УДК 004.896 *Научная статья* DOI: 10.18287/2223-9537-2025-15-3-418-435

Мультиагентный метод повышения адаптивности управления вычислительными ресурсами в реальном времени

© 2025, Ф.М. Кирьяков[1]✉, П.О. Скобелев[1,2]

[1] *Самарский государственный технический университет (СамГТУ), Самара, Россия*

[2] *Самарский федеральный исследовательский центр РАН (СамНЦ РАН), Самара, Россия*

Аннотация

Работа посвящена развитию мультиагентного метода для удовлетворения растущей потребности в вычислительных ресурсах за счёт повышения адаптивности и эффективности управления в реальном времени. На практике требуется иметь возможность оперативной и гибкой индивидуальноточечной адаптивной корректировки составленного расписания исполнения задач, чтобы обеспечить более высокую эффективность использования ресурсов. Рассматриваемый мультиагентный метод управления вычислительными ресурсами основан на ранее предложенной модели сети потребностей и возможностей, обеспечивающей скользящее адаптивное изменение расписания. Последовательность пошаговых атомарных модификаций в расписании ресурсов включает сдвиги задач на одном вычислительном ресурсе или вытеснение и перераспределение задач между ресурсами. Агент задачи рассчитывает оптимальный «патч» для глобальной эффективности системы, учитывая функции удовлетворённости всех затрагиваемых задач. Новым является механизм коллективного принятия решений через расчёт и согласование «патчей», обеспечивающий динамическую оптимизацию расписания без необходимости его полного перепланирования или переход к полностью распределённому решению, исключающему общую сцену данных мира агентов. Экспериментальное сравнение показало, что предложенный метод повышает эффективность системы на 25-30% по сравнению с неадаптивными управлением, не обладающим возможностью частичного пересмотра связей между агентами, сдвига и перераспределения задач по ресурсам. Разработанный метод позволяет повысить масштабируемость и отказоустойчивость систем, расширить его практическое применение для широкого круга задач динамического распределения ресурсов в вычислительных, производственных и логистических системах.

Ключевые слова: *мультиагентные системы, управление ресурсами, адаптивное планирование, оптимизация расписания, отказоустойчивость, системы реального времени.*

Цитирование: *Кирьяков Ф.М., Скобелев П.О.* Мультиагентный метод повышения адаптивности и эффективности управления вычислительными ресурсами в реальном времени. *Онтология проектирования*. 2025. Т.15, №3(57). С.418-435. DOI: 10.18287/2223-9537-2025-15-3-418-435.

Вклад авторов: *Скобелев П.О.* – постановка задачи и выработка подхода к разработке и исследованию, анализ результатов. *Кирьяков Ф.М.* – формализация задачи, разработка онтологической модели вычислительной сети и мультиагентного метода, примеры.

Конфликт интересов: авторы заявляют об отсутствии конфликта интересов.

Введение

Эффективное управление ресурсами (транспортные средства, производственные линии или вычислительные мощности и др.) остаётся ключевой проблемой для систем, где ограниченные ресурсы должны распределяться между конкурирующими задачами. Эта проблема решается в основном методами комбинаторного или эвристического планирования и оптимизации, основанными на пакетной обработке, где задачи и ресурсы известны заранее, а расписание строится без возможности последующей корректировки [1, 2]. Такой подход хорошо работает в предсказуемых и неизменных условиях – например, при планировании производ-

ственных циклов с фиксированными заказами или в вычислительных системах (ВС) с определённой нагрузкой.

В процессе роста сложности и динамики таких систем централизованные методы и алгоритмы управления стали встречать серьёзные трудности:

- с увеличением размерности решаемых задач традиционные методы оказываются малоприменимыми из-за комбинаторного роста числа вариантов и резкого увеличения объёма вычислений;
- требуется всё чаще учитывать не только интересы центра, но и отдельных узлов ВС, а также индивидуальные особенности поступающих задач;
- высокая неопределённость и динамика бизнес-процессов приближают их к границе «хаоса», а принятые ранее решения в любой момент могут изменяться;
- жёсткая привязка ресурсов к единому диспетчеру не обеспечивает необходимой масштабируемости и отказоустойчивости – ключевых требований для современных вычислительных и производственных сред.

Для преодоления этих трудностей в ВС в настоящей работе предложен метод, который обеспечивает возможность непрерывной квазиоптимизации временных параметров распределения задач между вычислительными ресурсами (ВР) в реальном времени. В этих целях разработаны модели и методы адаптивных пакетов изменений (патч, от англ. *patch*) для последовательных атомарных модификаций расписания, позволяющие точечно сдвигать задачи без их полной отмены, минимизировать потери при перепланировании и поддерживать актуальное состояние системы в реальном времени.

1 Обзор существующих решений

Недостатки централизованных ВС обусловили потребность в разработке распределённых и самоорганизующихся систем, управляемых коллективом автономных роботов, программных агентов, способных принимать согласованные локальные решения, находясь в коммуникации с другими агентами ВС [3, 4]. Такие ВС показывают превосходство в производительности, позволяя принимать решения в реальном времени [5], и обладают более высокой отказоустойчивостью [6].

Рассматриваемое направление получило развитие в Самарской школе мультиагентных (МА) систем [7]. На основе МА моделей сети потребностей и возможностей (ПВ-сетей) и метода компенсаций при взаимных уступках агентов сформулирована концепция эмерджентного интеллекта – коллективного искусственного интеллекта (ИИ), возникающего в открытых самоорганизующихся системах с использованием онтологий и МА технологий. Решение сложной задачи при таком подходе строится как процесс самоорганизации программных агентов с конфликтными интересами, которые выявляют и разрешают конфликты в ходе непрерывной конкуренции и кооперации на виртуальном рынке ВС. Разрешение конфликтов осуществляется за счёт переговоров агентов, сопровождающихся взаимными уступками, регулируемыми индивидуальными функциями удовлетворённости, и бонусов-штрафов агентов до достижения состояния консенсуса, отражающего «конкурентное равновесие» агентов (когда ни один из агентов не может улучшить своё состояние). Разработанный подход показал свою эффективность при решении ряда задач управления ресурсами в транспорте, промышленном производстве, цепочках поставок и других применениях [8].

Задача эффективного управления ВР стала особенно актуальной в связи с растущей сложностью решаемых задач, развитием суперкомпьютеров, методов и средств ИИ. В [9] предложена модель суперкомпьютерного кластера, развивающая методы систем массового обслуживания для повышения эффективности вычислений. В этой области разработан и применяется ряд решений на основе МА технологий. В [10] заложена теоретическая основа и проведён ряд исследований децентрализованного управления самоорганизующимися ВС. Однако предложенные в ней модели и методы имеют ограничение, связанное с невозможностью обеспечить максимальную адаптивность в реальном времени.

Известны решения, нацеленные на оптимизацию расписания в реальном времени. В [11] описан алгоритм оптимизации расписания задач *NDFS* (*Nearest Deadline First Scheduled*), исходя из заданного срока их выполнения с учётом их приоритетов. В [12] эта модель дополнена динамическим ранжированием ресурсов и резервным копированием задач на следующий по рангу ресурс, что повышает устойчивость системы к внештатному выходу ресурса из строя. В [13] описан подход к оптимизации расписания *FCFS-LRH* (*First Come First Served – Left-Right Hole*), нацеленный на минимизацию незаполненных промежутков в расписании.

Примеры решения задач динамического планирования в вычислительных средах приведены в [14-19], варианты применения МА систем для управления ресурсами и их расписаниями – в [20]. Большинство описанных в этих работах алгоритмов опирается лишь на общий приоритет задач и на временное окно, когда бронирование ресурса для задачи имеет смысл, и рассматривается только наиболее позднее возможное планирование задачи. При возникновении локального конфликта происходит полное перепланирование расписания или его участка, что может приводить к излишним и произвольным перестановкам в расписании.

Цель настоящей работы – переработать подход к заданию правил планирования задач и обеспечить более высокую адаптивность.

2 Формализация задачи

2.1 Элементы вычислительной системы и их основные параметры

Моделируемая ВС состоит из двух основных типов сущностей: ресурсов, которые выполняют работу, и задач, которые необходимо выполнить. Эти сущности представляются следующими наборами:

- набор ресурсов $R = \{r_j\}$, j $= \overline{1, J}$, где J – количество ресурсов;
- набор задач $O = \{o_i\}$, $\overline{1, I}$, где I – количество задач.

Важным условием является то, что в любой момент времени в системе могут происходить непредвиденные события, включая появление новых заказов или отказов в работе ВС, что должно вызывать перераспределение ресурсов и перепланирование работ.

2.2 Описание задач

Каждая задача o_i из набора O характеризуется длительностью выполнения и моделью функции удовлетворённости. Модель функции удовлетворённости s_i представляет собой набор точек $\left(t, s_i(t)\right)$, где $s_i(t)$ – показатель удовлетворённости задачи в случае её выполнения в момент времени t (см. рисунок 1). Функция удовлетворённости может принимать значения в диапазоне $s_i(t) \in [0,1]$. Временная ось не имеет ограничений. Для удобства восприятия графической информации в данной работе используется упрощённое представление – в секундах, от 0 до 10 включительно.

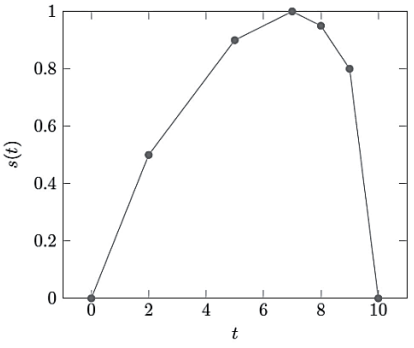

Рисунок 1 – Пример модели функции удовлетворённости

2.3 Постановка задачи

Необходимо для имеющихся групп ресурсов R и задач O максимизировать целевую функцию S, представляющую собой показатель общей удовлетворённости системы (ПОУС),

который включает все задачи и ресурсы: $S = \sum_{i=1}^{I} s_i(t_i)$, где t_i – время выполнения задачи o_i (в случае, если для задачи o_i не забронирован ресурс, то считать $s_i = 0$).

2.4 Модель функции удовлетворённости

Модель удовлетворённости представлена в виде равнобедренного треугольника. Такая модель позволяет упростить последующий анализ сложных функций, полученных путём комбинации (сложения, вычитания и пр.) множества других исходных моделей и функционально соответствует зависимости уровня удовлетворённости реальных задач от времени их выполнения.

На рисунке 2 показан график, где в качестве примера на общей временной оси представлена задача длительностью 4 секунды (фигура, заполненная сплошной заливкой) и её функция удовлетворённости (фигура, заштрихованная под углом 45 градусов). В данном примере задача находится в оптимальном положении, при котором время её выполнения соответствует наибольшему из возможных значений её функции удовлетворённости.

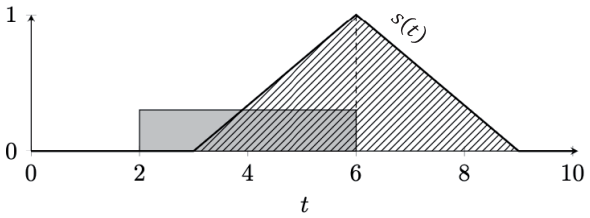

Рисунок 2 – Изображение задачи на графике и её функции удовлетворённости

3 Традиционный подход к решению задачи

3.1 Описание алгоритма

В числе традиционных алгоритмов планирования и оптимизации выполнения задач [1, 2] лежит принцип: «первый пришёл – первый исполнился». Для МА системы такой принцип часто применяется, и в алгоритмы не включается взаимодействие с другими агентами задач – агент нового заказа анализирует расписание на предмет свободных мест и выбирает наиболее подходящее место, согласно собственной функции удовлетворённости так, чтобы не затронуть другие задачи.

Пример 1 – наиболее благоприятный для этого алгоритма, когда задача o_{new} может свободно разместиться и выполниться в момент времени t_{peak}, совпадающий с пиком собственной функции удовлетворённости. На рисунке 3 видно, что свободное пространство, находящееся в диапазоне ненулевого значения функции удовлетворённости, представляет отрезок времени $t \in [2, 7]$ секунд. С учётом длительности выполнения задачи o_{new} возможное время выполнения задачи находится в пределах $t \in [4, 7]$ секунд (рисунок 4). Алгоритм ограничивает функцию удовлетворённости этим промежутком и выбирает наилучшую из возможных позиций для бронирования ресурса. Здесь функция удовлетворённости принимает наибольшее значение в момент времени 5 секунд. Следовательно, агент задачи забронирует ресурс в промежутке времени $t \in [3, 5]$ секунд (рисунок 5).

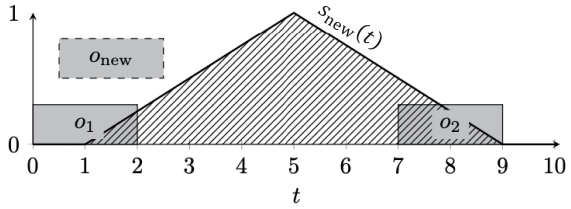

Рисунок 3 – Пример 1, исходная функция удовлетворённости задачи o_{new}

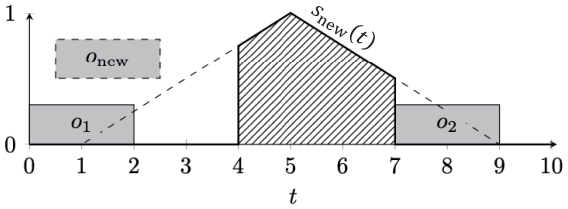

Рисунок 4 – Пример 1, функция удовлетворённости задачи o_{new} после её ограничения

Примере 2 – ограниченная функция удовлетворённости имеет область ненулевых значений $t \in [6,7]$ секунд (рисунок 6). Функция удовлетворённости принимает максимальное значение в момент времени 6 секунд. Следовательно, агент задачи забронирует промежуток времени $t \in [4,6]$ секунд (рисунок 7).

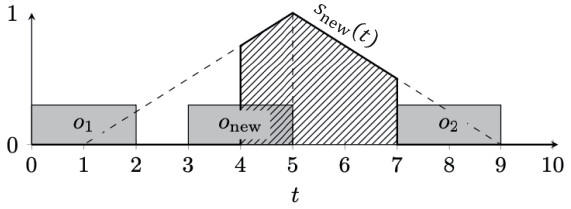

Рисунок 5 – Пример 1, результат размещения задачи o_{new}

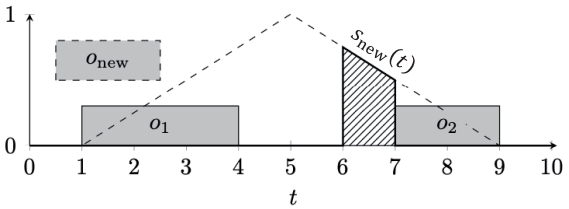

Рисунок 6 – Пример 2, функция удовлетворённости задачи o_{new} после её ограничения

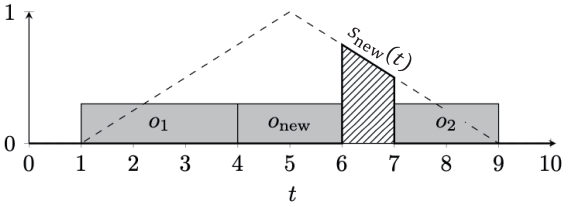

Рисунок 7 – Пример 2, результат размещения задачи o_{new}

3.2 Недостаток традиционного алгоритма

В *Примере 3* (рисунок 8) одна из задач (o_1) из последней конфигурации модели смещена вправо на 0.5 секунды. Это изменение в ВС приводит к резкому ухудшению работы алгоритма. На рисунке 8 можно видеть три свободных промежутка времени. Первые два – $t \in [0, 1.5]$ и $t \in [3.5, 5]$ секунд – имеют продолжительность 1.5 секунды и не могут вместить задачу продолжительностью 2 секунды. Последний промежуток времени (в конце расписания) не имеет ограничения длительности, но на нём невозможно расположить задачу o_{new} так, чтобы она выполнилась в момент времени, соответствующий ненулевому значению функции удовлетворённости.

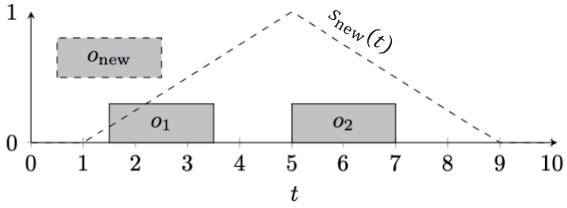

Рисунок 8 – Пример 3, агент задачи o_{new} не может забронировать ресурс

В примере 3 возможно сместить задачи o_1 и o_2 за счёт частичного снижения их показателя удовлетворённости так, чтобы между ними образовалось пространство в 2 секунды. Это позволит вместить новую задачу o_{new} и повысить ПОУС, но для этого нужны переговоры, чтобы учесть индивидуальные особенности каждого участника и его предпочтения.

4 Адаптивное формирование расписаний выполнения заданий на вычислительных ресурсах

4.1 Описание предлагаемого метода

Основная задача разрабатываемого метода – сделать процесс распределения ресурса коллективным, с учётом его актуального состояния. Такой алгоритм должен поддерживать возможность смещения задач, уже размещённых в расписании ресурса, и вытеснения и перемещения задач между ресурсами.

Вводятся два дополнительно термина.

- Модификация mod – атомарное изменение существующего расписания: добавление задачи, сдвиг задачи, вытеснение задачи, перераспределение на другой ресурс и т.д. Модификации не носят контекстуальный характер, они являются лишь составляющими конечного плана изменения расписания – патча.
- Патч $patch$ – конечный последовательный набор модификаций. Патч носит контекстуальный характер и всегда относится к конкретному ресурсу, он используется как универсальный пакет передачи информации об изменениях расписания конкретного ресурса как между агентами задач в процессе утверждения патча, так и при отправке утверждённого патча агенту ресурса.

Каждая модификация $mod_{n,m}$ патча $patch_m$ ($n = \overline{1, N_m}$, $m = \overline{1, M}$, N_m – количество модификаций в патче $patch_m$, M – количество патчей) вносит изменение в расположение задачи в расписании и влияет на её показатель удовлетворённости. Изменение показателя удовлетворённости задачи при применении к ней модификации $mod_{m,n}$ составляет $\Delta s_{n,m}$. Изменение ПОУС при применении патча $patch_m$ составляет $\Delta S^m_{\text{patch}}$ и равно сумме всех изменений показателей удовлетворённостей задач, вызванных применением модификаций этого патча $\Delta S^m_{\text{patch}} = \sum_{n=1}^{N_m} \Delta s_{n,m}$. Процесс формирования патча состоит из следующих этапов.

1) Агент задачи самостоятельно анализирует состояние расписания ресурса и вычисляет наиболее выгодный для системы патч. Здесь «наиболее выгодный» – это такой патч $patch_m$, который имеет наибольшее значение прироста ПОУС $\Delta S^m_{\text{patch}}$ среди всех доступных патчей.
2) Агент задачи передает патч другим агентам задач, затронутых этим патчем. Другие агенты задач могут принять патч или принять решение о смене ресурса и сообщить об этом агенту задачи, передавшему патч. При принятии решения о смене ресурса агент задачи, рассчитавший патч, рассчитывает его заново и сообщает об изменениях другим агентам задач. Этот цикл длится до тех пор, пока все агенты задач, затронутых патчем, не будут удовлетворены актуальным патчем либо сообщат о смене ресурса.
3) Агент задачи, рассчитавший патч, передаёт утверждённый патч агенту ресурса.
4) Агент ресурса применяет патч, и, в случае успешного применения, уведомляет всех затронутых агентов задач об успешном применении патча. В случае неудачи агент ресурса сообщает агенту задачи причину ошибки применения патча. Ошибки в применении патча могут возникать по различным причинам, например, устаревшая информация о расписании у агента задачи. Агент задачи может либо принять меры по разрешению проблемы, если такая возможность есть, либо перейти к рассмотрению другого ресурса.

4.2 Алгоритм расчёта наиболее выгодного патча

4.2.1 Формализация задачи

В контекст задачи входят:

- ресурс r и его расписание $schedule$;
- набор задач $O = o_i$, $\overline{1, I}$, уже находящихся в расписании $schedule$, где I – количество задач в расписании;
- задача o_{new}, не находящаяся в расписании $schedule$, которую необходимо разместить.

Новую задачу можно разместить между любыми двумя задачами или по краям расписания. При этом можно сместить любую из размещённых в расписании задач так, чтобы забронированные ими временные участки не пересекались. Всего вариантов размещения новой задачи относительно находящихся в расписании задач $I + 1$, где I – количество задач, уже размещённых в расписании. Для каждого варианта размещения рассматривается до K вариантов сдвига других задач: $K = (I + 1) \cdot \sum_{i=1}^{I} T_i$, где T_i – количество опорных временных точек модели функции удовлетворённости s_i задачи o_i. Область возможных решений представляет гиперповерхность в пространстве размерности $I + 2$, где:

- ось $axis_i, i = \overline{1, I}$ задаёт время t_i, к которому выполнится задача o_i;
- ось $axis_{I+1}$ задаёт время t_{new}, в которое выполнится размещаемая задача o_{new};
- ось $axis_{I+2}$ отражает приращение ПОУС $\Delta S^m_{\text{patch}}$ при расположении задач в расписании в соответствии со значениями остальных осей.

Изображение гиперповерхности для $I = 1$ представлено на рисунке 9. Проекция гиперповерхности на гиперплоскость $\Delta S = 0$ (заштриховано) представляет область допустимых вариантов размещения задач в расписании.

Задача алгоритма сводится к поиску патча с максимальным[1] приращением ПОУС. На рисунке 9 максимальному приращению ПОУС соответствует патч, устанавливающий время выполнения размещённой задачи $t_1 = 1$ секунда, а новой задачи $t_{new} = 4$ секунды.

Если нет ни одного варианта, для которого $\Delta S^m_{\text{patch}} > 0$, то добавление задачи o_{new} в расписание считается невозможным.

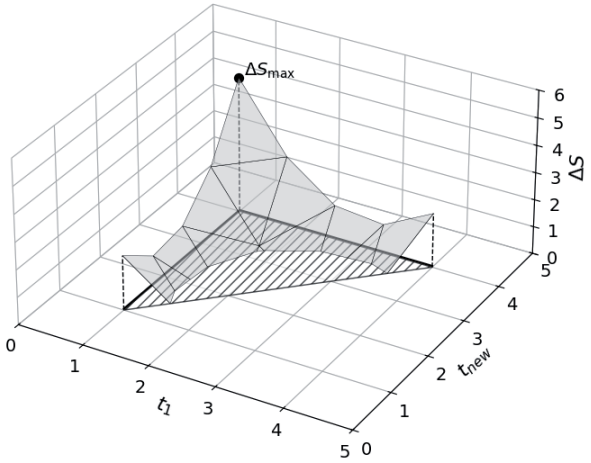

Рисунок 9 – Гиперповерхность возможных решений при $I = 1$

4.2.2 Описание алгоритма

Входными данными являются: состояние расписания ресурса; параметры новой задачи; параметры задач, размещённых в расписании. Результат работы алгоритма – патч с наибольшим приращением ПОУС среди всех возможных патчей.

Для каждого варианта размещения новой задачи в расписании итеративно применяются следующие этапы алгоритма.

1) Для выбранного размещения рассчитывается модель сдвига окружающих задач.
2) Из полученной модели сдвига задач и их функций удовлетворённости рассчитывается функция стоимости сдвига, отражающая зависимость потери в ПОУС от величины сдвига.
3) Рассчитывается функция приращения ПОУС как разница функции удовлетворённости новой задачи и функции стоимости сдвига.
4) Составляется патч, включающий добавление новой задачи и сдвиг окружающих задач в соответствии с максимальным значением функции приращения ПОУС.

Затем, из всех рассчитанных патчей выбирается один – с наибольшим значением приращения ПОУС.

Схема алгоритма представлена на рисунке 10.

Сдвиг задач, начиная с задачи o_n вперёд – это такое изменении времени бронирования $\Delta t_n^{\text{begin}} > 0$ задачи o_n, при котором время бронирования t_m^{begin} каждой последующий задачи $o_m, m > n$ в расписании изменяется на минимально возможное значение для выполнения условия $t_m^{\text{begin}} > t_{m-1}^{\text{end}}$, где t_{m-1}^{end} – это момент времени, в который выполнится задача o_{m-1}, предшествующая задаче o_m, если она будет принята к исполнению в момент времени o_{m-1}^{begin}.

Условия сдвигов назад и вперёд представлены формулами 1 и 2 соответственно.

$$\begin{cases} \Delta t_n^{\text{begin}} < 0 \\ t_m^{\text{end}} < t_{m+1}^{\text{begin}} \\ \Delta t_m^{\text{begin}} \to 0 \\ m > n \end{cases} \quad (1) \qquad \begin{cases} \Delta t_n^{\text{begin}} > 0 \\ t_m^{\text{begin}} > t_{m-1}^{\text{end}} \\ \Delta t_m^{\text{begin}} \to 0 \\ m < n \end{cases} \quad (2)$$

[1] Благодаря тому, что результирующая функция кусочно задана, она имеет конечное множество точек, где может быть найден максимум. Добавляя в алгоритм различные «фильтры», снижается количество вариантов перебора и повышается скорость работы алгоритма.

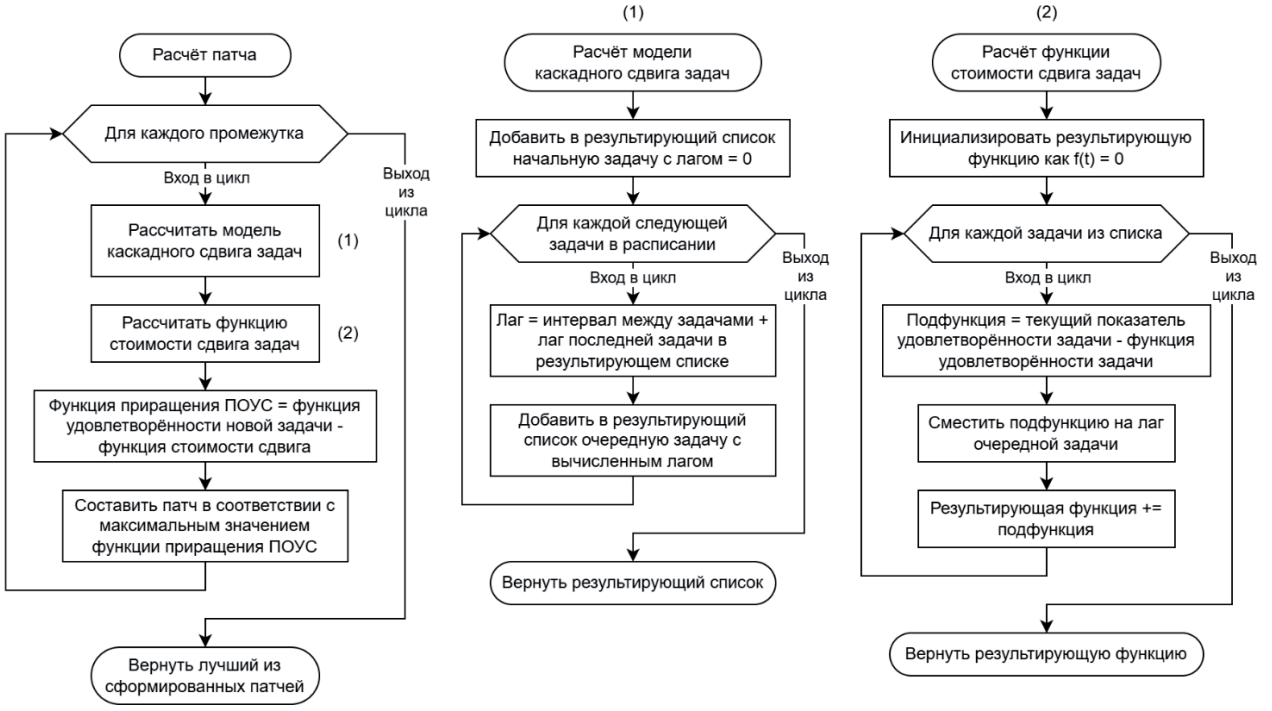

Рисунок 10 – Схема алгоритма расчёта и выбора наилучшего патча

4.2.3 Расчёт функции стоимости сдвига

В *Примере 4* (рисунок 11) представлена работа алгоритма расчёта функции стоимости сдвига для задач o_1 и o_2, где s_1 – функция удовлетворённости задачи o_1, s_2 – функция удовлетворённости задачи o_2. Обе задачи находятся в оптимальном положении. Задачи сдвигаются в сторону увеличения времени их брони (вправо), освобождая пространство перед задачей o_1.

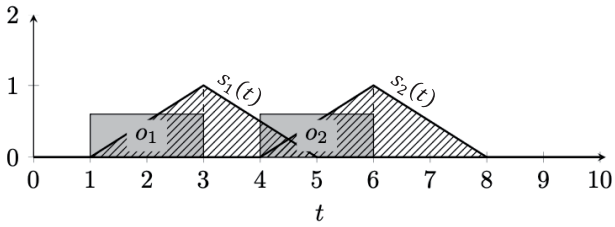

Рисунок 11 – Пример 4, функции удовлетворённости задач

Согласно изначальному расписанию задача o_1 должна начать обрабатываться в момент времени 1 секунда. Если новую задачу o_{new} возможно разместить так, чтобы она выполнилась в этот же момент времени, то задача o_1 не сдвигается и стоимость этого нулевого сдвига составит $\Delta S_{shift}(1) = s_1(3) - s_1(3) = 1 - 1 = 0$.

Отрезок времени $t \in [1, 2]$ секунд можно освободить, переместив только задачу o_1. При этом она встанет вплотную к задаче o_2. Стоимость сдвига составит $\Delta S_{shift}(2) = s_1(3) - s_1(4) = 1 - 0.5 = 0.5$. Функция стоимости сдвига (рисунок 12) заштрихована под углом 135 градусов. Чтобы освободить участок $t \in [2, 3]$ секунд, понадобится сдвигать обе задачи (см. условия сдвига (1) и (2) в разделе 4.2.2). Функция стоимости сдвига равна сумме изменений их показателей удовлетворённости $\Delta S_{shift}(3) = \big(s_1(3) - s_1(5)\big) + \big(s_2(6) - s_2(7)\big) = (1 - 0) + (1 - 0.5) = 1.5$ (рисунок 13). При дальнейшем смещении задачи o_1 функция удовлетворённости s_1 не будет оказывать влияния на результирующую, т.к. значение её функции удовлетворённости будет неизменно равно нулю (рисунок 14). Здесь $\Delta S_{shift}(4) = \big(s_1(3) - s_1(6)\big) + \big(s_2(6) - s_2(8)\big) = (1 - 0) + (1 - 0) = 2$. Дальнейшее смещение не повлияет на значение функции стоимости сдвига, т.к. значения обеих функций удовлетворённости неизменно будут равны нулю (рисунок 15).

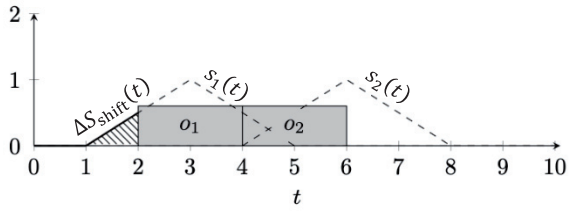

Рисунок 12 – Пример 4, функция стоимости сдвига для $t \in [1, 2]$

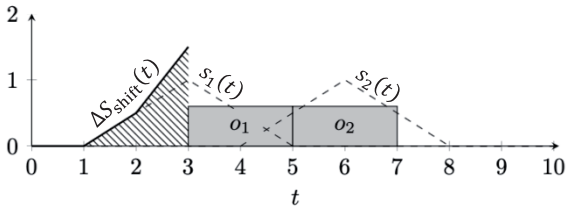

Рисунок 13 – Пример 4, функция стоимости сдвига для $t \in [1, 3]$

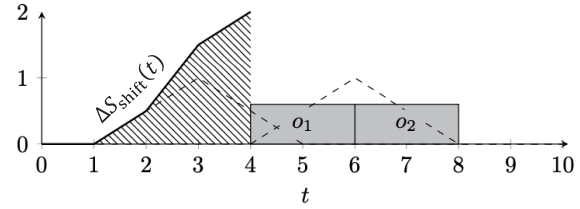

Рисунок 14 – Пример 4, функция стоимости сдвига для $t \in [1, 4]$

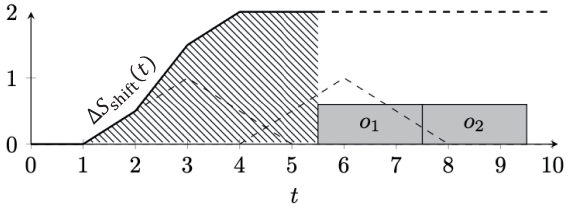

Рисунок 15 – Пример 4, полностью рассчитанная функция стоимости сдвига

4.2.4 Расчёт функции приращения ПОУС

Разница между функцией удовлетворённости s_{new} новой задачи o_{new} и функцией стоимости сдвига задач ΔS_{shift}, находящихся в расписании, покажет приращение ПОУС ΔS_{patch} при применении патча, содержащего модификации добавления новой задачи и смещения размещённых.

Если в расписании присутствуют задачи по обе стороны от вставки, то функция стоимости сдвига ΔS_{shift} рассчитывается из двух компонент: левосторонней $\Delta S_{shift}^{\text{left}}$ и правосторонней $\Delta S_{shift}^{\text{right}}$. Работа алгоритма расчёта приращения ПОУС представлена на примере 5 с одним отличием от примера 4: задача o_{new} размещается между задачами o_1 и o_2 (рисунок 16). Результат вычисления обеих компонент функции стоимости сдвига представлен на рисунке 17. Правосторонняя функция стоимости сдвига отражает потерю в ПОУС при сдвиге времени начала обработки задачи o_2. Начало обработки задачи o_2 будет совпадать с временем окончания обработки добавляемой задачи o_{new}. Функция удовлетворённости также привязана к времени окончания обработки задачи. При рассмотрении правосторонней функции стоимости сдвига разница между s_{new} и s_2 будет отражать приращение ПОУС в зависимости от расположения задач o_{new} и o_2.

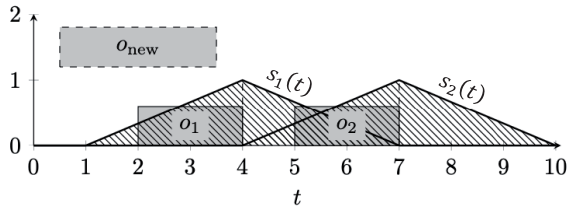

Рисунок 16 – Пример 5, функции удовлетворённости задач

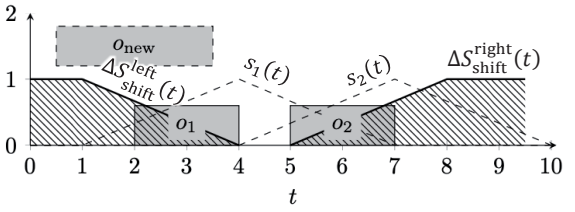

Рисунок 17 – Пример 5, подготовленные к объединению функции стоимости сдвига

В отличие от правосторонней функции стоимости сдвига левосторонняя отражает потерю в ПОУС при сдвиге времени окончания обработки задачи o_1 и выбранном начале времени обработки задачи o_{new}. Чтобы две функции стоимости сдвига можно было сложить, необходимо предварительно сместить левостороннюю компоненту вперёд на промежуток, равный длительности новой задачи o_{new} (рисунок 18). Результат сложения обеих частей функции стоимости сдвига и функция удовлетворённости s_{new} представлены на рисунке 19.

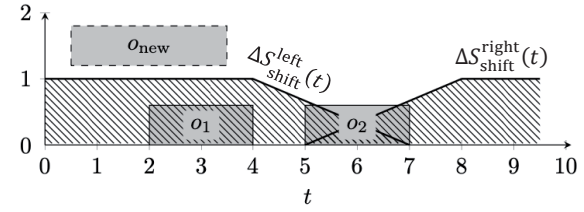

Рисунок 18 – Пример 5, компоненты функции стоимости сдвига для обоих направлений

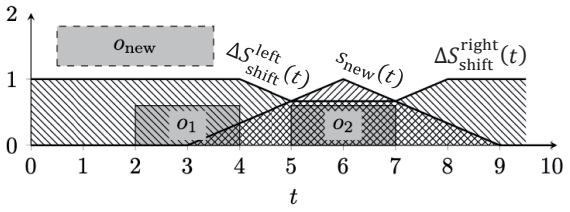

Рисунок 19 – Пример 5, функция стоимости сдвига (штриховка 135°) и функция удовлетворённости s_{new} новой задачи o_{new} (штриховка 45º)

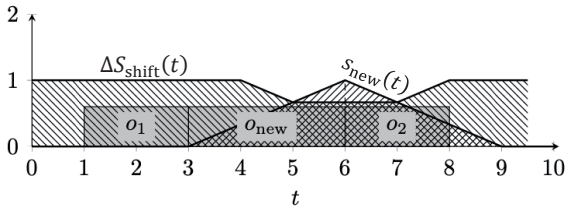

Рисунок 20 – Пример 5, результат размещения новой задачи

На данном этапе становится видно, как выглядит разница s_{new} и функции стоимости сдвига. Задача o_{new} размещается в соответствии с максимальным значением их разности (рисунок 20).

5 Прототип системы управления вычислительными ресурсами

Акторная система реализована на языке *Python* 3.13 с использованием библиотек:

- *thespian* – реализация модели акторов для создания распределённых и многопоточных систем;
- *pydantic* – валидация данных и генерация конструкторов классов;
- *loguru* – журналирование работы системы;
- *pytest* – модульное тестирование;
- *pickle* – сериализация сообщений между агентами.

5.1 Архитектура мультиагентной системы

Роль управления агентами полностью делегирована от центрального объекта акторной системы специальному агенту-менеджеру, который, в отличие от объекта акторной системы, находится в потоке, не связанном с системной оболочкой, что позволяет ему обрабатывать входящие сообщения, не блокируя интерфейс программы. Акторная система выполняет две задачи: инициализация агента-менеджера при запуске модели и чтение стандартного потока ввода. При вводе команд в стандартный поток акторная система отправляет сообщение с текстом команды актору-менеджеру и переходит в режим ожидания следующей команды. Это позволяет вносить изменения в ВС параллельно с протекающим в ней активным процессом, используя единый интерфейс взаимодействия с ВС (консоль программы, объединяющая стандартные потоки ввода и вывода), что упрощает проведение экспериментов.

Агент-популятор предназначен для автоматизации проведения множественных последовательных экспериментов. Он перебирает различные варианты конфигурации ВС, запускает акторную систему с выбранными параметрами, собирает данные, записывает в таблицу и повторяет процесс для каждой комбинации входных параметров.

Основные агенты модели – это агенты задач и агенты ресурсов. Схема иерархии агентов представлена на рисунке 21.

Агент ресурса хранит свойства ресурса и его текущее расписание, которое представляет собой хранящий упорядоченный список непересекающихся временных интервалов, представленных в виде объектов броней. Класс расписания реализует все необходимые функции для анализа расписания на свободные промежутки времени, валидацию и применение патчей, поиск брони по её свойствам или временной точке, визуализации текущего состояния. Класс брони хранит информацию о временном промежутке бронирования ресурса, информацию о задаче, а также информацию о соседних бронях, если бронь помещена в расписание.

Агент задачи хранит свойства задачи, текущее состояние бронирования ресурса и информацию о всех доступных ресурсах. В рамках проведения эксперимента использованы две модели поведения агента задачи – пассивная и активная, алгоритмы поведения которых рассмотрены в подразделах 3.1 и 4.2.

Рисунок 21 – Схема иерархии агентов

Класс агента библиотеки *thespian* дополнен функциональностью, позволяющей организовывать более сложные цепочки коммуникаций агентов.

- *Привязка к агенту-менеджеру*. Эта модификация позволяет агенту запрашивать информацию о текущем состоянии ВС, например, количество агентов выбранного типа.
- *Возможность широковещательной отправки сообщений*. Агент может отправить сообщение конкретному агенту или группе агентов, выделив их по какому-либо признаку.
- *Сообщение-запрос*. Это модифицированный вид сообщения с внедрённой в класс агента логикой, требующей от получателя ответить отправителю сообщением-ответом.
- *Множественный запрос*. Эта модификация позволяет связывать функцию для отложенной обработки ответа не с единственным запросом, а с группой запросов.
- *Отложенный ответ*. Эта модификация позволяет агенту-получателю сообщения-запроса не отвечать агенту-отправителю немедленно. Агент сохраняет в своём состоянии данные запроса и его содержание. Агент может проверить наличие отложенных ответов и отправить агенту-отправителю сообщение-ответ.

Класс агента получил дополнительные функции ведения журнала, счётчика и ограничителя отправленных сообщений, не позволяющего агентам бесконечно обмениваться цикличными сообщениями.

5.2 Протокол взаимодействия агентов

При запуске МА системы восстанавливается состояние акторной системы по предварительно заданной конфигурации. Считывается конфигурационный файл, и для каждого описанного агента отправляется агенту-менеджеру сообщение с данными для инициализации нового агента. Схема взаимодействия агентов при запуске модели представлена на рисунке 22.

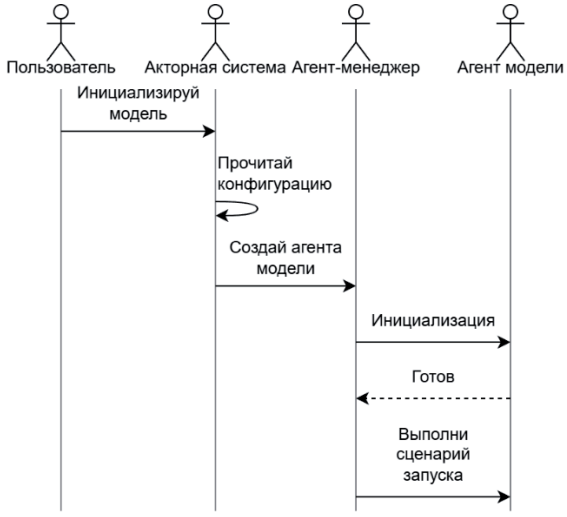

Рисунок 22 – Процесс восстановления состояния акторной системы по заданной конфигурации

Агент-менеджер выступает в роли посредника, так как только он хранит информацию о всех активных агентах модели. Агент-менеджер включает агента модели в список активных агентов после получения от него сообщения о готовности к работе и сообщает ему команду для выполнения сценария запуска. Данное разделение процесса служит для предотвращения двух неблагоприятных случаев: агент, не завершивший инициализацию, получает сообщение от другого агента модели; агент модели получает сообщение от агента, которого нет в списке активных агентов. Таким образом гарантируется, что отправитель и получатель любого сообщения между агентами модели включены в список активных агентов.

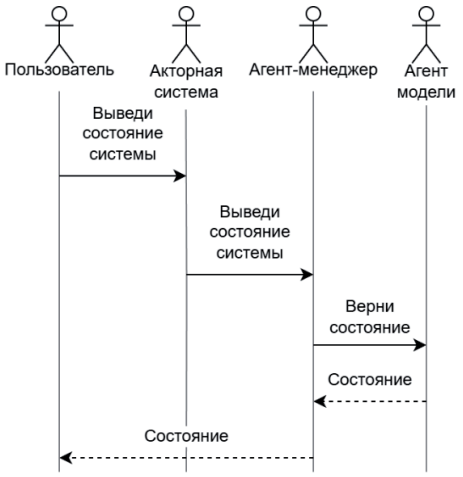

Рисунок 23 – Процесс выполнения пользовательской команды

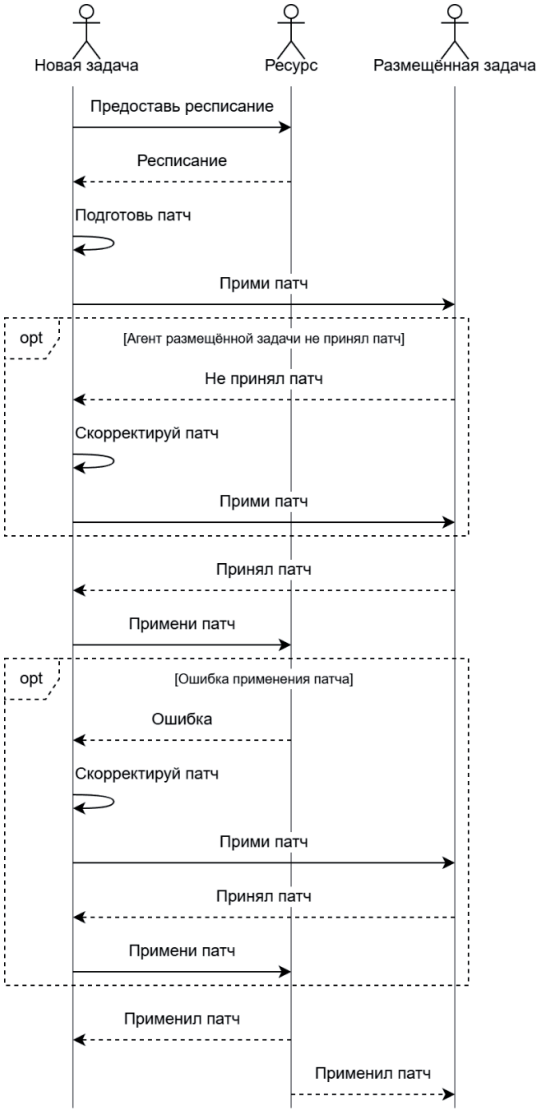

Рисунок 24 – Процесс формирования и принятия патча

Взаимодействие МА системы с пользователем также выполняется через агента–менеджера (см. рисунок 23).

5.3 Формирование и применение патча

Чтобы забронировать ресурс, агент задачи (агент-инициатор) запрашивает информацию о расписании у агента ресурса и рассчитывает патч, включающий добавление собственной задачи в расписание (рисунок 24). Если патч содержит изменения состояния других задач, то агент-инициатор направляет им патч на просмотр. Агент задачи может принять либо отвергнуть патч, в зависимости от своей модели поведения. Сообщение о непринятии патча содержит причину отказа, которая используется для его корректировки.

Если все затронутые агенты задач приняли патч, то его финальная версия направляется агенту ресурса, который валидирует патч на возможность его применения. Если применение патча невозможно, то агент ресурса сообщает об этом агенту-инициатору, указывая причину и актуальное состояние ресурса.

После корректировки патча агент-инициатор отправляет его на просмотр другим включённым в патч агентам задач. После принятия патча, агент-инициатор снова отправляет патч агенту ресурса. Если агент ресурса определит патч валидным, то он применит этот патч на собственном расписании и уведомит всех затронутых агентов задач об изменении их положения в расписании.

Метод разрешения конфликтов посредством переговоров между агентами потенциально может приводить к образованию замкнутых циклов взаимодействий, известных как взаимные блокировки.

Существует два сценария, ведущих к цикличности в переговорах: отказ агента задачи от предлагаемого патча и отказ агента ресурса от применения патча.

Отказ агента задачи возможен при наличии у него информации о другом ресурсе, который агент оценивает как потенциально более предпочтительный, исходя из возможностей реализации своей функции удовлетворённости. Такой агент немедленно прекращает участие в текущем процессе согласования и покидает соответствующий ресурс. В переговорах остаются те агенты, которые согласны с предложенными условиями.

Отказ агента ресурса от применения патча возможен в ситуации, когда предоставленный патч был сформирован на ос-

нове устаревшей информации о состоянии ресурса. В этом случае агенту задачи направляется обновлённая информация о состоянии ресурса, на основании которой он пересматривает свою стратегию.

Применение патча – синхронный и атомарный процесс: в каждый момент времени только один агент задачи может успешно применить изменения к расписанию ресурса. Это означает, что в случае поступления нескольких конкурирующих патчей будет гарантировано, что один из них будет применён первым, а остальные получат информацию об изменившемся состоянии ресурса. Благодаря этому агенты, получив обновлённые данные, гарантированно смогут применить патч на ресурсе.

6 Вычислительный эксперимент

Сравнение двух методов между собой производилось путём тестирования, которое выполнял специальный агент. Ему передавалась информация о варьируемых элементах конфигурации ВС: пределы варьирования, шаг варьирования или заранее заданные варианты:

- тип поведения агента задачи – активный или пассивный (пассивное поведение соответствовало следованию традиционному алгоритму, активное – предлагаемому);
- количество задач (от 5 до 25 задач с шагом 5);
- длительность задач (от 2 до 6 часов с шагом 1 час):
- продолжительность модели функции удовлетворённости задач (для всех задач модель функции удовлетворённости имела форму равнобедренного треугольника: длина основания треугольника является продолжительностью модели функции удовлетворённости задачи и варьировалась в промежутке от 3 до 12 часов с шагом 3).

Агент-популятор поочерёдно воспроизводил состояние системы для каждой из возможных комбинаций параметров, все задачи имели одинаковые исходные данные за исключением благоприятного времени выполнения задачи. После завершения переговоров между агентами задач и ресурсов агент-популятор запрашивал их показатели удовлетворённости, на этой основе получал информацию о количестве задач, успешно зарегистрировавших ресурс, и рассчитывал ПОУС.

Каждый запуск ВС имеет случайный фактор, вызванный невозможностью предсказать, в каком порядке будут обрабатываться сообщения агентов, запущенных одновременно. Чтобы компенсировать влияние случайного фактора каждая конфигурация ВС воспроизводилась несколько раз. Всего проведено 5 последовательных экспериментов.

На рисунке 25 показано распределение ПОУС, где видно, что мода распределения у предлагаемого (активный тип поведения агентов задач) алгоритма выше, чем у традиционного (пассивный тип поведения агентов задач).

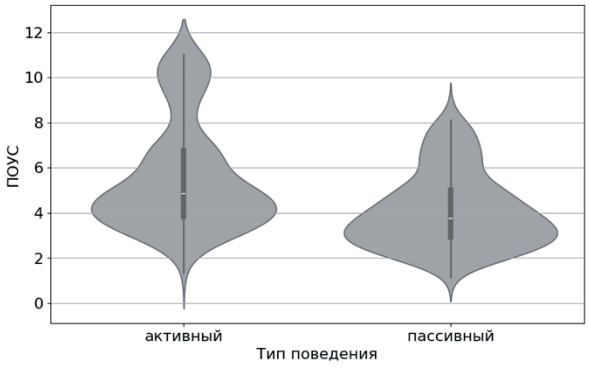

Рисунок 25 – Распределение ПОУС для активного и пассивного типов поведения агентов задач

На рисунках 26 – 28 показаны зависимости ключевых показателей ВС – ПОУС и количество агентов задач, успешно забронировавших ресурс от варьируемых параметров системы.

На приведённых графиках видно, что адаптивный метод показывает на 25–30% лучший результат по сравнению с традиционным. Это объясняется большей гибкостью предлагаемого алгоритма, который способен найти способ вместить в расписание задачу и повысить ПОУС, в то время как традиционный алгоритм сочтёт невозможным добавление новой задачи в расписание.

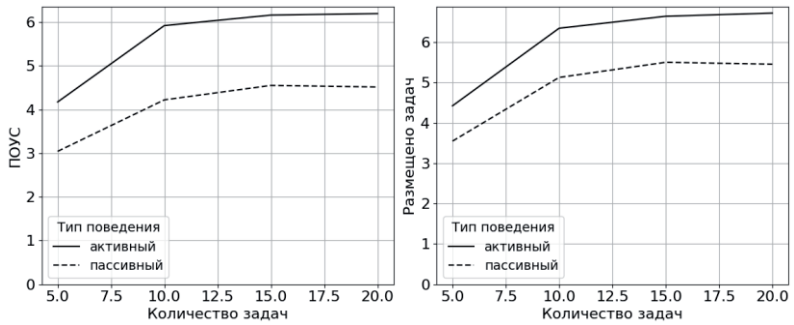

Рисунок 26 – Зависимость ключевых показателей от количества задач в системе

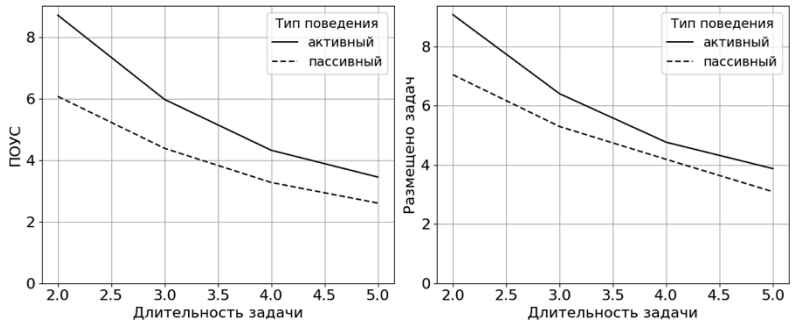

Рисунок 27 – Зависимость ключевых показателей от длительности задач

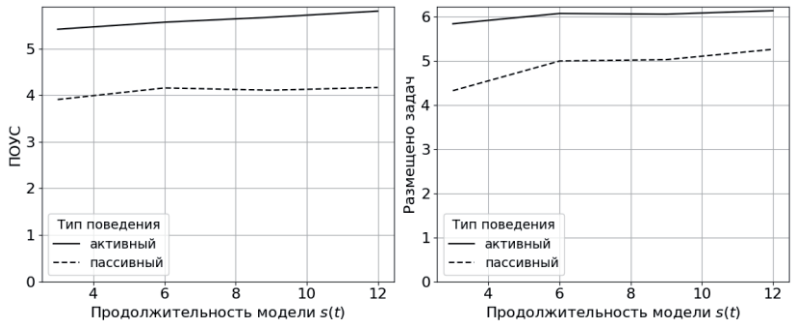

Рисунок 28 – Зависимость ключевых показателей от продолжительности модели функции удовлетворённости

Заключение

Разработанный адаптивный метод управления ВР на основе МА технологий ПВ-сетей имеет значительный потенциал для повышения эффективности перераспределения задач в реальном времени. В отличие от алгоритмов, ограниченных пакетным планированием, предложенный метод с адаптивными патчами позволяет выполнять модификации расписания, минимизируя потери при перепланировании и сохраняя устойчивость ВС. Преимуществами метода являются:

- высокая адаптивность, т.к. допускается изменять любые ранее принятые решения по включению и сдвигам заказов в расписания ВР;
- коллективное принятие решений, при котором агенты задач анализируют расписание не только с позиции собственных интересов, но и потенциально учитывают влияние изменений на показатель общей удовлетворённости системы;
- гибкость и масштабируемость, достигаемые за счёт децентрализованного взаимодействия агентов;

- подтверждённое вычислительным экспериментом повышение ПОУС на 25–30% за счёт применения адаптивных патчей по сравнению с традиционным методом.

Предложенный метод обладает адаптивностью для интеграции новых функций. Его развитие открывает перспективы практического применения для задач динамического распределения ресурсов в вычислительных, производственных и логистических системах.

Список источников

[1] ***Kulkarni A.J., Siarry P.*** Handbook of AI-based Metaheuristics. 2021. P.301–322. DOI: 10.1201/9781003162841.

[2] ***Gendreau M., Potvin J.-Y.*** Handbook of Metaheuristics (3rd Edition). 2019. P.353–384. DOI: 10.1007/978-3-319-91086-4.

[3] ***Coulouris G., Dollimore J., Kindberg T. Blair G.*** Distributed Systems. Concepts and Design. 5th edition. 2011. P.41–51. ISBN 13: 978-0-13-214301-1.

[4] ***Kalyaev A., Korovin I.*** New Method to Use Idle Personal Computers for Solving Coherent Tasks. AASRI Procedia. Vol. 9. 2014. P.131–137. DOI: 10.1016/j.aasri.2014.09.021.

[5] ***Calvaresi D., Dicente Cid Y., Marinoni M., Dragoni A. F., Najjar A., Schumacher M.*** Real-time Multi-Agent Systems: Rationality, Formal Model, and Empirical Results. 2021. P.27–31. DOI: 10.1007/s10458-020-09492-5.

[6] ***Мельник Э.В., Клименко А.Б.*** Комплексный метод организации управления отказоустойчивой информационно-управляющей системой на основе многоагентного взаимодействия. *Известия Тульского государственного университета. Технические науки*. 2017. № 9-1. С.136–149. EDN: ZXOOCL.

[7] ***Грачев С.П., Жиляев А.А., Ларюхин В.Б., Новичков Д.Е., Галузин В.А., Симонова Е.В., Майоров И.В., Скобелев П.О.*** Методы и средства построения интеллектуальных систем для решения сложных задач адаптивного управления ресурсами в реальном времени. *Автоматика и телемеханика*. 2021. №11. С.30-67. DOI: 10.31857/S0005231021110039.

[8] ***Амелина Н.О., Лада А.Н., Майоров И.В., Скобелев П.О., Царев А.В.*** Исследование моделей организации грузовых перевозок с применением мультиагентной системы для адаптивного планирования мобильных ресурсов в реальном времени. *Проблемы управления*. 2011. № 6. С.32–38.

[9] ***Zayats O.I., Baksheev V.E., Zaborovsky V.S., Muliukha V.A.*** Model of a supercomputer cluster in the form of a queueing system with a random limit on the execution time of applied tasks. *Computing, Telecommunications and Control*. 2024. T.17. No.3. P.71–83. DOI: 10.18721/JCSTCS.17307.

[10] ***Каляев А.И., Каляев И.А.*** Самоорганизующиеся распределённые системы. Москва: РАН, 2023. С.156.

[11] ***Goswami S., Das A.*** An Adaptive Resource Allocation Scheme in Computational Grid. *International Journal of Control Theory and Applications*. 2016. No.9. P.721–736.

[12] ***Goswami S.***, Mukherjee K. High Performance Fault Tolerant Resource Scheduling in Computational Grid Environment. *International Journal of Web-Based Learning and Teaching Technologies*. 2020. No.15. P.73–87. DOI: 10.4018/IJWLTT.2020010104.

[13] ***Pujiyanta A., Nugroho L.E., Widyawan.*** Resource allocation model for grid computing environment. *International Journal of Advances in Intelligent Informatics*. 2020. Vol.6, No.2. P.185–196. DOI: 10.26555/ijain.v6i2.496

[14] ***Galuzin V., Galitskaya A., Grachev S., Laruchkin V., Novichkov D., Skobelev P., Zhilyaev A.*** The Autonomous Digital Twin of Enterprise: Method and Toolset for Knowledge-Based Multi-Agent Adaptive Management of Tasks and Resources in Real Time. *Mathematics*. 2022. No.10. P.13–15. DOI: 10.3390/math10101662.

[15] ***Xie, N., Li, W., Zhang, J., Zhang, X.*** Research on Cloud Task Scheduling Algorithm with Conflict Constraints Based on Branch-and-Price. 2013. P.11. DOI: 10.3390/app13137505.

[16] ***Wang G., Feng J., Jia D., Song J., Li G.*** Cloud Task Scheduling using Particle Swarm Optimization and Capuchin Search Algorithms. 2023. P.1009–1017. DOI: 10.14569/IJACSA.2023.01407109.

[17] ***Movahedi Z., Defude B., Hosseininia A. M.*** An efficient population-based multi-objective task scheduling approach in fog computing systems. 2021. P.15-23. DOI: 10.1186/s13677-021-00264-4.

[18] ***Garg. R., Singh A.K.*** Adaptive workflow scheduling in grid computing based on dynamic resource availability. *Engineering Science and Technology, an International Journal*. 2015. Vol.18. P.256–269. DOI: 10.1016/j.jestch.2015.01.001.

[19] ***Haque A., Alhashmi S.M., Parthiban R.*** An optimization-based adaptive resource management framework for economic Grids: A switching mechanism. *Future Generation Computer Systems.* 2015. Vol.47. P.48–59. DOI: 10.1016/j.future.2014.10.022.

[20] ***Binyamin S.S., Slama S.B.*** Multi-Agent Systems for Resource Allocation and Scheduling in a Smart Grid. *Sensors*. 2022. No.22. P.13. DOI: 10.3390/s22218099.

Сведения об авторах

Кирьяков Федор Михайлович, 2001 г. рождения. Окончил бакалавриат СамГТУ в 2023 г. Магистрант кафедры «Информатика и вычислительная техника» Института автоматики и вычислительной техники СамГТУ. ORCID: 0009-0000-1585-9852; *kiriakov.f.m@gmail.com*. ✉

Скобелев Пётр Олегович, 1960 г. рождения. Окончил Куйбышевский авиационный институт (1983), к.т.н. (1986), д.т.н. (2003). Главный научный сотрудник СамНЦ РАН, профессор кафедры «Информатика и вычислительная техника» Института автоматики и вычислительной техники СамГТУ. Автор более 350 научных работ и учебных пособий. ORCID 0000-0003-2199-9557. *petr.skobelev@gmail.com*.

Поступила в редакцию 16.04.2025. после рецензирования 26.05.2025. Принята к публикации 10.06.2025.

Scientific article DOI: 10.18287/2223-9537-2025-15-3-418-435

Multi-agent method to improving adaptive real-time management of computing resources

© 2025, F.M. Kiriakov[1]✉, P.O. Skobelev[1-2]

[1] *Samara State Technical University (SamSTU), Samara, Russia*

[2] *Samara Science Centre of RAS (SamSC RAS), Samara, Russia*

Abstract

The paper presents the development of a multi-agent method aimed at meeting the growing demand for computing resources by enhancing the adaptability and efficiency of real-time management. In practical scenarios, it is essential to enable prompt and flexible adaptive adjustments to the task execution schedule in order to improve overall resource utilization. The proposed multi-agent resource management method is based on a previously developed "network of needs and capabilities" model, which enables smooth, adaptive modifications to the execution schedule. This process involves a sequence of atomic stepwise changes to the resource allocation plan, including local task shifts within a single computing resource, as well as the displacement and redistribution of tasks across multiple resources. Each task agent calculates an optimal “patch” to maximize the global efficiency of the system, accounting for the satisfaction functions of all tasks affected by the change. A key innovation is the introduction of a collective decision-making mechanism based on the computation and coordination of these patches. This allows for dynamic optimization of the schedule without requiring full rescheduling or transitioning to a fully decentralized solution, which would eliminate the shared data environment of the agent system. Experimental results demonstrate that the proposed method increases system efficiency by 25–30% compared to non-adaptive control approaches, which lack the ability to selectively revise agent interactions or reallocate tasks among resources. The method also enhances the scalability and fault tolerance of the system, expanding its applicability to a broad range of dynamic resource allocation problems in computing, manufacturing, and logistics.

Keywords: *multi-agent systems, resource management, adaptive planning, schedule optimization, fault tolerance, real-time systems.*

For citation: *Kiriakov FM, Skobelev PO.* Multi-agent method to improving adaptive real-time management of computing resources [In Russian]. *Ontology of designing*. 2025; 15(3): 418-435. DOI: 10.18287/2223-9537-2025-15-3-418-435.

Authors' contributions: *P.O. Skobelev* – problem statement and development of the research approach, analysis of the results. *F.M. Kiryakov* – task formalization, development of the ontological model of the computing network and the proposed multi-agent method, preparation of illustrative examples.

Conflict of interest: The authors declare no conflict of interest.

List of figures

References

[1] ***Kulkarni AJ, Siarry P.*** Handbook of AI-based Metaheuristics. 2021. P.301–322. DOI: 10.1201/9781003162841.

[2] ***Gendreau M, Potvin J-Y.*** Handbook of Metaheuristics (3rd Edition). 2019. P.353–384. DOI: 10.1007/978-3-319-91086-4.

[3] ***Coulouris G, Dollimore J, Kindberg T, Blair G.*** Distributed Systems. Concepts and Design. 5th edition. 2011. P.41–51. ISBN 13: 978-0-13-214301-1.

[4] ***Kalyaev A, Korovin I.*** New Method to Use Idle Personal Computers for Solving Coherent Tasks. AASRI Procedia. 2014; 9: 131–137. DOI: 10.1016/j.aasri.2014.09.021.

[5] ***Calvaresi D, Dicente Cid Y, Marinoni M, Dragoni AF, Najjar A, Schumacher M.*** Real-time Multi-Agent Systems: Rationality, Formal Model, and Empirical Results. 2021. P.7–31. DOI: 10.1007/s10458-020-09492-5.

[6] ***Melnik EV, Klimenko AB.*** A Complex Method for Organizing the Management of a Fault-Tolerant Information and Control System Based on Multi-Agent Interaction. [In Russian]. *Izvestiya Tula State University* (Izvestiya TulGU), 2017, No. *9*, P.136–149. EDN: ZXOOCL.

[7] ***Grachev SP, Zhilyaev AA, Laryukhin VB, Novichkov DE, Galuzin VA, Simonova EV, Mayorov IV, Skobelev PO.*** Methods and tools for development intelligent systems for solving complex real-time adaptive resource management problems [In Russian]. *Automation and Remote Control*. 2021; 11: 30–67. DOI: 10.31857/S0005231021110039.

[8] ***Amelina NO, Lada AN, Mayorov IV, Skobelev PO, Tsarev AV.*** Investigation of Models for Organizing Freight Transportation Using a Multi-Agent System for Adaptive Real-Time Planning of Mobile Resources. [In Russian]. *Problemy Upravleniya* (*Control Sciences*), 2011; 6: P.32–38.

[9] ***Zayats OI, Baksheev VE, Zaborovsky VS, Muliukha VA.*** Model of a supercomputer cluster in the form of a queueing system with a random limit on the execution time of applied tasks. *Computing, Telecommunications and Control*. 2024; 17: P.71–83. DOI: 10.18721/JCSTCS.17307.

[10] ***Kalyaev AI, Kalyaev IA.*** Self-Organizing Distributed Systems [In Russian]. 2023. Moscow: RAS. P.156.

[11] ***Goswami S, Das A.*** An Adaptive Resource Allocation Scheme in Computational Grid. International Journal of Control Theory and Applications. 2016; 9: 721–736.

[12] ***Goswami S***, ***Mukherjee K***. High Performance Fault Tolerant Resource Scheduling in Computational Grid Environment. *International Journal of Web-Based Learning and Teaching Technologies*. 2020; 15: 73–87. DOI: 10.4018/IJWLTT.2020010104.

[13] ***Pujiyanta A, Nugroho LE, Widyawan.*** Resource allocation model for grid computing environment. International *Journal of Advances in Intelligent Informatics*. 2020; 6(2): 185–196. DOI: 10.26555/ijain.v6i2.496

[14] ***Galuzin V, Galitskaya A, Grachev S, Laruchkin V, Novichkov D, Skobelev P, Zhilyaev A.*** The Autonomous Digital Twin of Enterprise: Method and Toolset for Knowledge-Based Multi-Agent Adaptive Management of Tasks and Resources in Real Time. Mathematics. 2022; 10: 13–15. DOI: 10.3390/math10101662.

[15] ***Xie N, Li W, Zhang J, Zhang X.*** Research on Cloud Task Scheduling Algorithm with Conflict Constraints Based on Branch-and-Price. 2013. P.11. DOI: 10.3390/app13137505.

[16] ***Wang G., Feng J., Jia D., Song J., Li G.*** Cloud Task Scheduling using Particle Swarm Optimization and Capuchin Search Algorithms. 2023. P.1009–1017. DOI: 10.14569/IJACSA.2023.01407109.

[17] ***Movahedi Z., Defude B., Hosseininia A. M.*** An efficient population-based multi-objective task scheduling approach in fog computing systems. 2021. P.15-23. DOI: 10.1186/s13677-021-00264-4.

[18] ***Garg. R., Singh A.K.*** Adaptive workflow scheduling in grid computing based on dynamic resource availability. *Engineering Science and Technology, an International Journal*. 2015. Vol.18. P.256–269. DOI: 10.1016/j.jestch.2015.01.001.

[19] ***Haque A., Alhashmi S.M., Parthiban R.*** An optimization-based adaptive resource management framework for economic Grids: A switching mechanism. *Future Generation Computer Systems.* 2015. Vol.47. P.48–59. DOI: 10.1016/j.future.2014.10.022.

[20] ***Binyamin SS, Slama SB.*** Multi-Agent Systems for Resource Allocation and Scheduling in a Smart Grid. *Sensors*. 2022; 22: 13. DOI: 10.3390/s22218099.

About the authors

Fedor Mikhailovich Kiryakov (b. 2001) graduated with a Bachelor's degree from SamSTU in 2023. Master's student at the Institute of Automation and Computer Engineering of the SamSTU. ORCID: 0009-0000-1585-9852; *kiriakov.f.m@gmail.com.* ✉

Petr Olegovich Skobelev (b. 1960) graduated from Kuibyshev Aviation Institute in 1983. Candidate of Technical Sciences (1986), Doctor of Technical Sciences (2003). Chief Researcher at the SamSC RAS, Professor at the Department of Computer Science and Computing Systems of the Institute of Automation and Computer Engineering of the SamSTU. Author of more than 350 scientific publications and textbooks. ORCID: 0000-0003-2199-9557; *petr.skobelev@gmail.com.*

Received April 16, 2025. Revised May 26, 2025. Accepted June 10, 2025.