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

Препринты ИПМ • Препринт № 43 за 2023 г.

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

А.В. Агеев, А.А. Богуславский, С.М. Соколов

Планирование заданий в бортовой вычислительной системе

Рекомендуемая форма библиографической ссылки: Агеев А.В., Богуславский А.А., Соколов С.М. Планирование заданий в бортовой вычислительной системе // Препринты ИПМ им. М.В.Келдыша. 2023. № 43. 27 с. https://doi.org/10.20948/prepr-2023-43
https://library.keldysh.ru/preprint.asp?id=2023-43

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

А.В. Агеев, А.А. Богуславский, С.М. Соколов

Планирование заданий в бортовой вычислительной системе

Москва – 2023

Агеев А.В., Богуславский А.А., Соколов С.М.

Планирование заданий в бортовой вычислительной системе

Рассматривается проблема рационального распределения ресурсов в бортовой вычислительной системе робототехнического комплекса. В качестве первого шага анализируется возможность использования алгоритмов онлайн планирования без вытеснения для распределенных систем – алгоритма кругового обслуживания (Round Robin). На примере решения задачи сегментации видеопотока демонстрируются особенности обработки задач для систем технического зрения реального времени. Решается проблема межузловой синхронизации данных с датчиков. Учитывается такая особенность бортовых ресурсов робототехнических комплексов, как необходимость использования связующего программного обеспечения в виде Robot Operating System. Для разработки планировщика задач используются язык программирования C++ и фреймворк ROS2, обеспечивающий асинхронное сетевое взаимодействие. Строится модель планирования и создается программное обеспечение, реализующее эту модель, для выполнения задач в распределенной среде с целью управления обработкой видеопотоков в системе технического зрения.

Ключевые слова: *планирование в реальном времени, синхронизация, Robot Operating System, ROS2, система технического зрения, связующее программное обеспечение.*

Aleksey Vladimirovich Ageev, Andrey Alexandrovich Boguslavsky, Sergey Michailovich Sokolov

Task scheduling in the onboard computer system

The problem of rational resource allocation in the on-board computing system of a robotic complex is considered. As a first step, the possibility of using online scheduling algorithms without preemptive for distributed systems, the Round Robin cyclic algorithm, is analyzed. To demonstrate the basic capabilities of the developed scheduler, a video stream segmentation task is used. The peculiarities of task processing for real-time vision systems are demonstrated. The problem of inter-node synchronization of sensor data is solved. A feature of on-board robotics resources, such as the need for a linking software in the form of Robot Operation System, is taken into account. To develop the task scheduler, the C++ programming language and the ROS2 framework, which provides asynchronous networking, are used. A scheduling model and software implementing this model are being built to perform tasks in a distributed environment in order to control the processing of video streams in a vision system.

Key Words: *real-time scheduling, synchronization, Robot Operating System, ROS2, computer vision, middleware systems.*

Введение

В последние годы в различных прикладных областях все большее востребование получают робототехнические комплексы (РТК) с повышенной степенью автономности, что, в свою очередь, ставит задачу оснащения этих комплексов адаптивными системами управления. Наиболее перспективным каналом информации о внешнем мире в РТК признается и уже становится зрительный канал – системы технического зрения (СТЗ) [1-3]. Адаптивная система управления, использующая в качестве канала информации о внешнем мире зрительные данные, требует большой производительности вычислительной системы. Кроме того, такие бортовые системы должны быть отказоустойчивыми и допускать оперативную перекомпоновку. В связи с этим целесообразно в качестве вычислительной основы выбирать распределенную вычислительную систему. Такой выбор позволяет повысить отказоустойчивость с помощью перераспределения нагрузки, а также наращивать производительность системы по мере необходимости. Признанным фактом является то, что разные типы задач могут быть эффективно решены на разных типах вычислительного оборудования. Например, задача вычисления оптического потока, которая используется в методе визуальной одометрии, может быть эффективно решена на графическом ускорителе. Указанные обстоятельства выдвигают необходимость и целесообразность создания бортовых распределенных гетерогенных вычислительных систем.

Подобные вычислительные ресурсы сами нуждаются в управлении. В условиях ограничений, накладываемых на бортовые вычислительные ресурсы, необходимо определять текущую ситуацию, в которой находится робототехнический комплекс, и, исходя из ситуации, перераспределять приоритеты между решаемыми задачами. На основе приоритетов принимается решение о выделении ресурсов. Для обеспечения такой возможности требуется ввести модуль планирования задач, назначением которого является управление использованием ресурсов распределенной системы. Этот программный модуль должен учитывать изменения ситуации (приоритетов задач), а также возможность выхода из строя части вычислительных модулей. Также необходимо учесть следующий факт. При построении робототехнических комплексов в качестве основы все более активно на практике применяется система ROS (Robot Operating System) [1]. Данная система предоставляет механизм асинхронной коммуникации между компонентами бортового программного обеспечения РТК. Создаваемый модуль должен обеспечивать встраиваемость, функционирование с учётом такого окружения.

В работе рассматривается проектирование и разработка системы планирования задач в распределённой вычислительной среде в рамках фреймворка ROS2 (современной модификации системы ROS). Предлагается модель планирования, формируется структура алгоритмического и программного обеспечения. Перед системой планирования ставится задача

управления распределенными ресурсами, соединенными сетью Ethernet. В качестве алгоритма планирования используется циклический алгоритм (Round Robin), что позволяет на текущем этапе исследований, для построения общей схемы, не углубляться в детали реализации более сложных алгоритмов планирования и выявить базовые возможности каркаса системы планирования: модели планирования и протокола обмена между планировщиком и исполнителем. Возможности предложенной системы планирования продемонстрированы на примере решения задачи сегментации видеопотока в составе бортовой системы информационного обеспечения РТК, включающей несколько вычислительно-управляющих блоков (ВУБ).

Архитектура вычислительной системы

В качестве экспериментальной, исследовательской базы используется система, состоящая из трех вычислительно-управляющих блоков (ВУБ – ЭВМ на архитектуре Intel x64). Три вычислительных блока связаны друг с другом локальной сетью с помощь технологии 1Gb Ethernet в топологии звезда. К каждому ВУБ подключена система датчиков (рис. 1).

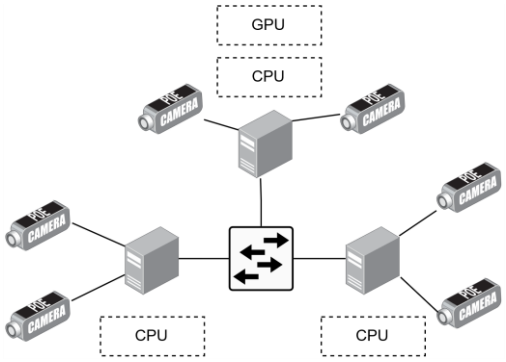


Рис. 1. Схема рассматриваемой гетерогенной распределенной системы обработки зрительных данных. У каждого вычислительно-управляющего блока указан тип используемого вычислителя

Предполагается, что подобная система может оснащаться и другими специализированными вычислителями. Система датчиков создает поток данных, который необходимо обрабатывать, используя доступные вычислительные ресурсы. Для разрабатываемой системы планирования вводятся следующие требования:

1. Необходимо использовать операционную систему общего назначения Linux (Ubuntu);
2. Система должна выполнять синхронизацию локальных часов всех узлов:
   – это необходимо для возможности объединения данных с разных датчиков, подключенных к разным вычислительным узлам, на основе временных меток;

3. В качестве механизма сетевого взаимодействия должна быть использована система ROS2;
4. Система гетерогенна:
   – ВУБ могут иметь разную архитектуру и сопроцессоры, таким образом, разные задачи могут иметь разную скорость решения на разных ВУБ;
5. Динамически изменяющаяся структура вычислительной сети:
   – ВУБ или элементы обеспечения коммуникационной сети могут выходить из строя.

В концепции ROS [1] процесс, который хочет участвовать в сетевом взаимодействии с другим процессом, называется узлом (node). ROS предлагает несколько шаблонов сетевого взаимодействия.

1. Тема (Topic) – это шаблон Издатель-Подписчик. Создается тема, на которую могут подписаться несколько узлов для получения данных. В качестве источника данных для темы могут выступать несколько Издателей;
2. Сервис (Service) – это механизм удаленного вызова процедуры (remote procedure call - RPC). Узел ROS публикует информацию о Сервисе, и другие узлы могут его вызвать;
3. Действие (Action) – то же самое, что Сервис, но с механизмом уведомления о стадии исполнения запроса. Используется для вызовов, которые исполняются продолжительное время.

ROS предоставляет язык Interface Definition Language (IDL), с помощью которого описываются типы передаваемых сообщений при сетевом взаимодействии. На основе IDL выполняется генерация кода интерфейсов, описывающих передаваемые сообщения между узлами ROS.

Для построения распределенной системы будем использовать ROS2. Ключевое отличие второй версии от первой заключается в том, что первая версия ROS использует централизованный брокер сообщений. Таким образом, в системе существует единая точка отказа. В отличие от первой версии, вторая версия ROS использует протокол OMG DDS (Object Management Group Data Distribution Service), который предоставляет механизм передачи сообщений без использования брокера. Протокол DDS использует широковещательные рассылки UDP для автоматического обнаружения участников и организации потока передачи данных между издателями и подписчиками [4].

Межузловая синхронизация времени

Одной из первоочередных задач распределенной системы управления РТК (СУ РТК) является синхронизация времени получения данных с систем датчиков. Система датчиков формирует поток данных. Синхронизация направлена на обеспечение того, чтобы данные, имеющие одинаковую

временную метку, соответствовали одному событию, которое произошло в окружающем мире. Выделяют два уровня синхронизации [2]:

- внутриузловая синхронизация данных с датчиков;
- межузловая синхронизация данных с датчиков.

Целью первого подхода является синхронное получение данных со всех датчиков, подключенных к одной машине. Например, этот подход применен в работе [3]. В качестве сети датчиков используется технология CAN. Камеры связаны триггерной шиной, которая подключена к микроконтроллеру. Микроконтроллер выступает в роли сервера точного времени протокола PTP. Он синхронизирует свои часы по GPS, а на триггерную шину передаёт периодический сигнал. На основе этого периодического сигнала датчики одновременно фиксируют состояние окружающего мира и по сети CAN доставляют их вычислительному блоку. Поскольку данные были созданы в один и тот же момент времени, то ВУБ устанавливает для них одну и ту же метку времени. Вычислительный блок синхронизирует свои локальные часы с микроконтроллером.

Подход межузловой синхронизации использует локальные часы ВУБ для проставления временной метки прибывающим данным, поскольку в этой ситуации датчики подключены к разным блокам и их невозможно синхронизовать триггерной шиной. Межузловая синхронизация имеет сложность применения, которая возникает из-за следующих свойств локальных часов:

- смещение (skew/offset) – часы на разных узлах показывают разное время;
- снос (drift) – часы на узлах идут с разной скоростью.

Первое свойство связано со сложностью настройки идентичного показания разных часов, и после настройки все равно будет отличие. Значит, необходимо добиваться синхронизации часов с некоторой приемлемой точностью. Вторая же проблема связана с устройством часов. Стандартные компьютеры оснащены часами на основе кварцевого генератора. Частота колебаний от одного экземпляра к другому может отличаться, а также зависеть от внешних условий. В результате этого разные кварцевые часы со временем начинают расходиться в своих показаниях.

Для поддержания синхронности часов на разных ВУБ используются протоколы синхронизации времени. Наиболее известные из них представлены в таблице 1. Применимость этих протоколов ограничена используемым типом программного обеспечения. Например, протоколы 2-4 эффективно могут функционировать в операционных системах (ОС) реального времени, при этом реализация должна учитывать особенности аппаратного обеспечения. Протоколы 1, 5, 6 могут работать в ОС общего назначения. Наша система использует операционную систему Ubuntu, а протокол 6 требует аппаратной

поддержки. Поэтому в качестве протокола синхронизации времени будем использовать протокол NTP.

Таблица 1.

Протоколы синхронизации времени

№	Протокол синхронизации времени	Экспериментальная точность
1	TEMPO [5]	$18-20\,ms$
2	RBS [6]	$6.29\,\mu s$
3	TPSN [7]	$16.9\mu s$
4	FTSP [8]	$2\mu s$
5	NTP [9]	$1-10\,ms$
6	PTP [10]	$1\mu s$

Для демонстрации необходимости синхронизации времени проведем эксперимент. Поскольку для синхронизации времени в дальнейшем мы будем использовать протокол NTP, то для проведения эксперимента воспользуемся его подходом к измерению смещения. Для измерения смещений NTP определяет on-wire protocol [9], опишем его:

Пусть имеется клиент *А* и сервер *Б*. Клиент хочет выполнить синхронизацию времени с сервером. Первое, что он должен сделать, – это получить временные отметки с сервера Б и сформировать собственные временные отметки по следующему алгоритму.

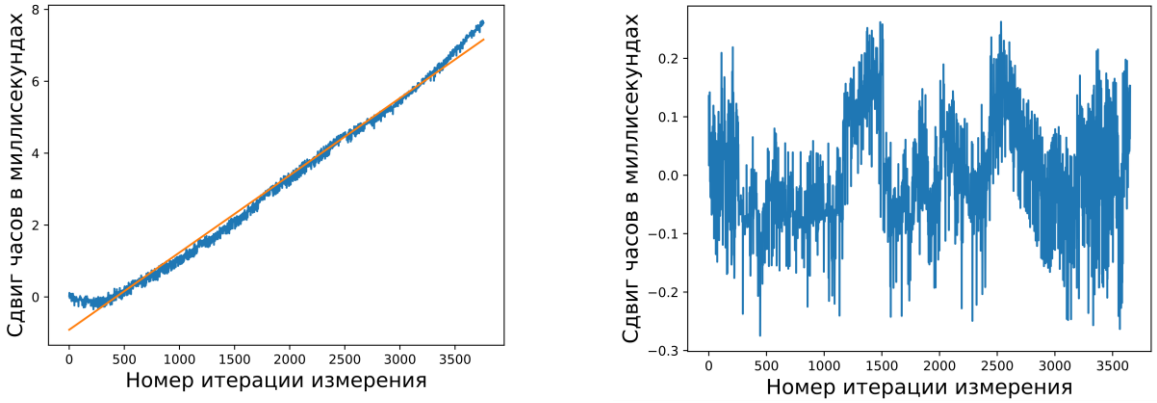
1. *А* выполняет отправку запроса с временной меткой t_1 серверу *Б;*
2. Сервер *Б,* получив запрос, формирует пакет с ответом, который содержит следующие временные метки: t_1 – метка, которую прислал клиент *А*; t_2 – время получения сообщения от клиента; t_3 – время отправки сервером *Б* ответа клиенту *А;*
3. Клиент *А,* получив ответ от сервера *Б,* формирует временную метку t_4;
4. Используя полученные временные метки, вычисляем статистики:

$$\begin{cases} t_2 = t_1 + \Theta + \delta \\ t_4 = t_3 - \Theta + \delta \end{cases} \rightarrow \begin{matrix} \Theta = \dfrac{(t_2 - t_1) + (t_3 - t_4)}{2}, \\ \delta = \dfrac{(t_4 - t_1) - (t_3 - t_2)}{2}, \end{matrix} \quad (1)$$

где Θ – сдвиг часов клиента относительно сервера; δ – задержка передачи в одну сторону; предполагается, что задержки в сети симметричные, тогда 2δ – RTT (Round Trip Time).

С использованием вышеописанного протокола для выполнения эксперимента по измерению времени были реализованы клиент и сервер. Считаем сервер эталоном точного времени, несмотря на то что в качестве источника времени он использует свои локальные часы. На рисунке 2 показаны результаты сравнения локальных часов двух вычислительных блоков по вышеописанному алгоритму. Измерения проводились в течение 5 часов. На

рисунке 2А видно, что с течением времени часы расходятся. Их необходимо синхронизировать.



(А) (Б)

Рис. 2. Результаты измерения различия времени между двумя вычислительными узлами. Измерения проводились с периодичностью 5 секунд в течение 5 часов. (А) Между узлами синхронизация не выполнялась. Кривая – результаты измерения. Прямая – линейная регрессия измерений. За данный период сдвиг часов составил 8 мс. (Б) Между узлами была включена синхронизация по протоколу NTP – сдвиг часов находится в пределе 0,3 мс.

Регрессионная прямая на рисунке 2А имеет наклон 0,002мс за секунду. Описательные статистики случайной величины, представленной на рисунке 2Б, показаны в таблице 2.

Проведенный эксперимент показывает действительную скорость расхождения локальных часов на имеющемся оборудовании. Например, при использовании высокоскоростных камер (200 Гц) расхождение локальных часов для межузловой синхронизации датчиков становится существенным через 2 часа, так как сдвиг часов уже превышает 2 мс, что близко к половине длительности интервала формирования кадра (5 мс). Такой сдвиг между локальными часами ВУБ может привести к ошибкам сопоставления кадров [2]. Таким образом, при длительной работе системы необходимо настраивать синхронизацию локальных часов для борьбы с их расхождением.

Таблица 2.

Измерения сдвига часов при синхронизации локальных часов ВУБ протоколом NTP

Характеристика	Значение (мс)
Число измерений	3653 шт.
Средняя	-0.001389
Медиана	-0.014527
Стандартное отклонение	0.085265

Характеристика	Значение (мс)
Минимальное значение	-0.274647
Максимальное значение	0.262616

Особенности планирования задач в системах технического зрения

Для принятия решения во время управления РТК необходимо организовать обработку непрерывного потока сенсорных данных. Эту обработку можно описать тремя этапами [11]:

1. Сбор и предобработка зрительных данных:
   - это этап инициализации, который необходимо проходить при следующих событиях:
     - запуск системы;
     - смена условий наблюдений;
     - потеря объекта интереса в процессе слежения;
   - применяемые алгоритмы: эквализация гистограммы, изменение контрастности, яркости;
   - большой поток данных, необходимо аппаратное ускорение;
   - необходимо хранение исходных данных с целью выполнения других операций над кадром, если текущий путь исполнения завершился неудачей;
2. Выделение первичных признаков:
   - выполняемые операции зависят от истории наблюдения:
     - первоначальный поиск объектов интереса;
     - слежение за найденным объектом интереса;
     - поиск объекта после его потери в текущем поле зрения;
   - применяемые алгоритмы: сегментация; вычисление оптического потока;
   - может потребоваться аппаратное ускорение для обработки большого количества кадров;
3. Формирование/использование моделей для распознавания объектов и сцен:
   - количество входных данных невелико, выполняется анализ различных комбинаций множества признаков по заранее заданным моделям.

Для извлечения информации из потоков сенсорных данных может потребоваться их объединение. В этом случае необходимо, чтобы временные метки данных были корректно проставлены (вопрос синхронизации был рассмотрен в пункте «Межузловая синхронизация времени»). Объединение кадров может понадобиться в случае, когда необходимо рассмотреть текущую сцену с разных ракурсов, а нужные камеры подключены к разным ВУБ.

Механизм управления обработкой при прохождении этих трех этапов должен учитывать накладываемые ограничения по времени. Если не удается

обрабатывать данные по той или иной причине, то необходимо уведомлять об этом объект, принимающий решение. Теория операционных систем реального времени (RTOS) предоставляет множество механизмов для планирования исполнения задач в многопроцессорных системах [12]. При этом предполагается, что время выполнения экземпляра периодической задачи (job) не превышает частоту генерации сенсорных данных. В нашем же случае это не так. Например, один из используемых алгоритмов сегментации – это метод сегментации Фальценсфальба на основе графа [13] – выполняется дольше интервала формирования очередного кадра камерой с частотой 30 Гц. Время исполнения алгоритма представлено в таблице 3. Применение сегментации не является самой целью, полученные сегменты подлежат дальнейшей обработке. Таким образом, время извлечения информации из одного кадра увеличивается с увеличением сложности информации (необходимого количества шагов для ее извлечения). Исходя из значений в таблице становится понятно, что со всеми кадрами система не справится, будут пропуски.

Здесь и необходимы приоритеты, на основе которых будет выделяться некоторая часть ресурсов для обработки части видеопотока. Остальные кадры будут пропущены. Важно, что этот процесс контролируемый. В этой ситуации принято решение использовать алгоритмы из области Cloud/Grid-вычислений [14, 15]. Эти алгоритмы выполняют заполнение расписания без передвижения уже запланированных задач.

Таблица 3.

Характеристика работы алгоритма Фальценсфальба на примере обработки изображений 1280x720.

	CPU версия	GPU версия (CUDA)
Время обработки одного кадра	660 мс	32 мс
Пик потребления памяти	155 Мб	ОЗУ: 664 Мб GPU: 471 Мб

Требования и модель системы планирования задач

Для планирования задач существует множество алгоритмов, которые различаются целями планирования и особенностью среды выполнения.

Разные системы планирования могут использовать одни и те же алгоритмы, но используемая модель, с помощью которой описывается распределенная вычислительная инфраструктура, может отличаться. Таким образом, алгоритмы планирования будут реализованы по-разному, вследствие чего результаты работы будут разными. Задачи могут быть двух типов:

- периодические задачи – обработка сенсорных данных (в нашем случае это видео кадры);
- недетерминированные задачи – это задачи, для которых нельзя определить момент времени, когда они будут поставлены или когда входные данные будут доступны для обработки задачи.

В системе планирования каждой задаче присваивается приоритет. Приоритеты могут быть представлены в разном виде:

- в числовой форме, как атрибут задачи;
- в количественной форме – количество экземпляров одного класса задачи, одновременно существующих в системе планирования.

Числовая форма приоритетов предполагает, что для каждого экземпляра задачи определяется численное значение важности. Задачи с более высоким приоритетом попадают в расписание первыми. Количественная форма приоритезации позволяет управлять объемом выделяемых ресурсов на обработку задачи. Как было написано выше, некоторые задачи могут иметь более длительное исполнение относительно интервала времени между поступлением сенсорных данных. Наличие одновременно нескольких копий задачи в системе планирования позволяет добиться параллельного выполнения задачи для нескольких измерений, получаемых от датчика. При планировании очередной копии задачи алгоритм должен учитывать время старта и завершения предыдущей запланированной копии, а также период публикации входных данных для этой задачи с целью корректного размещения в расписании и своевременного запуска копии задачи.

Разработанная система планирования оперирует экземпляром объекта класса задачи. Этот экземпляр является единицей планирования. Периодические и недетерминированные задачи отличаются друг от друга тем, что периодическая задача содержит информацию об периодичности формирования данных, а недетерминированная задача попадает на планирование уже с данными, которые необходимы для запуска задачи. Таким образом, разработанная система планирования заполняет расписание экземплярами классов задач. Расписание получается конечным, в отличие от алгоритмов операционных систем реального времени, в которых составляется бесконечное расписание на основании описания задач при инициализации системы. Система планирования не использует приоритеты для задач. Но одну и ту же периодическую задачу можно отправить на планирование несколько раз, тогда задача будет обрабатываться параллельно для нескольких экземпляров данных.

Другой аспект планирования — это возможность вытеснения запущенной задачи:

- планировщик без вытеснения запущенных задач;
- планировщик с вытеснением запущенных задач.

Под вытеснением понимается возможность приостановить выполнение одной задачи и вместо нее запустить другую. Разработанная система планирования является системой без вытеснения, она не позволяет заменить задачу на исполнителе раньше, чем текущая задача будет завершена.

Также необходимо учитывать то, что распределенная система может быть неоднородной. На разных исполнителях одна и та же задача может выполняться разное время.

В свази с тем, что система планирования спроектирована для многопользовательской операционной системы общего назначения, планировщик **не может контролировать** следующие аспекты:

- Процессы в ОС, не созданные планировщиком. Если кроме компонентов системы планирования в операционной системе запущены сторонние программы, то планировщик не может учесть использование ресурсов этими программами – количество активных процессов в ОС может отличаться от того количества, которое предполагает планировщик;
- Не предполагается **учет разделяемых ресурсов всей системы**. Например, если задача обращается к СУБД и была заблокирована на продолжительный период времени, то задача может опоздать к планируемому времени завершения;
- Частично асинхронная модель сети:
    - В общем случае сообщения, отправляемые в сеть, могут доставляться до получателя бесконечно долго;
    - Существуют интервалы времени, в которых возможно предположение о задержках передачи – α мс. В этом случае для надежной доставки сообщений протоколы используют механизм подтверждений. Если подтверждение не приходит в течение 2α, то сообщение отправляется повторно. Так, например, поступает протокол TCP.

Из-за неконтролируемых обстоятельств в системе планирования будут существовать задачи, которые опоздали к планируемому сроку выполнения. Из-за отсутствия возможности вытеснения задач невозможно освободить исполнителя от опоздавшей задачи. Добавление механизма вытеснения в распределенную систему планирования связано с трудностью определения причины опоздания задачи. Если задача действительно выполняется дольше, то это значит, что либо при конфигурации системы была установлена ошибочная оценка максимальной длительности исполнения задачи, либо на ВУБ работают сторонние программы, занимающие процессорное время. В ином случае задержка связана с аппаратной проблемой: отказала система коммуникации между двумя ВУБ или отказал сам ВУБ. В случае задержки выполнения ядро ВУБ будет занято на неопределенное время, в случае аппаратной проблемы постановка новых задач для данного ВУБ будет невозможна. Система планирования должна обладать механизмом уведомлений лица, принимающего

решение, о том, что в системе возникла опоздавшая задача и конкретный ресурс временно недоступен.

Алгоритм планирования задач Round Robin

Для первой реализации планировщика было принято решение использовать простой алгоритм планирования Round Robin. Такое решение позволит проработать нюансы реализации каркаса системы планирования в соответствии с моделью системы (описанной в пункте «Требования и модель системы планирования задач»). Использование простого алгоритма упростит поиск проблем и покажет базовые возможности реализации. После проработки разработанной системы можно будет переходить к другим алгоритмам для планирования задач в реальном времени. Блок-схема алгоритма планирования Round Robin представлена на рисунках 3-5.

Алгоритм Round Robin представляет собой циклический перебор списка процессов. Когда поступает запрос на планирование исполнения задачи, в качестве исполнителя назначается следующий исполнитель из списка. При получении очередной задачи этот список циклически перебирается по кругу. Но в соответствии с моделью также необходимо учитывать тот факт, что в расписание могут попадать задачи одного класса. Тогда необходимо сдвигать новую задачу в соответствии с частотой формирования сенсорных данных относительно уже запланированной копии задачи.

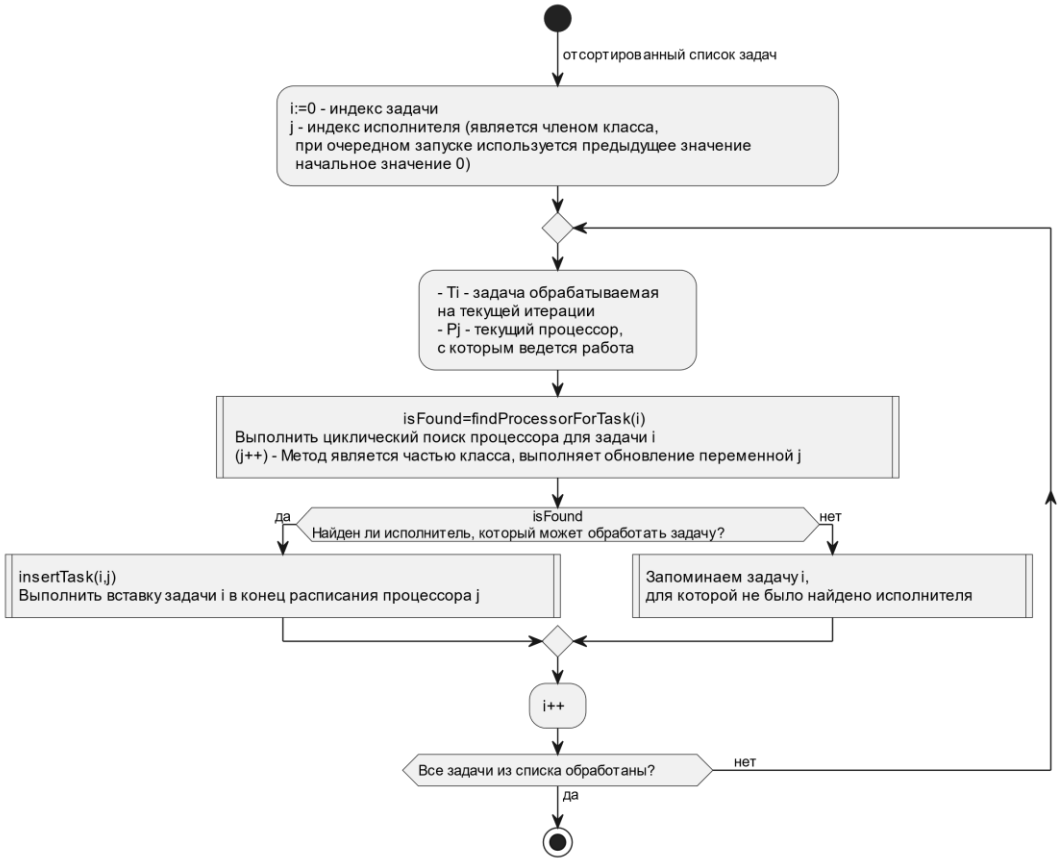


Рис. 3. Алгоритм добавления списка задач в расписание Round Robin

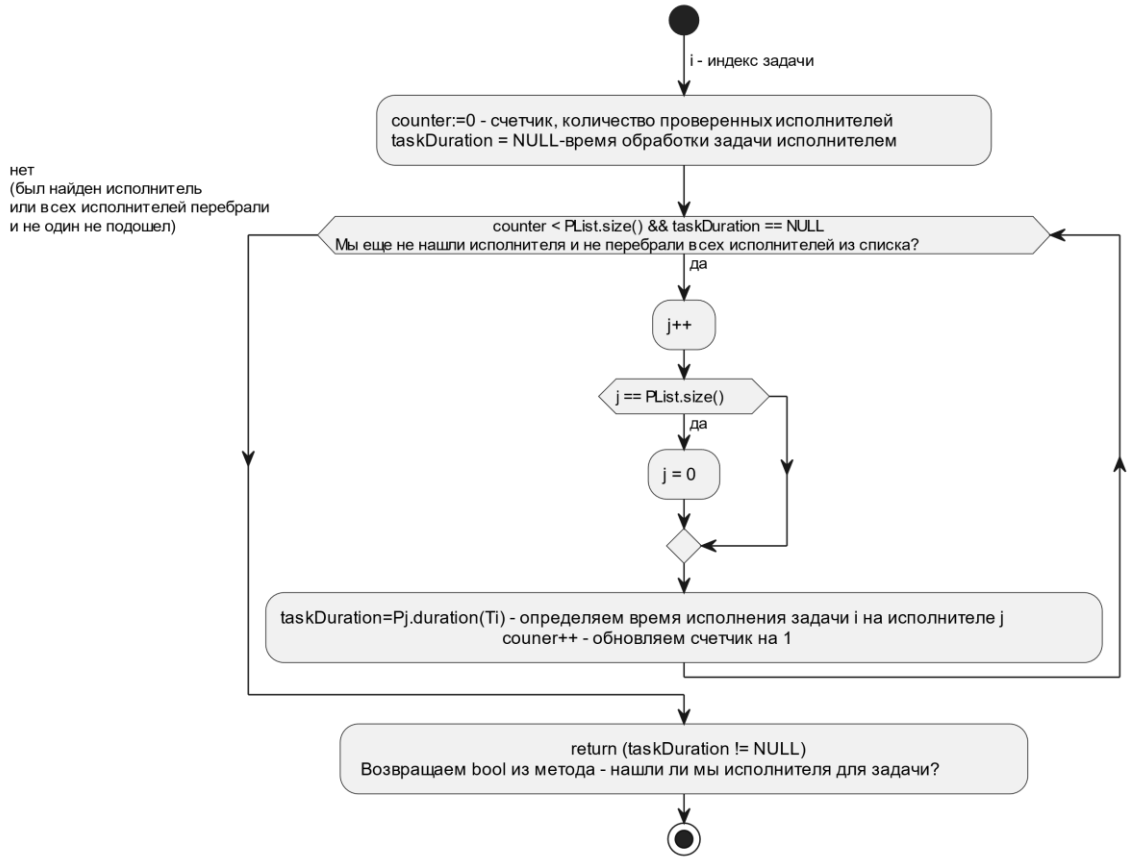


Рис. 4. Метод findProcessorForTask, используемый в блок-схеме на рис. 3. Метод выполняет поиск процессора, на котором может быть выполнена задача T_i

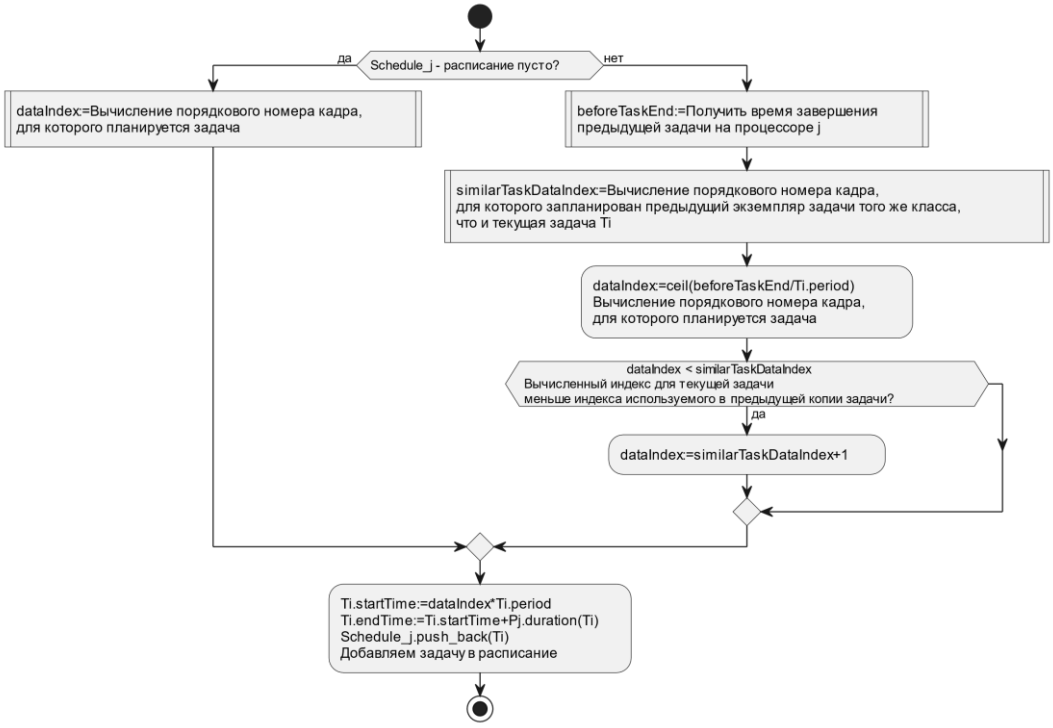


Рис. 5. Метод InsertTask, используемый в блок-схеме на рис. 3. Метод выполняет добавление задачи T_i в конец расписания для процессора j

Это необходимо учитывать, чтобы экземпляры задач одного класса обрабатывали разные сенсорные данные. Если задача недетерминированная, то этот учет выполнять не надо, так как данные уже сохранены внутри контекста задачи.

На рисунках 3-5 используются следующие обозначения:

1. P_i – процессор i, на котором может быть запущена задача (один поток исполнения);
2. $PList$ – список процессоров $\{P_i\}_{i=0..n}$, на которых запускаются запланированные задачи;
3. $PList[i]$ – процессор P_i в списке процессоров;
4. $PList.size()$ – размер списка процессоров;
5. T_i – задача i, которая имеет атрибуты:
   - $T_i.startTime$ – время старта задачи в расписании;
   - $T_i.endTime$ – время завершения задачи в расписании;
   - $T_i.period$ – частота формирования сенсорных данных для задачи;
6. $P_i.duration(T_i)$ – наихудшее время исполнения задачи T_i на процессоре P_i в миллисекундах. Если задача не может быть запущена на данном процессоре, то метод возвращает $Null$ – признак отсутствия возможности запуска задачи;
7. $Schedule_j[i]$ – список запланированных задач $\{T_i\}_{i=0..l}$ для процессора P_j.

Детали реализации системы планирования на базе ROS 2

Система планирования реализована на языке C++ и представляет собой асинхронную библиотеку. Для работы с асинхронными событиями и сетью используется фреймворк ROS2. Используя библиотеку планирования, пользователю необходимо реализовать несколько программ:

- планировщик задач;
- исполнитель – реализация задачи, которая должна планироваться и исполняться.

На рисунке 6 показана диаграмма последовательности, которая описывает логику работы механизма планирования. Сначала мы приведем краткое описание логики работы, а затем более подробное, с описанием сущностей и раскрытием всех шагов, представленных на рисунке 6.

Краткое описание механизма планирования

При реализации процесса, выполняющего планирование, необходимо создать объект сущности планировщика и реализовать классы задач (используя публичное наследование от интерфейса $ITask$), которые их будут описывать. Планирование выполняется для каждого экземпляра класса задачи отдельно. Для того чтобы добавить задачу в расписание, необходимо создать экземпляр

класса задачи и передать его планировщику, который, в свою очередь, добавит задачу в расписание.

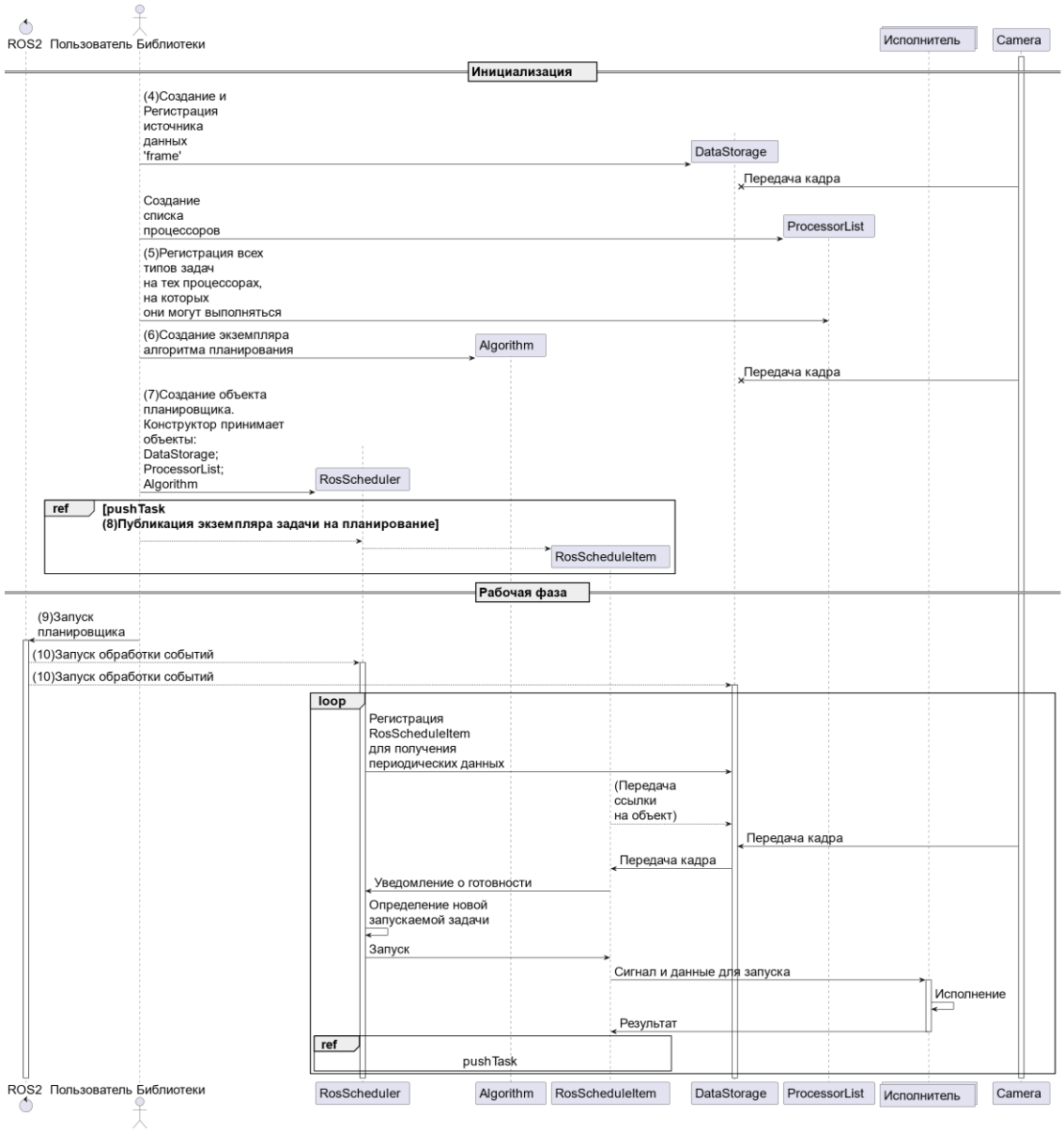


Рис. 6. Диаграмма последовательностей, описывающая работу планировщика. Блок ref.pushTask раскрыт на рисунке 7

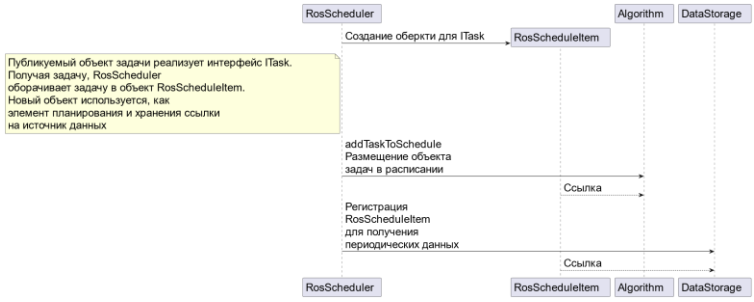


Рис. 7. Детализация реализации метода pushTask. В данном методе выполняется передача задачи объекту RosScheduler для выполнения запланированного запуска задачи

Каждый класс задачи имеет метод, который должен реализовывать логику предобработки результатов выполнения задачи, например, если задача периодическая, то этот метод должен создавать новый экземпляр текущего класса и отправлять его на планирование. Таким образом, будет обрабатываться периодическая задача, которая будет занимать только одно ядро процессора. Если задача обрабатывается дольше, чем интервал формирования сенсорных данных, то имеет смысл в расписание добавлять несколько экземпляров одной и той же задачи, тогда одновременно будут обрабатываться несколько кадров, соответственно, будет занято несколько ядер процессора.

Реализация процесса исполнителя заключается в использовании объекта $rclcpp::Service$ для ожидания получения данных и сигнала о старте исполнения задачи. Процессов исполнителей одной и той же задачи на одном вычислительном блоке должно быть запущено столько, сколько ядер на процессоре. Эти исполнители должны быть запущены на всех ВУБ, которые могут исполнять данную задачу. Например, пусть имеется 4 задачи и 2 ВУБ с 8-ядерными процессорами, каждая задача может быть исполнена на каждом ВУБ. Тогда всего необходимо запустить 64 процесса ($4 \cdot 8 \cdot 2$). Все эти процессы находятся в спящем режиме, и Планировщик (посредством ROS) отсылает команду на запуск. Планировщик самостоятельно контролирует, чтобы на одном ВУБ не было активно больше процессов, чем ядер, путем отслеживания состояния задач.

Подробное описание механизма планирования

Планировщик представляет из себя централизованную систему, которая состоит из следующих сущностей (рис. 8):

- Алгоритм планирования (интерфейс $IAlgorithm$) – алгоритм получает на вход объект $Schedule$ и список задач ($RosScheduleItem$), которые необходимо запланировать. Выполняет заполнение расписания в соответствии с реализованным алгоритмом планирования;
- Расписание ($Schedule$) – это сопоставление списка процессоров списку задач. Какая задача, когда будет запущена на каком процессоре;
- Процессор ($Processor$) – это сущность, описывающая исполнителя. Каждый $Processor$ ассоциирован с одним потоком исполнения операционной системы. Содержит атрибуты:
  - Название процессора – Название узла ROS;
  - Оценка худшего времени работы для каждой задачи, которые могут быть исполнены на данном процессоре;
- $DataStorage$ – Временное хранилище сенсорных данных. Эта сущность собирает данные со всех датчиков системы. Поток данных от конкретного датчика – это пронумерованная последовательность экземпляров данных. Экземпляры задач ($RosScheduleItem$): после планирования регистрируют подписку на получение экземпляра данных под определенным номером;

- Описатель задачи ($ITask < RosService >$) – объект, описывающий задачу:
    - Тип задачи: периодичная; недетерминированная;
    - Периодичность задачи (для периодических задач);
    - Название источника данных;
    - Название задачи;
    - Метод обработки результатов - данный метод вызывается после завершения исполнения задачи. Метод в качестве параметра принимает результат исполнения задачи. Он принимает решение о постановке новой задачи на исполнение. Метод является интерфейсным, должен быть реализован для каждого типа задачи;
- Передатчик на исполнение ($Executor < RosService >$) – данный объект используется для того, чтобы отправить сигнал о запуске конкретной задачи и ее входные данные конкретному исполнителю;
- Исполнитель – это отдельный процесс, в котором используется объект $rclcpp :: Service < RosService >$, для приема запроса на исполнение и возврата результатов;
- Логические часы – это глобальный объект, который ведет учет времени в системе. Представляет собой постоянно растущий счетчик событий. Для каждого класса события известны характеристики. Когда возникает событие, то логические часы обновляют свое значение (концепция логических часов представлена на рис. 9). Для события определены следующие характеристики:
    - предыдущий момент времени (по логическим часам) возникновения события;
    - периодичность возникновения событий данного класса.

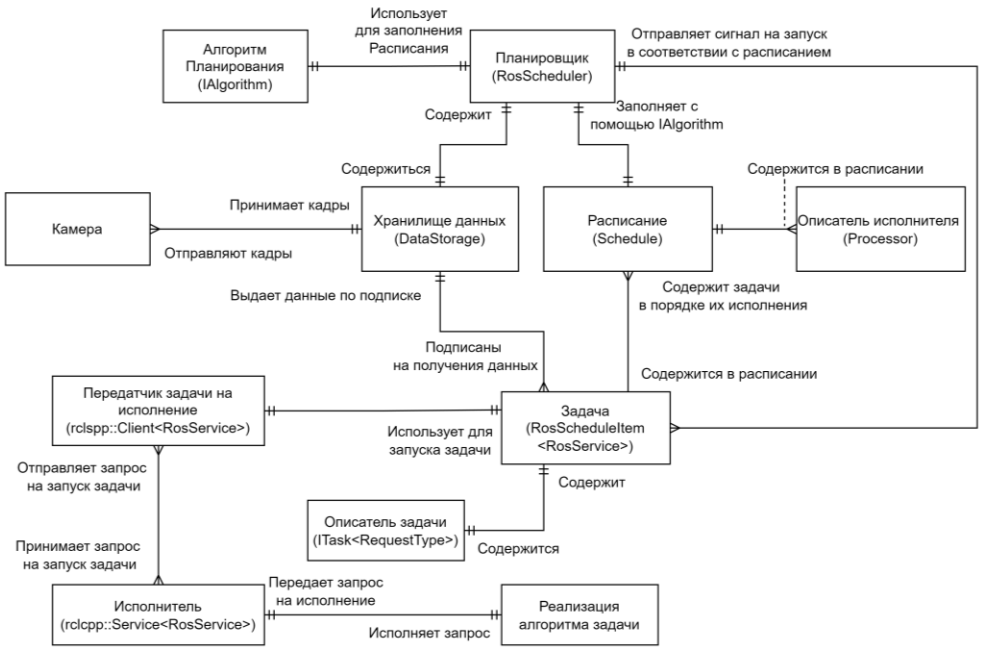


Рис. 8. ER-Диаграмма сущностей планировщика

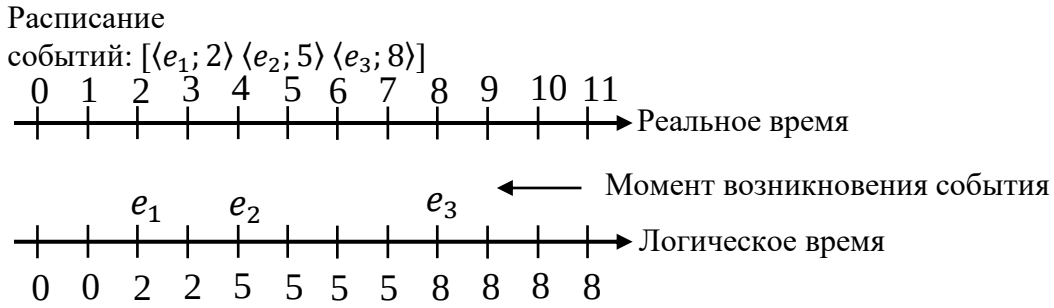


Рис. 9. Сравнение хода логических часов с часами, измеряющими реальное время. Логические часы увеличивают значение счетчика в соответствии с временной меткой, предписанной событию в расписании. Расписание событий представляет собой упорядоченное множество пар: ⟨событие, логическое время возникновения⟩, порядок определяется по второму полю пары (логическое время)

Рассмотрим этап инициализации планировщика (рис. 6) – последовательность действий, которые необходимо предпринять для запуска механизма планирования:

1. Необходимо реализовать интерфейс $ITask < RosService >$ для всех типов задач;
2. Необходимо создать объект $rclcpp::Node$ – для вхождения в сеть ROS2;
3. Необходимо создать экземпляр $DataStorage$ и зарегистрировать источник данных для каждой задачи, которые были созданы на шаге 1;
4. Для каждого исполнителя необходимо завести описание в виде сущности $Processor$; Исполнитель – это ROS2-Сервис ($rclcpp::Service < RosService >$), который представляет собой однопоточный процесс в операционной системе;
5. Выбрать алгоритм планирования и создать его экземпляр;
6. Создать объект планировщика ($Scheduler$) и зарегистрировать в нем задачи из шага 1;
7. Создать экземпляры задач из шага 1 и передать их планировщику на планирование (формируется стартовое расписание);
    a. Экземпляров одного класса задачи может быть несколько. Один экземпляр задачи при планировании занимает одного исполнителя. Это место распределения ресурсов между периодическими задачами.
    b. При передаче задачи на планирование она немедленно добавляется в расписание в соответствии с алгоритмом планирования, выбранным на шаге 5. После добавления задачи в расписание в соответствии с ее планируемым временем старта планировщик выполняет регистрацию получения экземпляра сенсорных данных. Временные метки в расписании устанавливаются в соответствии с текущим логическим временем;

После создания сущностей и регистрации задач в объекте планировщика, переходим к рассмотрению рабочей фазы планировщика (рис. 6):

8. Запуск планировщика ($rclcpp::spin(node)$);
9. Во время запуска планировщик инициализирует объекты типа $rclcpp::Client$ для каждого исполнителя и каждой задачи. Проверит их доступность. Если какой-то объект просигналит о невозможности запуска, то планировщик немедленно будет остановлен.
10. Планировщик находится в состоянии сна и реагирует на асинхронные события:
    a. Приход сенсорных данных;
    b. Завершение выполнения задачи;
11. Если произошло событие «Приход сенсорных данных», то оно приходит в объект $DataStorage$. И тогда происходит следующее:
    a. Двигается логическое время в соответствии с периодичностью источника данных;
    b. Запускается механизм поиска задачи, подписанной на получение данного экземпляра данных. Если такая задача найдена, то этой задаче передаются эти данные;
    c. Выполняется обход по статусам задач среди кандидатов на запуск (это задачи, которые стоят первыми в расписании), каждую готовую задачу планировщик запускает;
12. Если произошло событие «Завершение выполнение задачи», то оно происходит в соответствующем экземпляре класса $RosScheduleItem < RosService >$:
    a. Двигается логическое время в соответствии с расписанием завершения задачи;
    b. Вызывается «метод обработки результата» объекта $ITask < RosService >$. Результат работы метода – это, возможно, добавление копии текущей или новой задачи в расписание.
    c. Текущая задача удаляется из расписания;
    d. Запускается обход задач (Смотри шаг 11.c).

Результаты работы планировщика для распределенной обработки задачи сегментации видеопотока

Для демонстрации работы системы планирования будет использоваться задача сегментации видеопотока. Рассмотрим следующий сценарий:

1. Имеется два вычислительных блока: $node1$; $node2$, на которых будут работать процессы, выполняющие сегментацию кадров методом Фальценсфальба;
2. Планировщик запускается в едином экземпляре на узле $node1$;

3. Каждый процесс-исполнитель публикует себя как ROS-сервис, название которого задается в конфигурационном файле исполнителя. Также планировщику в конфигурационном файле задается название всех сервисов-исполнителей;
4. Сначала запускаются все исполнители;
5. Затем запускается планировщик, который подключается ко всем исполнителям, используя названия ROS-сервисов. Если планировщик не сможет подключиться хотя бы к одному из сервисов-исполнителей, то он принудительно завершит работу;
6. Затем на узле $node1$ запускается процесс-источник кадров видеопотока. Этот процесс будет брать данные из видеофайла и выдавать кадры в систему с частотой 30 Гц. Кадры имеют разрешения 1280x720 пикселей;
7. Кадры видеопотока будут публиковаться в ROS-тему «frame», на которую подписывается планировщик;
8. Каждый раз, когда планировщик будет получать очередной кадр, будет запущен процесс поиска задачи в расписании, которая ожидает данный конкретный кадр;
9. Если запланированной задачи для кадра нет, то обработка кадра пропускается;
10. Если найдена задача, то кадр сохраняется внутри объекта задачи (RosScheduleItem) и задача переводится в состояние готовности;
11. Когда придет время запуска задачи, то исходные данные (кадр) будут переданы соответствующему ROS-сервису, что будет сигнализировать о необходимости запуска обработки;
12. По завершении обработки ROS-сервис обязан направить обратно планировщику результаты обработки. В свою очередь, планировщик выполняет отправку результатов сегментации в тему «segments». Операцию публикации результатов выполняет реализация интерфейса $ITask$, когда тот получает сигнал о завершении обработки.

Вышеописанный сценарий порождает несколько сущностей ROS. Схема взаимодействия этих сущностей представлена на рисунке 10.
Описание сущностей рисунка 10:

- video_source – процесс, получающий кадры из видеофайла и отправляющий их в тему «frame»;
- video_viewer – процесс (программа), предназначенный для просмотра данных с датчиков и результатов сегментации. Данные с датчика берутся из темы «frame», а результаты сегментации берутся из темы «segment»;
  scheduler – получает данные для сегментации из темы «frame», которые поступают в объект DataStorage. Если в расписании существует задача, требующая данный экземпляр данных, то эти данные передаются задаче, если нет, то данные забываются. После завершения исполнения задачи

результаты возвращаются в планировщик, который их публикует в тему «segment»;

- $node1p1, node1p2, \ldots\ nodeIpK\ \ldots\ node2p8$ – это название ROS-сервисов, которые запущены на соответствующих узлах. nodeIpK – сервис, запущенный на узле I, имеющий номер K.

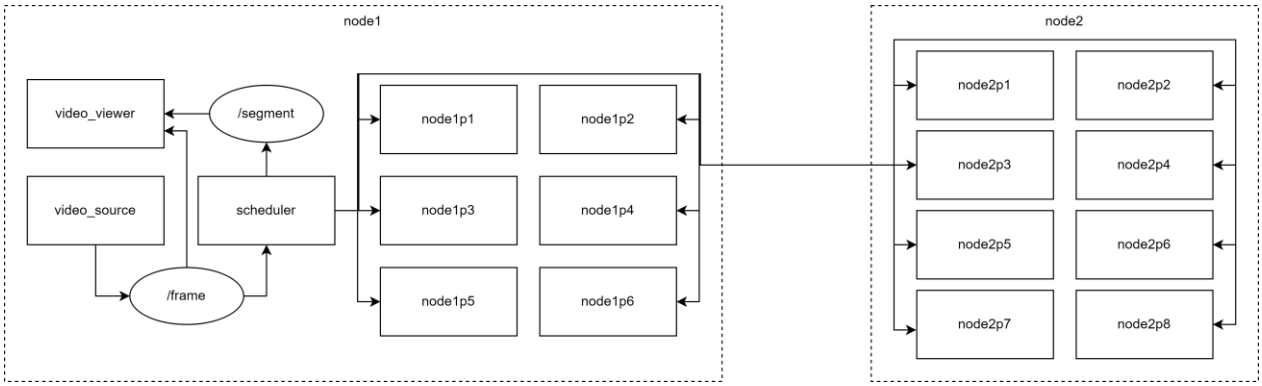


Рис. 10. Схема взаимодействия компонентов системы с планировщиком для задачи сегментации видеопотока. Обозначения: прямоугольник – процесс, узел ROS; овал – тема ROS; стрелки обозначают потоки данных. Если два процесса соединены стрелками через овал, то это означает использование шаблона сетевого взаимодействия Издатель-Подписчик. Прямое соединение процессов стрелкой обозначает использование шаблона Клиент-Сервер

Для реализации этого сценария используются два ВУБ, которые имеют 8-ядерный процессор. При этом на одном ВУБ запущено 8 исполнителей задачи, а на другом 6. Для того чтобы планировщик использовал все ресурсы системы (14 процессов), мы выполним планирование 14 задач для сегментации видеопотока. Контекст задачи ($ITask$) по завершении ее исполнения будет отправлять запрос планировщику на планирование задачи сегментации. Этот запрос будут отправлять каждый экземпляр задачи, и таким образом будет достигнута обработка кадров видеопотока с использованием всех доступных ресурсов. Результат исполнения сценария показан на рисунке 11.

На рисунке 11 используются следующие обозначения. На диаграмме 11Б каждый прямоугольник, описывающий исполняющуюся задачу, разделен на три части. Левая часть (оранжевого цвета) описывает время, затраченное на получение данных исполнителем. Центральная часть (синего цвета) показывает интервал времени, в течение которого исполнялось задание. Правая часть (красного цвета) описывает время, затраченное на передачу результата решения задачи планировщику.

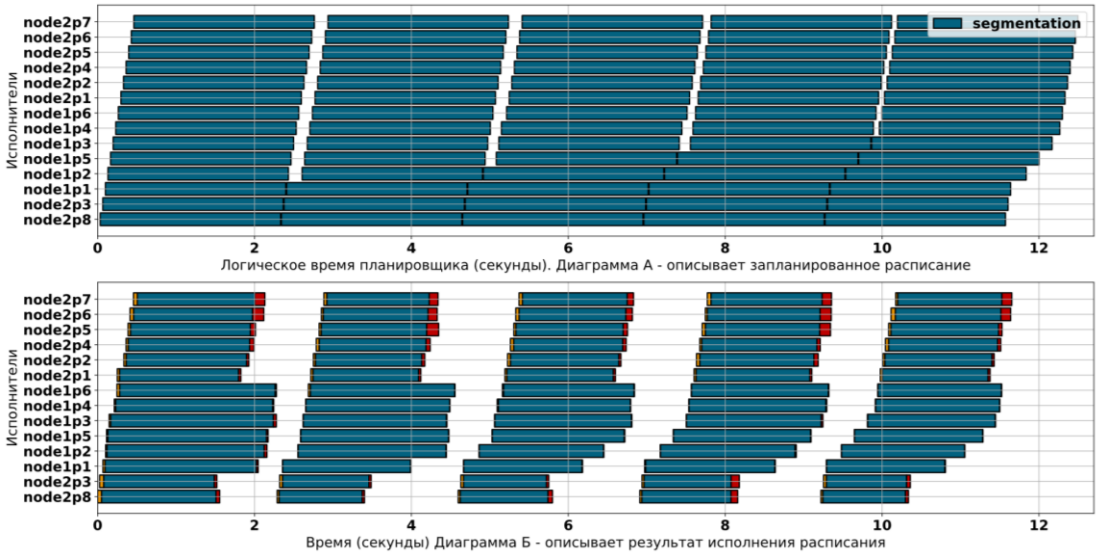
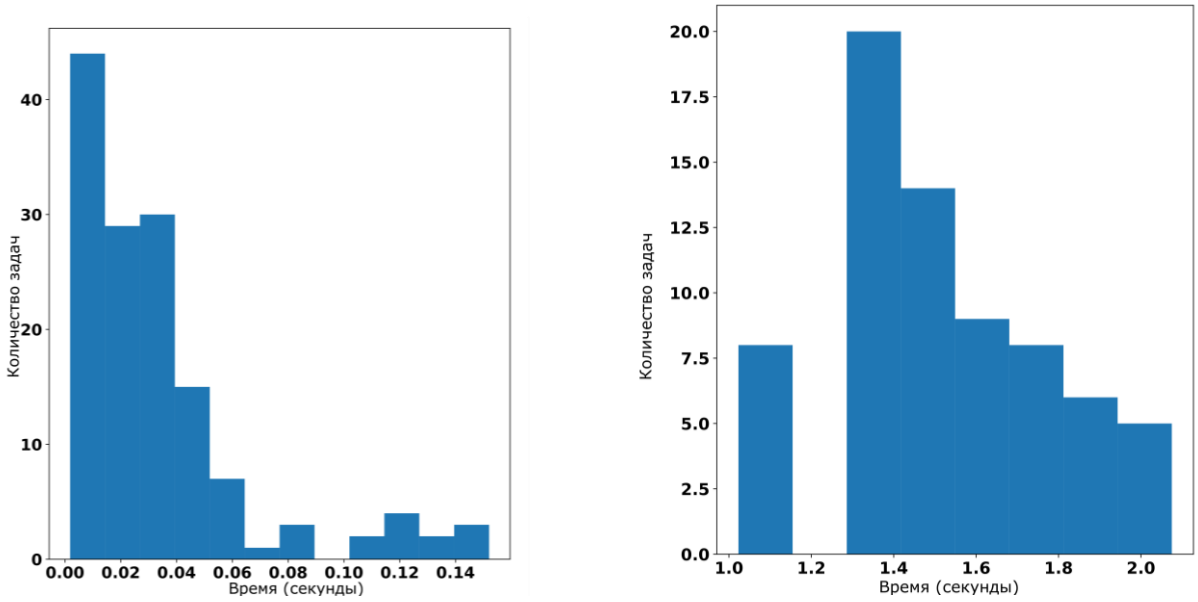


Рис. 11. (А) Диаграмма Ганта, описывающая расписание, созданное планировщиком с использованием алгоритма Round Robin. (Б) Диаграмма Ганта, описывающая результат исполнения расписания. На диаграмме (А) используется логическое время. На диаграмме (Б) используется реальное нормализованное время

Рисунок 11 описывает результаты работы планировщика. Видно, что интервалы между задачами на диаграмме 11Б значительно больше интервалов на диаграмме 11А. Это явление объясняется следующим: для описания задачи используется характеристика «наихудшее время исполнения» (worst case execution time, WCET), которое больше или равняется фактическому времени исполнения задачи. При планировании задачи они не могут стать ближе друг к другу, чем:

$$DataPeriod \cdot ceil\left(\frac{WCET}{DataPeriod}\right) - WCET, \qquad (2)$$

где $DataPeriod$ – периодичность публикации сенсорных данных. Поскольку WCET – это оценка, а большая часть задач выполняется быстрее, поэтому фактические интервалы между задачами больше. Также, как было указано в пункте «Требования и модель системы планирования задач», расписание заполняется постепенно, по факту прибытия запроса на планирование задачи. Сенсорные данные в хранилище попадают только, если для некоторого экземпляра сенсорных данных уже есть запланированная задача. Если мы начали планировать задачу и только что пришли данные, то они будут отброшены, поскольку планирование не завершено, и в итоге задача будет ожидать следующий экземпляр сенсорных данных. Из-за этого возникают промежутки простоя между задачами в расписании. Фактическое распределение времени выполнения задачи сегментации изображения методом Фальценсфальба представлено на рисунке 12.



(А) (Б)

Рис. 12. (А) Распределение времени передачи изображений между процессами (как по сети, так и внутри ОС) и (Б) времени исполнения задачи сегментации. Значения случайных величин зависят как от входных данных, так и от неконтролируемых событий внутри операционной системы

Использовать метрику WCET по ее определению в контексте ОС общего назначения некорректно, так как операционная система Linux (Ubuntu) может отложить процесс на неопределенный срок. Поэтому под «наихудшим временем исполнения» мы понимаем следующее: максимальное время работы алгоритма (задачи), которое зависит от входных данных, при условии, что процесс не будет отложен операционной системой. Таким образом, система планирования должна быть готова к тому, что в ней будут существовать задачи, которые опоздали к сроку завершения.

Также расписание и процесс выполнения точно не совпадают. Это связано с тем, что в качестве измерителя времени используются внешние события, в данном конкретном примере таким внешним событием является получение очередного видеокадра. На эту неточность влияет точность удержания частоты генерации кадров, а также задержки, возникающие в процессе передачи кадров между процессами.

Также интересным является рассмотрение вопроса, дает ли распределенная обработка увеличение количества обрабатываемых кадров в секунду. Ответ: да (рис. 13), но поскольку мы рассматриваем задачи, выполняющиеся продолжительное время, порядка 2 секунд, то график строится как «количество кадров за 2 секунды». На рисунке 13 можно обратить внимание на провалы в кадровой частоте. Они связаны с тем, что при вычислении кадровой частоты просматривается интервал в 2 секунды с шагом в 2 секунды, а процесс выполнения задач циклически сдвигается. Таким образом, существуют просматриваемые интервалы, в которых не завершилась ни одна задача. На

рисунке 14 представлен один из таких интервалов (выделен красным прямоугольником), который на левом графике рисунка 13 отображен в виде красной точки на 30-м двухсекундном интервале, на котором не была выполнена ни одна задача. Для двух узлов (рис. 13, правый график) ситуация аналогична, но в колебания на графике также вносят вклад дополнительные недетерминированные задержки передачи изображения по сети. Добавление дополнительных вычислительных узлов повышает кадровую частоту обработки видеопотока.

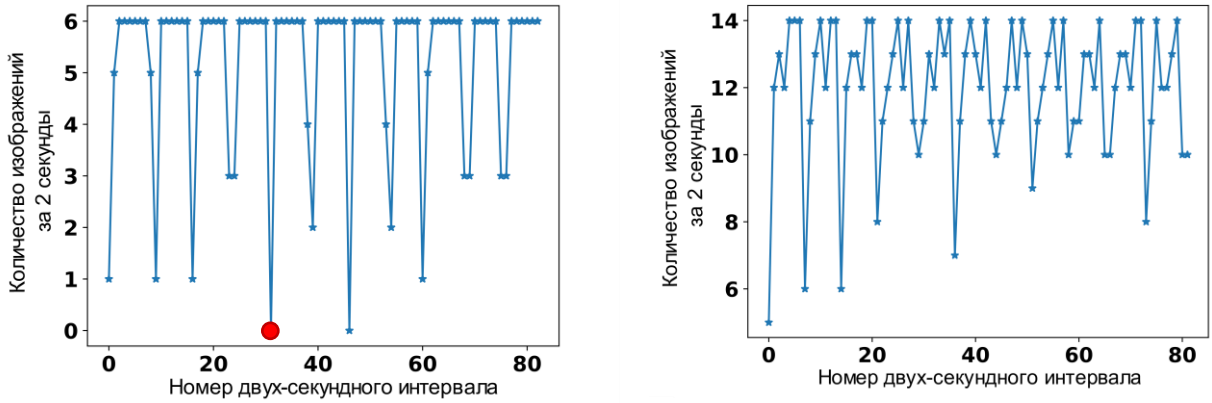


Рис. 13. Кадровая частота результатов сегментации (количество кадров каждые 2 секунды). Слева – при обработке видеопотока на одном узле. Справа – при обработке видеопотока на двух узлах

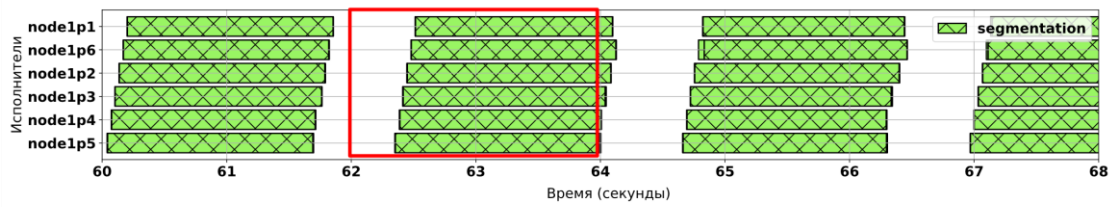


Рис. 14. Демонстрация двухсекундного интервала на диаграмме Ганта, в котором не была выполнена ни одна задача. Такой интервал возникает из-за постоянного циклического сдвига периодических задач в расписании. Этот интервал на графике кадровой частоты (рис. 10) обозначен красной точкой

Заключение

Разработаны модель и программное обеспечение для планирования вычислений в распределенной вычислительной системе. Реализованный планировщик составляет/модифицирует расписание при получении запросов на планирование, затем в соответствии с расписанием запускает задачи на исполнение. Целевой платформой системы планирования является операционная система общего назначения Linux.

Система позволяет планировать как детерминированные, так и недетерминированные задачи, поддерживает неявную зависимость между заданиями – новая задача добавляется в систему на основе результата обработки предыдущей. На текущем этапе исследования для построения общей схемы планирования был использован алгоритм планирования Round Robin.

Возможности предложенной системы планирования продемонстрированы на примере задачи сегментации видеопотока в составе бортовой системы информационного обеспечения (разрешение 1280x720, частота 30 Гц). В основе алгоритма сегментации использован графовый метод Фальценсфальба. Продемонстрировано, что добавление вычислительного узла позволяет поднять пропускную способность системы. Для графового алгоритма сегментации существует реализация для GPU, использование которой также позволит поднять пропускную способность вычислений.

В дальнейшем мы планируем модифицировать модель планирования с целью предоставления возможности планирования многопоточных задач, а также задач для гетерогенных вычислений (например, для CPU+GPU). Также необходимо исправить проблему единой точки отказа.

Список источников и литературы

1. Macenski S. et al. Robot Operating System 2: Design, architecture, and uses in the wild // Science Robotics. – 2022. – Т. 7. – №. 66.
2. Liu S. et al. The Matter of Time – A General and Efficient System for Precise Sensor Synchronization in Robotic Computing // arXiv preprint arXiv:2103.16045. – 2021
3. Fregin A. et al. Building a computer vision research vehicle with ROS // Proc. of the ROSCon 2017. – 2017. – Т. 21.
4. OMG Data Distribution Service (DDS) Version 1.4 URL: https://www.omg.org/spec/DDS/1.4/PDF
5. Gusella R., Zatti S. The accuracy of the clock synchronization achieved by TEMPO in Berkeley UNIX 4.3 BSD // IEEE transactions on Software Engineering. – 1989. – Т. 15. – №. 7. – С. 847-853.
6. Elson J., Girod L., Estrin D. Fine-Grained Network Time Synchronization Using Reference Broadcasts // 5th Symposium on Operating Systems Design and Implementation (OSDI 02). – 2002.
7. Ganeriwal S., Kumar R., Srivastava M. B. Timing-sync protocol for sensor networks // Proceedings of the 1st international conference on Embedded networked sensor systems. – 2003. – С. 138-149.
8. Maróti M. et al. The flooding time synchronization protocol // Proceedings of the 2nd international conference on Embedded networked sensor systems. – 2004. – С. 39-49.
9. Mills D. L. Internet time synchronization: the network time protocol // IEEE Transactions on communications. – 1991. – Т. 39. – №. 10. – С. 1482-1493.

10. Correll K., Barendt N., Branicky M. Design considerations for software only implementations of the IEEE 1588 precision time protocol // Conference on IEEE. – 2005. – Т. 1588. – С. 11-15.
11. Соколов С. М., Богуславский А. А., Романенко С. А. Программно-аппаратные средства для бортовых систем информационного обеспечения подвижных средств с использованием СТЗ // Известия Южного федерального университета. Технические науки. – 2020. – №. 1 (211). – С. 246-257.
12. Davis R. I., Burns A. A survey of hard real-time scheduling for multiprocessor systems // ACM computing surveys (CSUR). – 2011. – Т. 43. – №. 4. – С. 1-44.
13. Felzenszwalb P. F., Huttenlocher D. P. Efficient graph-based image segmentation // International journal of computer vision. – 2004. – Т. 59. – С. 167-181.
14. Bittencourt L. F. et al. Scheduling in distributed systems: A cloud computing perspective // Computer science review. – 2018. – Т. 30. – С. 31-54.
15. Topcuoglu H., Hariri S., Wu M. Y. Performance-effective and low-complexity task scheduling for heterogeneous computing // IEEE transactions on parallel and distributed systems. – 2002. – Т. 13. – №. 3. – С. 260-274.

Оглавление