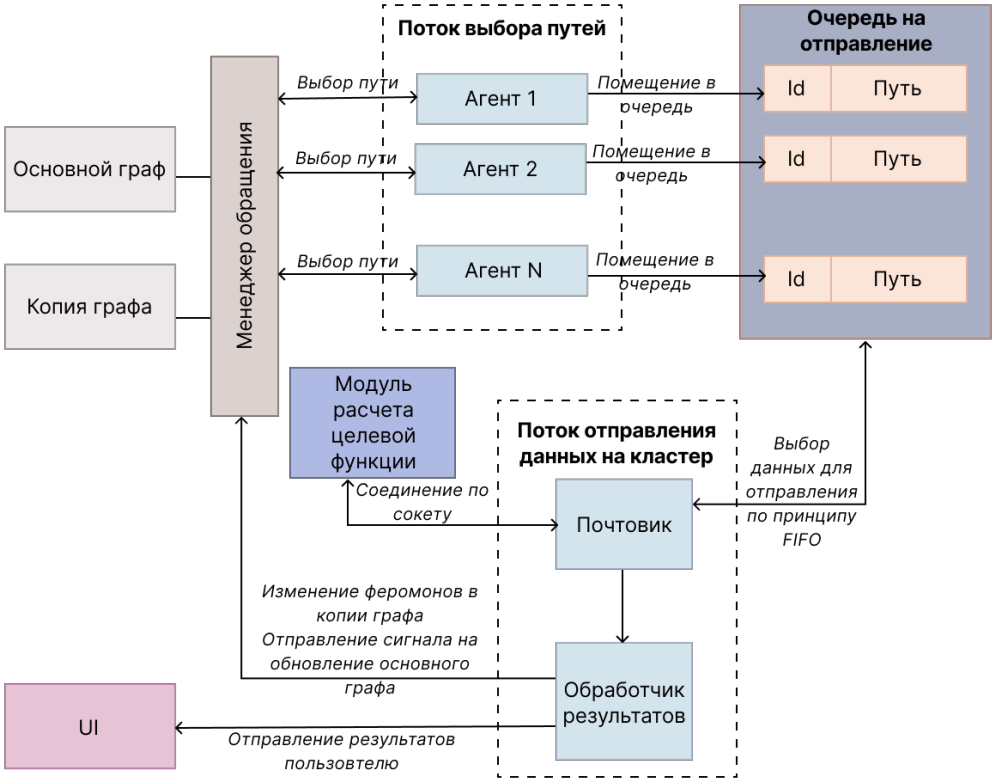


Рис. 6. Схема основных модулей асинхронного алгоритма метода муравьиных колоний

Соединение с модулем расчета целевой функции

Для обеспечения переупорядочивания параметров необходимо обеспечить взаимодействие между модулем расчета целевой функции и модулем переупорядочивания. Поскольку модуль может не только быть интегрированным как часть приложения, работающего на том же компьютере, что и модуль расчета целевой функции, но и располагаться на отдельной вычислительной машине, то было принято решение осуществлять их взаимодействие через сокеты на основе протокола TCP, что обеспечивает гарантированное получение набора значений параметров для расчета целевой функции. Для сохранения мощностей принято решение сохранить одно соединение на весь процесс общения модели расчета целевой функции и модуля переупорядочивания параметров.

На модуль расчета целевой функции должны передаваться запросы двух типов:

- сообщения с наборами значений для расчетов;
- сообщения-сигналы для сбора статистики работы кластера.

Для взаимодействия модулей был разработан специальный протокол, работающий по аналогии с протоколом HTTP. Данные передаются в формате JSON, который имеет два основных ключа:

- header – содержит строку с типом запроса;
- body – тело запроса, содержащее передаваемый путь агента или команду для сбора статистики в формате json.

Для повышения производительности модуля расчета целевой функции наборы значений параметров в модуль передаются пакетами по N штук.

2. Эксперимент

Сбор статистики работы модуля осуществляется для оценки времени работы алгоритма, оценки работы отдельных модулей алгоритма, а также оценки производительности модуля расчета целевой функции.

Так как алгоритм работает с дискретными случайными величинами, то для сбора статистики необходимо осуществить множество его прогонов и провести оценку математического ожидания (1) и дисперсии (2).

$$M_{t+1} = \frac{M_t \cdot n + x_i}{n+1}. \quad (1)$$

$$D = \frac{\sum (x_i - m_x)^2}{n+1}. \quad (2)$$

Для собранных временных характеристик производится оценка производительности работы вычислительного кластера, которая рассчитывается по формуле (3):

$$P = \frac{100\% * T_{work}}{T_{all}}. \quad (3)$$

где: T_{work} – время обработки полезной нагрузки; T_{all} – все время работы вычислительного кластера.

Для анализа работы разработанного модуля для переупорядочивания агентов асинхронным метом муравьиных колоний и производительности модуля расчета целевой функции были собраны следующие статистические данные:

- временные характеристики синхронного метода муравьиных колоний;
- временные характеристики модуля поиска путей, модуля переупорядочивания, общего времени работы асинхронной модификации метода муравьиных колоний с пересозданием агентов и без него, с изменением потоков агентов в диапазоне от 1 до 32 и различными типами организации очереди данных на отправление;

- временные характеристики модуля поиска путей, модуля переупорядочивания, общего времени работы асинхронной модификации метода муравьиных колоний при многопакетной отправке данных на вычислительный кластер с размерностью пакетов от 1 до 16;
- временные характеристики работы модуля расчета целевой функции и работы при обработке полезной нагрузки для каждой модификации асинхронного метода муравьиных колоний.

Сравнение синхронного АСО и его параллельной модификации

Как видно из графиков (рис. 7), процент полезной нагрузки вычислительного кластера примерно составляет 1,3% от общего времени работы, в то время как для асинхронного метода этот параметр составляет 79,8% (больше в 59,6 раз). Среднее время работы синхронного метода составляет 422,75 секунд, а асинхронного – 109,08 секунд, что в 3,87 раз быстрее. Таким образом, можно сделать вывод, что асинхронный метод муравьиных колоний не в самой эффективной реализации лучше синхронного и по полезной нагрузке кластера, и по времени работы программы в целом.

a) b)

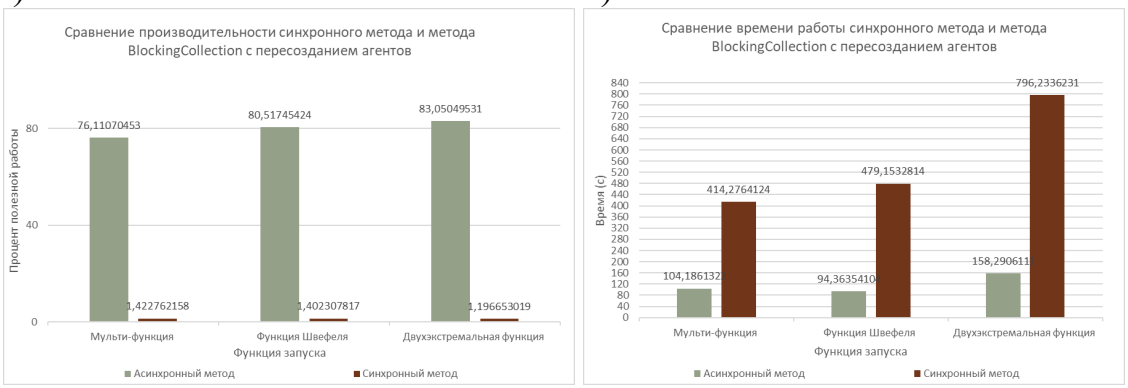


Рис. 7. График соотношения полезной нагрузки кластера при синхронном и асинхронном алгоритмах (a), график соотношения общего времени работы при синхронном и асинхронном алгоритмах (b)

Также из графиков видно, что целевая функция на кластере влияет на производительность модуля незначительно для обоих методов. Однако для синхронного метода изменение целевой функции сильно влияет на время работы программы (при запуске модуля с двухэкстремальной функцией время работы увеличивается почти в 2 раза).

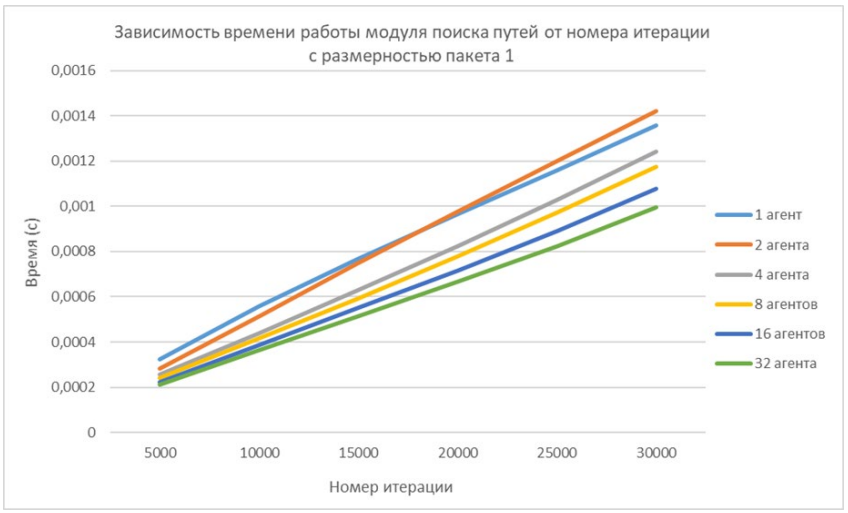
Оценка временных характеристик для сравнения реализации с пересозданием агентов и без него

На рисунке 8 представлены графики зависимости времени работы модуля поиска путей от номера итерации с размерностью пакета 1 для реализации

очереди через BlockingCollection (встроенная организация очереди в язык разработки C#) с пересозданием агентов и через QueueOnCluster (реализована вручную для поставленной задачи) с пересозданием агентов. По вертикали отображается ось времени, затрачиваемого в среднем на итерацию одного агента (в секундах), по горизонтали отображается номер итерации, на которой замеряется время. Шаг замера составляет 5000. Для замера времени количество агентов варьировалось от 1 до 32.

Как видно из графиков, изменение количества потоков агентов в среднем не сильно влияет на время, затрачиваемое на одну итерацию. При этом наблюдается ситуация, когда с увеличением итерации время, затрачиваемое на каждую итерацию, также увеличивается. Это объясняется тем, что с прохождением каждой новой итерации увеличивается количество добавлений в хэш, тем самым вероятность попадания также увеличивается, что приводит к тому, что агентам нужно чаще искать новые пути в графе.

a)



b)

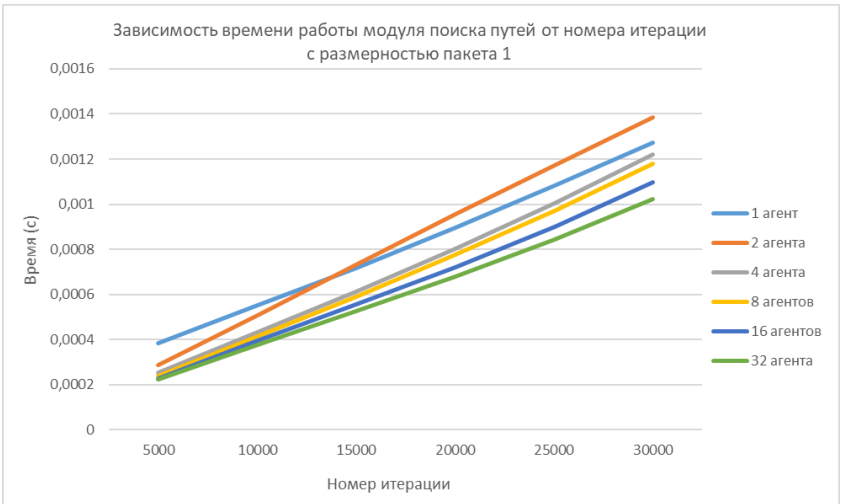
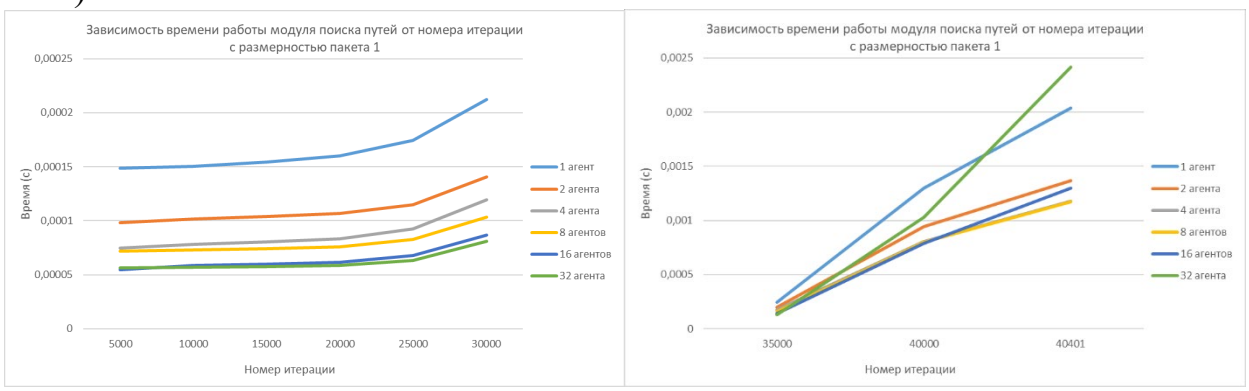


Рис. 8. График зависимости времени работы модуля поиска путей от номера итерации с размерностью пакета 1 для реализации очереди через BlockingCollection с пересозданием агентов (a) и QueueOnCluster с пересозданием агентов (b)

По аналогии построены графики для случаев без пересоздания агентов (рис. 9). Как и в случае с графиками выше, наблюдается увеличение времени, в среднем затрачиваемого на одну итерацию, однако в реализациях без пересоздания агентов наблюдается более медленное увеличение времени. Данную ситуацию можно объяснить тем, что смена ID агента и его поиск по истории занимают значительно меньше времени, чем удаление и пересоздание всей группы.

a)



b)

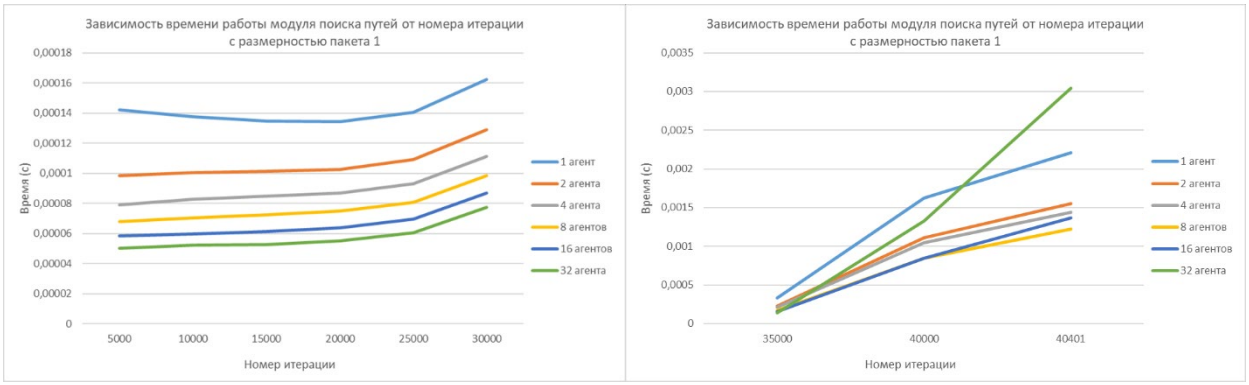


Рис. 9. График зависимости времени работы модуля поиска путей от номера итерации с размерностью пакета 1 для реализации очереди через BlockingCollection без пересоздания агентов (a) и QueueOnCluster без пересоздания агентов (b)

Исходя из всего вышесказанного, можно сделать вывод, что общее время работы программы уменьшается с увеличением количества потоков агентов. При этом организация очереди через QueueOnCluster без пересоздания агентов дает больший выигрыш во времени и возможность дальнейшего увеличения потоков агентов.

На рисунках 10 и 11 представлены графики зависимости времени, затрачиваемого на отправление данных на одной итерации, от числа итераций.

a) b)

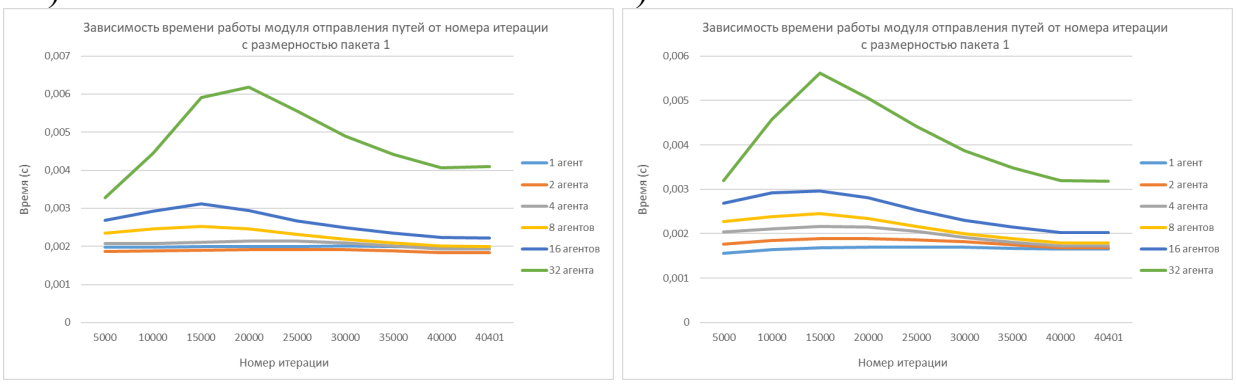


Рис. 10. График зависимости времени работы модуля отправления путей от номера итерации с размерностью пакета 1 для реализации очереди через BlockingCollection без пересоздания агентов (a) и QueueOnCluster без пересоздания агентов (b)

Из графика видно, что для 32 агентов среднее время, затрачиваемое на одну итерацию при отправлении, растет до 15000–20000 итераций, а затем начинает падать. Данную ситуацию можно объяснить тем, что очередь заполняется медленней, что приводит к тому, что потоку отправления приходится ждать дольше.

При этом независимо от способа организации очереди начиная с какой-то итерации среднее время, затрачиваемое на одну итерацию, начинает падать. Данная ситуация обусловлена тем, что при постепенном заполнении очереди потоку отправления не нужно ждать, пока в очереди появятся данные.

Время работы модулей без пересоздания агентов быстрее почти в 2 раза, независимо от способа организации очереди. Данное явление наблюдается из-за того, что на каждой итерации параллельного поиска агенты меняют ID в памяти, не переопределяя ее, что сокращает процесс инициализации итерации.

a) b)

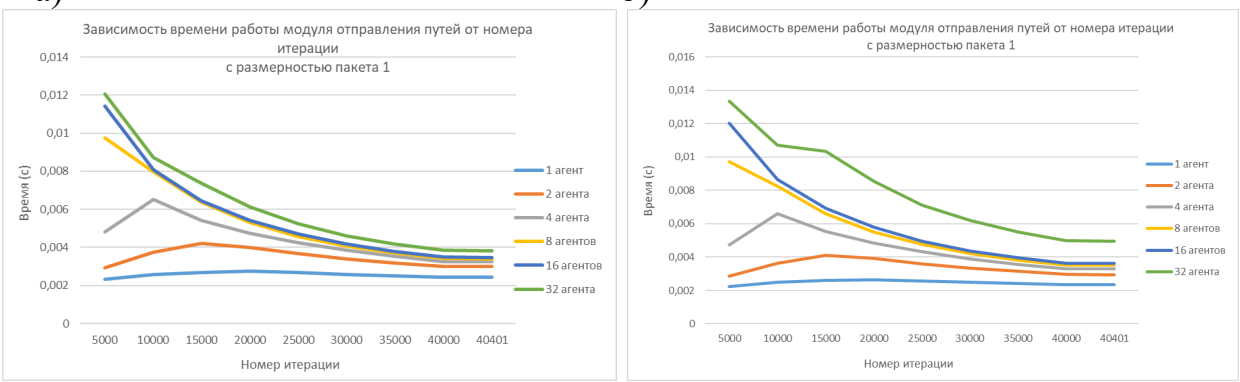


Рис. 11. График зависимости времени работы модуля отправления путей от номера итерации с размерностью пакета 1 для реализации очереди через BlockingCollection с пересозданием агентов (a) и QueueOnCluster с пересозданием агентов (b)

Для оценки временных характеристик вычислительного кластера также были построены соответствующие графики.

Из графиков на рисунке 12 видно, что с увеличением потоков агентов производительность модуля расчета модели увеличивается. При этом максимальная производительность достигается на 15000–20000 итераций. Данная ситуация может объясняться тем, что в этот промежуток агенты имеют меньший процент попадания хэш, кроме того, очередь на отправления достаточно заполнена, что позволяет максимально увеличить поток поступающих данных.

a) b)

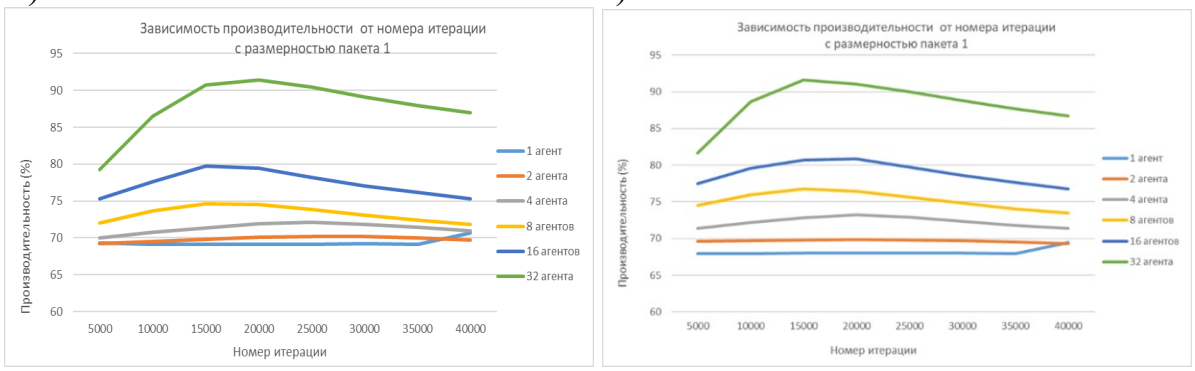


Рис. 12. График зависимости производительности расчета модели от номера итерации с размерностью пакета 1 для реализации очереди через BlockingCollection без пересоздания агентов (a) и QueueOnCluster без пересоздания агентов (b)

a) b)

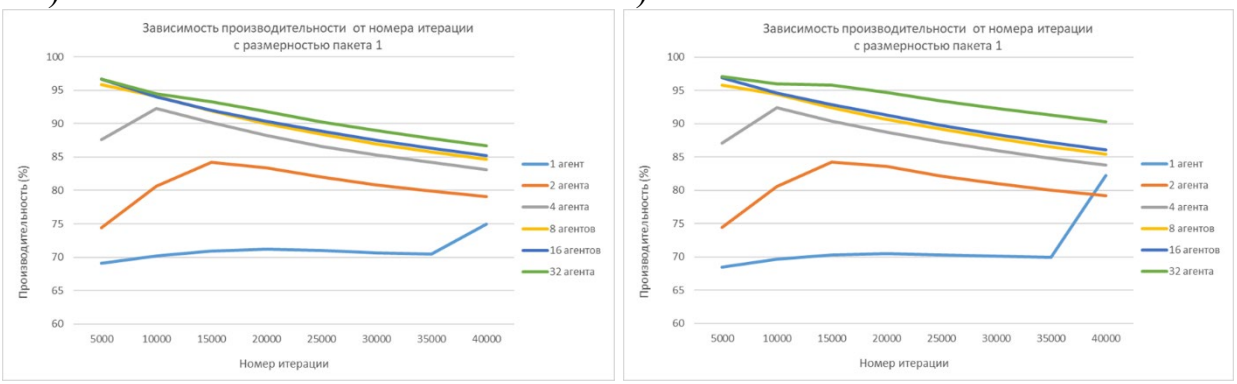


Рис. 13. График зависимости производительности расчета модели от номера итерации с размерностью пакета 1 для реализации очереди через BlockingCollection с пересозданием агентов (a) и QueueOnCluster с пересозданием агентов (b)

Как видно из графиков на рисунке 13, производительность с пересозданием агентов значительно выше, чем без пересоздания. Это связано с тем, что пересоздание агентов требует много времени, что приводит к тому, что на выполнение алгоритма в целом уходит больше времени, следовательно,

производительность растет. Поэтому можно сделать вывод, что если необходимо загрузить вычислительный кластер, тогда следует выбирать реализацию с большим временем работы. Если важна скорость выполнения алгоритма, то выбираются реализации без пересоздания агентов. Также можно сделать вывод, что независимо от способа реализации при задании количества агентов больше двух с увеличением количества итераций производительность падает. Если задать количество агентов равным 1, производительность на последних итерациях, наоборот, растет. Можно предположить, что данная ситуация связана со скоростью процесса наполнения очереди данными. С увеличением количества агентов очередь на отправление наполняется раньше (10000–20000 итераций), в то время как для базового разделения на потоки очередь достаточно наполняется только ближе к концу выполнения программы.

При увеличении размерности пакетов максимальная нагрузка сдвигается ближе к последним итерациям. В случае, когда агенты не пересоздаются, зависимость производительности от количества агентов в потоке почти не изменяется. В то же время, когда агенты пересоздаются, количество агентов в потоке компенсируется размерностью пакета.

3. Заключение

По результатам работы программного обеспечения удалось показать, что параллельный метод муравьиных колоний позволяет не только повысить производительность модуля расчета целевой функции, но и сократить общее время работы программы. Исследование на тестовых функциях показало, что при базовом разделении на потоки выигрыш в производительности составляет от 70% до 80%. При этом время работы алгоритма сокращается в 4-5 раз.

Вариации размерностей пакетов и очередей дают возможность адаптировать алгоритм под ситуации, когда требуются быстрые вычисления с низкой производительностью и высокая производительность при более низкой скорости работы алгоритма.

Реализованный алгоритм планируется развивать с использованием MPI для расчетов более сложных задач на суперкомпьютерах. Необходимо оптимизировать алгоритм с учетом особенностей распределённых вычислений. Для этого требуется разделить параллельный поиск путей агентами в графе между процессами MPI. Предлагается разделить хэш-таблицы между процессами для возможности параллельного поиска и записи без единой очереди к критической секции. Также предлагается рассмотреть работу параллельного метода с добавлением в него элитных агентов.

Библиографический список

1. Судаков В.А., Титов Ю.П., Сивакова Т.В., Иванова П.М. Применение метода муравьиных колоний для поиска рациональных значений параметров технической системы // Препринты ИПМ им. М.В.Келдыша – 2023

– № 38. – С. 18 – URL: https://doi.org/10.20948/prepr-2023-38 https://library.keldysh.ru/preprint.asp?id=2023-38

2. Певнева А.Г., Калинкина М.Е., Методы оптимизации– СПб: Университет ИТМО, 2020. – 64 с.

3. Ant Algorithms for Discrete Optimization. Artificial Life 5, 137-172

4. Eiben A.E., Smith J.E. Introduction to Evolutionary Computing Second Edition Natural Computing Series // Springer, 2015 DOI:10.1007/978-3-662-05094-1

5. Elitist Ant System for Route Allocation Problem [Электронный ресурс] URL: https://wseas.us/e-library/conferences/2008/rhodes/aic/aic08.pdf

6. Ant Colony Optimization Artificial Ants as a Computational Intelligence Techniqu [Электронный ресурс] URL: https://kpfu.ru/staff_files/F2066012578/Ant_Colony_Optimization.pdf

7. MAX-MIN ant systemhttps [Электронный ресурс] URL: https://www.researchgate.net/publication/277284831_MAX-MIN_ant_system

8. Титов Ю.П., Судаков В.А. Модификация асинхронного метода муравьиных колоний для задач поиска рациональных решений параметрической задачи // Труды МАИ. – 2024 – № 135. – URL: https://trudymai.ru/published.php?ID=179694

9. Таненбаум Э., Бос Х. Современные операционные системы. 4-е изд. — СПб.: Питер, 2015. — 1120 с.:ил. — (Серия «Классика computer science»). ISBN 978-5-496-0139

10. Роберт Лав. Ядро Linux: описание процесса разработки, 3-е изд.: Пер. с англ. — М.: ООО «И.Д. Вильямс», 2013. — 496 с. : ил. — Парал. тит. англ.

Оглавление