Санкт-Петербургский государственный университет

ПЕТРОВ Валентин Юрьевич

Выпускная квалификационная работа

Гибридный симулятор мультиагентной системы с виртуальной визуализацией и подключением реальных объектов

Уровень образования: магистратура

Направление *02.04.03 «Математическое обеспечение и администрирование информационных систем»*

Основная образовательная программа *СВ.5665.2019 «Математическое обеспечение и администрирование информационных систем»*

Научный руководитель:
профессор кафедры системного программирования, д.ф.-м.н., Граничин О. Н.

Рецензент:
старший научный сотрудник СПбНИУ ИТМО, к.т.н., Маргун А. А.

Санкт-Петербург
2022

Saint Petersburg State University

Valentin Petrov

Master's Thesis

Hybrid simulator of a multi-agent system with virtual visualization and real objects connectivity

Education level: master

Speciality *02.04.03 "Software and Administration of Information Systems"*

Programme *CB.5665.2019 "Software and Administration of Information Systems"*

Scientific supervisor:
Sc.D, prof. O.N. Granichin

Reviewer:
Senior Resercher at "ITMO University", Ph.D, A.A. Margun

Saint Petersburg
2022

Оглавление

Введение

В современном мире всё более популярной становится концепция децентрализации: вместо монолитных архитектур разработчики всё чаще строят свои приложения на микросервисах, вычисления всё чаще производят не на локальных серверах, а на удаленных и распределенных облаках. На этом фоне стремительно набирает популярность задача мультиагентного взаимодействия групп или роев роботов, так как существует ряд задач, выполнение которых можно достичь гораздо быстрее, используя сеть распределенных роботов [15].

Проверка гипотез и алгоритмов на реальном рое является не самым простым занятием. Необходимо иметь достаточное количество роботов, запрограммировать каждого из них, найти достаточно пространства для опыта, настроить среду, в которой будут находиться и с которой будут взаимодействовать роботы. Кроме того, в случае, если необходимо внести исправление в код логики робота, приходится делать это отдельно для каждого устройства.

Компромиссным решением является использование не реального роя, а виртуального. Современные симуляторы способны реалистично моделировать физику, при этом, упрощаются рутинные процессы, связанные с настройкой роботов и их окружения.

Большой интерес имеет задача достижения консенсуса в группах роботов с переменной топологией. В них участник может общаться лишь с несколькими узлами-соседями, при этом, эти соседи непостоянны, и на смену одних приходят другие, а каждому агенту этой сети неизвестна топология всей сети. С достижением консенсуса в таких сетях эффективно справляется «протокол локального голосования» [2].

На текущий момент было предложено решение [21], [5] задачи о реализации группового движения роботов на источник светового излучения в среде с препятствиями в симуляторе Webots. Алгоритм успешно справляется с поставленной задачей и хорошо себя показывает на небольших группах роботов, до 25 единиц. Однако, при попытке увеличить размер группы роботов на порядок, система оказывается довольно

неэффективной по времени. Это связано со спецификой работы симулятора Webots, процессы которого синхронны и последовательны, в то время как реальные роботы действуют асинхронно.

Таким образом, для задачи группового взаимодействия роботов хочется предложить решение, основанное на симуляторе Webots, в рамках которого роботы могли бы осуществлять асинхронное мультиагентное взаимодействие. При этом, система должна иметь визуализацию, и позволять моделировать физические процессы.

Постановка задачи

Целью данной работы является реализация гибридного симулятора, имеющего визуализацию и способного производить асинхронное мультиагентное взаимодействие внутри группы роботов. Для достижения этой цели были сформулированы следующие задачи:

1. Провести обзор предметной области
2. Определить требования для предлагаемого решения
3. Предложить и реализовать прототип гибридного симулятора с мультиагентным взаимодействием на основе существующих сервисов
4. Апробация предложенного симулятора с виртуальными роботами на примере реальной задачи о мультиагентном взаимодействии. Система должна эффективно работать с роем, достигающим 100 роботов

1. Обзор предметной области

1.1. Мультиагентные системы

Мультиагентными системами (МАС) являются такие системы, в рамках которых происходит взаимодействие множества интеллектуальных процессов, которые называются агентами. МАС являются эффективным инструментом для создания алгоритмов, связанных с организацией децентрализованного взаимодействия в статических и динамических средах. Такие системы часто примменяются при решении задач, которые сложно или невозможно решить, если использовать одну монолитную систему.

Агенты в МАС имеют ряд отличительных свойств [18]:

- автономность – каждый агент независим и имеет своё сознание, свою модель поведения
- локальность – в каждый момент времени агент не имеет всей информации об окружающем его мире, только лишь о его части
- целеустремленность – агент не просто существует, а хочет достичь некоторой цели, в том числе прибегая к помощи остальных агентов системы
- идентичность – не обязательная, но часто встречающаяся черта агентов внутри одной МАС, говорящая о том, что каждый агент в сети полностью идентичен остальным; кроме того, в системе нет главного агента, они все равны между собой, что позволяет сохранять децентрализованность системы

1.2. Мультиагентный фреймворк JADE

JADE является платформой для разработки мультиагентных систем [10], основанной на спецификациях FIPA (Foundation for Intelligent Physical Agents) [6]. Эти спецификации лежат в основе многих мультиагентных систем, причем, спецификация позволяет общаться агентам,

находящимся в разных платформах, при условии, что обе платформы реализуют эти спецификации. Несмотря на довольно долгое существование спецификаций FIPA и самого фреймворка JADE, последний до сих пор остается одним из самых популярных инструментов при реализации задач, связанных с мультиагентным взаимодействием [20], [3].

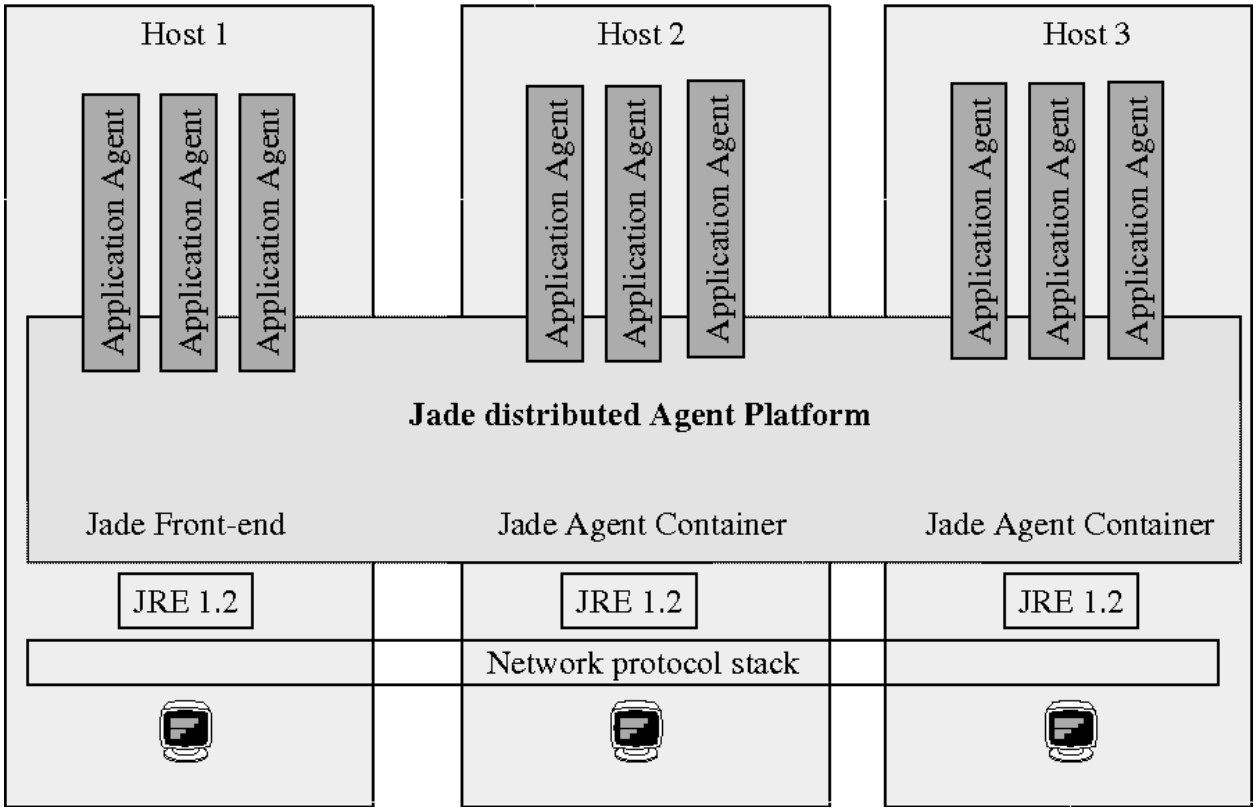

Рис. 1: Пример распределенного приложения на JADE

К основным элементам платформы относятся:

- агент – независимый процесс, который может принимать какие-либо решения и общаться с другими агентами
- поведение – логика каждого агента реализуется в рамках одного или нескольких поведений (Behaviour). В зависимости от класса-наследника, поведение может обладать теми или иными чертами, а логика, прописанная в рамках него, запускаться тем или иным образом. К примеру, TickerBehaviour запускается циклически через интервал T, в то время как CyclicBehaviour подразумевает постоянную работу

- контейнер – логическая совокупность агентов, поднятых на одном хосте. Основной контейнер на каждом хосте именуется, как Main Container

- сервис «желтые страницы» – процесс, хранящий информацию о сетевом адресе каждого агента; принимает участие в процессе передачи сообщений между агентами

- AMS (agent management service) – сервис, отвечающий за регистрацию каждого агента в мультиагентной системе; если данный сервис не зарегистрирует какого-либо агента, можно считать, что этого агента вовсе не существует

- MTS (message transport service) – сервис, осуществляющий передачу сообщений между агентами. Формат сообщений определён FIPA и известен как ACL (Agent Communication Language)

Из явных плюсов использования JADE можно отметить:

- асинхронность – каждый агент выполняется независимо, в отдельном потоке

- потоки в JADE достаточно легковесны и нет особых проблем в том, чтобы создать несколько десятков или даже сотен агентов

- позволяет строить распределенные приложения, состоящие из набора узлов, на каждом из которых может быть расположен один или несколько контейнеров с агентами

Из недостатков, применительно к рассматриваемой задаче, можно отметить следующее:

- отсутствие графического интерфейса для взаимодействия со средой и агентами; также отсутствует какое-либо физическое воплощение агентов, по сути агент в JADE – просто процесс, и не более

- отсутствие каких-либо встроенных пакетов для симулирования физических процессов между агентами и с виртуальной средой

1.3. Симуляторы роботов

Симуляторы роботов являются отличным решением для проверки тех или иных алгоритмов в виртуальной симуляции перед тем, как делать это на реальных роботах. Это экономит массу времени, так как отсутствует необходимость собирать и настраивать роботов, подготовливать тестовое окружение. Виртульная симуляция исключает возможность повреждения оборудования при проверке того или иного алгоритма. Кроме того, современные симуляторы способны реалистично моделировать физику реальных процессов, что позволяет в полной мере полагаться на результаты тестов, проведенных над роботами в симуляции.

В рамках следующего обзора актуальных симуляторов роботов будут затронуты лишь те решения, которые являются 3D-симуляторами с возможностью создания нескольких роботов в рамках одной симуляции. Кроме того, не будут затронуты симуляторы, целью создания которых было предоставление инструмента для проверки AI алгоритмов и алгоритмов компьютерного зрения. К последним относятся такие инструменты, как iGibson, AI2-THOR, AirSim и т.д.

1.3.1. CoppeliaSim (ex. V-Rep)

Симулятор роботов CoppeliaSim [14] основан на распределенной архитектуре: каждый объект или модель может самостоятельно контролироваться внутренним скриптом, плагином, ROS узлом или любым другим программным решением. Симулятор поддерживает множество физических движков, такие как Bullet, ODE, Vortex и Newton. Логика контроллеров роботов может быть написана на большоим количестве языков: C/C++, Python, Java, Lua, Matlab или Octave.

Несмотря на хорошую визуализацию и широкие возможности в сфере построения роботов и их окружения, отмечается, что данный симулятор является одним из наиболее требовательных к системным ресурсам среди аналогичных решений [7].

1.3.2. Gazebo

Симулятор Gazebo [8] является одним из самых популярных и известных симуляторов роботов. По-умолчанию в нём используется физический движок ODE, однако из исходного кода можно собрать версию с Simbody, Bullet или DART. Этот проект имеет хорошую документацию и множество примером, с помощью которых можно научиться работать с ним.

В работе [19] отмечается чрезмерное использование CPU. В статье же [7] на ряде опытов показано, что данный симулятор затрачивает много ресурсов в режиме работы с GUI, без него же уровень нагрузки на CPU снижается в разы. Кроме того, отмечается, что Gazebo не поддерживает возможности изменения импортированных 3D моделей через редактор симулятора, что, несомненно, является большим минусом.

1.3.3. Webots

Webots представляет из себя графическое мульти-платформенное приложение с открытым исходным кодом, которое позволяет моделировать виртуальную среду, располагать в ней статические и динамические объекты, в том числе роботов [17]. В нём можно заниматься проектированием и конструированием роботов из набора готовых элементов, прописывать им различными внешние и физические характеристики, а также программировать логику их работы. Логика робота размещается в контроллере и представляет собой ничто иное, как программу, написанную на одном из поддерживаемых Webots языках: C/C++, Java, Python или MATLAB.

Среда, в которой существуют и с которой взаимодействуют роботы, называется миром. Мир представляет из себя все объекты, из которых состоит среда конкретной симуляции: статические объекты, вроде пола, стен и препятствий, и ряд нефизических параметров среды, вроде задаваемой локальной системы координат или размера шага симуляции. Каждый физический объект можно довольно гибко настроить под

свои нужны, так как у каждого объекта есть своя своя физика, размеры, внешний вид и т.д.

Частью мира также являются и роботы. Webots предоставляет возможность взять как заранее готового робота из стандартной библиотеки, поставляемой вместе с симулятором, так и создать своего собственного, с нуля, из представленных объектов и устройств. Кроме тела, у робота есть различные передвижные части, такие как шарниры и колёса. Также роботв может иметь различные датчики, позволяющие получать информацию из виртуального мира.

Кроме того, одним из важнейших качеств Webots является наличие мощного физического движка (модификация движка ODE), способного симулировать процессы, аналогичные тем, что происходят в реальном мире. Отдельно стоит отметить прекрасную документацию данного симулятора и наличие подробного руководства, с помощью которого можно быстро обучиться всем нюансам работы с Webots.

Этот симулятор является одним из наиболее популярных инструментов для симуляции простых роботов и проверке на них тех или иных алгоритмов [4], [9]. Он имеет стабильные обновления версий, которые выпускаются два раза в год. Для построения роботов в данном симуляторе существует большое количество встроенных в систему блоков и устройств, кроме того, данный симулятор позволяет импортировать CAD модели.

Особенностью работы контроллеров данного симулятора является их синхронность. То есть, каждый следующий контроллер отрабатывает только после того, как завершится логика работы предыдущего.

1.3.4. MuJoCo

В отличие от других рассмотренных инструментов, MuJoCo представляет из себя физический движок [11], сильная сторона которого – многосуставная динамика с контактом. Данный проект был закрытым коммерческим решением до октября 2021 года, сейчас же он перешёл в open-source. Помимо самого движка, MuJoCo предоставляет интерактивное окно визуализации, в котором можно взаимодействовать с

находящимися в симуляции объектами.

Модели окружения и роботов представлены в MuJoCo в виде XML файлов в форматах MJCF и URDF. В данном симуляторе отсутствует поддержка визуального создания и редактирования роботов, что, несомненно, является большим недостатком. Кроме того, было отмечено отсутствие простого руководства по плавному вхождению в данный симулятор.

1.3.5. ARGoS

ARGoS является довольно интересным решением в сфере симуляторов роботов [1]. Как утверждает автор этого проекта, основной задачей при его создании ставилась высокая производительность работы системы, в том числе при большом числе роботов. По вышеописанным критериям, можно сказать, что ARGoS является хорошим решением для симуляции мультиагентных систем. Симулятор поддерживает написание логики контроллеров роботов на двух языках: C++ и Lua.

Однако, проект не лишён минусов, и некоторые из них довольно значительны. В качестве физического движка в симуляторе представлено своё самописное решение, которое проигрывает в возможностях таким популярным движкам, как ODE или Bullet [7]. Кроме того, явными минусами ARGoS является отсутствие графического редактора окружения и возможности импортировать готовые модели из программных комплексов для моделирования, например, CAD – вместо этого, при желании создать свою модель объекта или робота, необходимо использовать OpenGL.

Также, было отмечено, что документация по симулятору находится в довольно сыром состоянии – многие темы, заявленные для добавления, помечены как «TODO» (to be done). А при обзоре репозитория проекта на github сложилось впечатление, что главный разработчик по той или иной причине приостановил разработку проекта, так как крупных изменений не было уже довольно долгое время.

1.3.6. Итог

По результатам обзора можно сделать вывод, что на текущий момент существует множество зрелых решений, позволяющих создавать симуляции роботов. Каждый из них имеет свои особенности, хотя многие из них присутствуют сразу у ряда решений.

Из рассмотренных выше симуляторов роботов можно отметить, что симулятор Webots на данный момент всё ещё остается достаточно удобным инструментом для построения простых симуляций роботов. Симулятор имеет богатую стандартную библиотеку моделей, возможность с легкостью добавлять свои, используя встроенный GUI-редактор. И хотя возможности физического движка Webots ограничены, по сравнению с аналогами, при работе симуляции он затрачивает существенно меньше системных ресурсов, по сравнению с аналогами [19].

Таким образом, симулятор Webots является хорошим решением при создании виртуальных симуляций роботов в случаях, когда необходимо легко и быстро создать виртуальное окружение с большим количеством простых роботов, и при этом задача не зависит от качества визуализации и интерактивности окружающей среды.

Основным минусом этого симулятора является синхронность работы логики контроллеров роботов, что очень критично при симулировании большого количества роботов, являющегося частью одной мультиагетной системы.

Во-первых, в реальном мире роботы работают одновременно друг с другом, что может приводить к тому, что на очередной итерации того или иного алгоритма у робота ещё не будет части данных, которые ему должны были передать его соседи. В условиях полностью синхронных процессов, уже со второго шага симуляции у каждого робота будет полная информация обо всех его соседях, полученная на предыдущем шаге. Таким образом, Webots, с идеологической точки зрения не подходит для моделирования мультиагентных систем.

Во-вторых, достаточно большое количество роботов в тестовой среде Webots будет приводить к долгой работе всей симуляции, так как

количество необходимых подсчетов увеличивается, а вычислительная мощность остается та же. Этот факт приводит к тому, что реальное затраченное на симуляцию время и виртуальное отличаются на порядок.

Следовательно, в случае проверки того или иного мультиагентного алгоритма в симуляторе Webots на достаточно большом рое роботов, мы будем получать очень медленную скорость работы и нереалистичную модель поведения. Для решения данной проблемы необходимо предложить некоторое гибридное решение, в котором основная логика контроллеров будет вынесена во внешнюю систему, с возможностью размещения на отдельном вычислительном узле, для облегчения самой симуляции в Webots.

2. Гибридный симулятор

2.1. Постановка требований

Предложенное решение должно удовлетворять следующему ряду требований:

- Визуализация – у пользователя гибридного симулятора должна быть возможность наблюдать за процессом симуляции
- Асинхронность процессов – как уже было упомянуто, в реальном мире роботы не ждут друг друга перед тем, как принять то или иное решение, все они думают и двигаются одновременно друг с другом, и того же хочется достичь в финальном решении
- Распределенная архитектура – хочется, чтобы построенный прототип мог легко масштабироваться на несколько вычислительных узлов, тем самым повышая вычислительную мощность конкретной инсталляции и позволяя решать мультиагетные задачи с достаточно большим числом участников роя
- Возможность и простота интеграции с реальными устройствами – система должна уметь работать как с виртуальными, так и с реальными роботами

Рассмотренный симулятор Webots удовлетворяет требования по визуализации и низкому порогу вхождения. Действительно, строить симуляции в этой программе достаточно просто через предоставленный редактор. Кроме того, он имеет бесплатную версию, которая содержит в себе всё необходимое для проектирования роботов и окружающей их среды.

В качестве сервиса для мультиагентного общения роботов был выбран фреймворк JADE. Он отлично подходит для создания легковесных агентов-процессов, каждый из которых работает независимо, и в рамках которого происходит асинхронное выполнение логики этого

агента. Кроме того, JADE имеет возможность легко произвести горизонтальное масштабирование, так что не составит труда запустить контейнеры JADE на множестве связанных друг с другом по сети машин.

Выбранные сервисы покрывают большинство поставленных требований. Однако, всё ещё хочется, чтобы спроектированная система имела простую возможность интеграции с реальными устройствами. Для этого алгоритм, запущенный в кооперативе JADE и Webots, должен быть легко переносим без существенных изменений в организации общения и архитектуре.

2.2. Интеграционное ПО

Для интеграции описанных выше сервисов в единую систему было решено использовать платформу для разработки роботов Robot Operating System (ROS) [12], [13]. Данная платформа является продуктом с открытым исходным кодом и активно поддерживается, имеет стабильные обновления. Для создания программ под ROS поддерживаются два языка программирования: Python и C++.

ROS дает возможность создавать распределенную архитектуру системы, что дает возможность горизонтального масштабирования системы при необходимости [16]. Кроме того, в случае, когда придет время переносить тестируемый алгоритм на реальные устройства, данный переход можно будет осуществить поэтапно, более плавно. Промежуточно, перед полным переносом, можно будет подключить роботов к построенной системе, вместо симулятора Webots, и перенести лишь логику, отвечающую за передвижение устройств. Также не исключается возможность одновременного подключения как виртуальных, так и реальных роботов к одной системе, что может быть интересным при решении определенного ряда задач.

Кроме того, стоит отметить, что многие популярные решения, используемые при построении своих реальных роботов, такие как arduino, имеют нативную поддержку ROS. Что не мене важно, Webots также имеет нативную реализацию, позволяющую общаться с ROS.

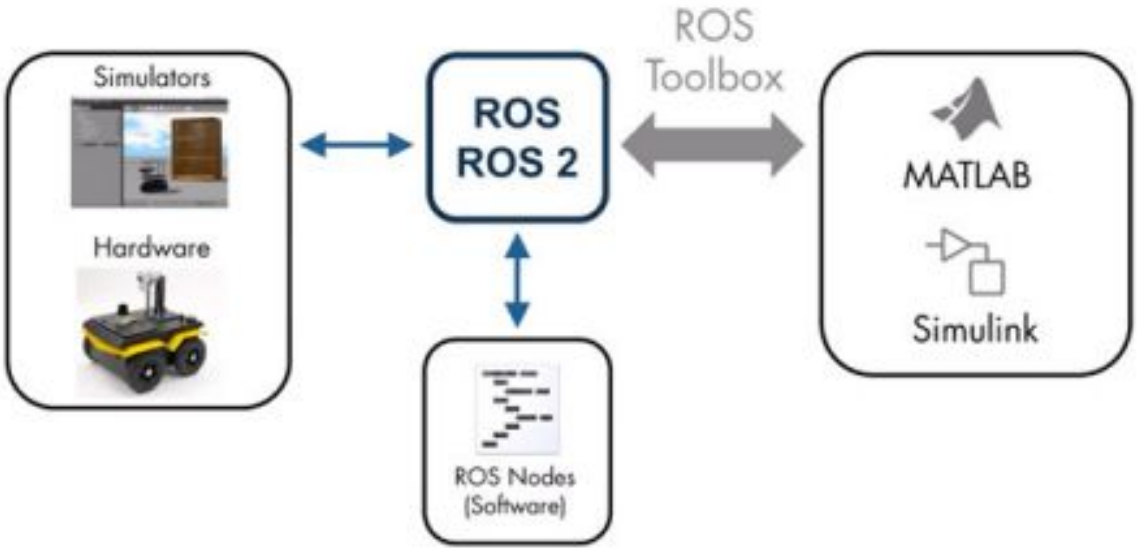

Рис. 2: Платформа ROS

Существует две версии данной платформы: ROS и ROS2. Вторая версия, появившаяся в 2015 году, хоть и добавляет ряд новшеств, к примеру, поддержку для Windows ОС, является менее популярной у сообщества, в силу меньшего доступного числа примеров систем, поэтому далее будет рассмотрена лишь первая версия платформы. Обе версии системы представлены на операционных системах Ubuntu и Mac OS.

Основными элементами в экосистеме ROS являются узлы (nodes). Узел является некоторой программой, включенной в экосистему ROS, и имеющий возможность коммуницировать с другими узлами.

Для коммуникации внутри системы ROS предлагает три различных подхода:

- топики (topics) – коммуникация основана на паттерне publisher/subscriber. Внутри ROS регистрируются топики, на которые могут подписываться subscriber'ы, и в которые могут посылать данные publisher'ы. Чаще всего данный вид коммуникации используется для передачи потоковой информации

- сервисы (services) – коммуникация основана на принципе «remote

procedure call» (RPC), то есть, сервис, зарегистрированный в ROS, может быть вызван одним из участников коммуникации, а сервис, в свою очередь, вызывает ту или иную функцию на вызываемой стороне. Сервис должен предоставляться отдельно для вызова каждого конкретным узла, а сами вызовы должны быть использованы в целях запроса каких-либо данных, так как они являются блокирующими

- действия (actions) – представляет из себя гибрид двух предыдущих подходов. Action позволяет совершать запросы, как это делают сервисы, а результат запроса получать, как поток данных, получаемый из соответствующего топика. Данный поток является прерываемым по соответствующему вызову для того же action'а

Архитектурно ROS придерживается распределенного подхода, то есть, узлы построенной на ROS системы могут работать на разных машинах без каких-либо проблем. Связующим же звеном в данной системе выступает roscore – сервис, который позволяет выстраивать коммуникацию между всеми существующими в системе узлами. Одними из важнейших процессов, входящих в roscore, являются:

- Master – регистрирует узлы, входящие в состав системы. Кроме того, отслеживает существующие в данный момент времени сервисы и publisher'ов и subscriber'ов топиков. Основная задача Master процесса - осуществление peer-to-peer коммуникации между двумя узлами

- Parameter Server – используя данный процесс, узлы могут предоставлять свои конфигурациооные параметры посредством API и получать параметры от других узлов

Кроме того, в случае отсутствия нативной поддержки ROS со стороны той или иной платформы, всё же можно получить доступ к поднятой экосистеме. Для этого используется сервис rosbridge, общение с которым происходит по специальному протоколу, работающему через JSON API. Сам же rosbridge представляет из себя WebSocket сервер.

Дополнительно, ROS имеет ряд встроенных вспомогательных утилит, помогающих при работе с системой. Так, например, есть утилита rqt_graph, которая отображает график связей между узлами экосистемы. Также есть ряд утилит, отвечающих за измерение скорости поступления данных в топики, что может быть полезным при исследовании эффективности построенного решения на базе ROS.

2.3. Верхнеуровневая архитектура

Верхнеуровневая архитектура предлагаемого решения выглядит следующим образом (Рис. 3). Есть некоторый вычислительный кластер, в котором запущены процессы, каждый из которых представляет реального или виртуального робота. Эти процессы могут общаться между собой для организации мультиагентного поведения.

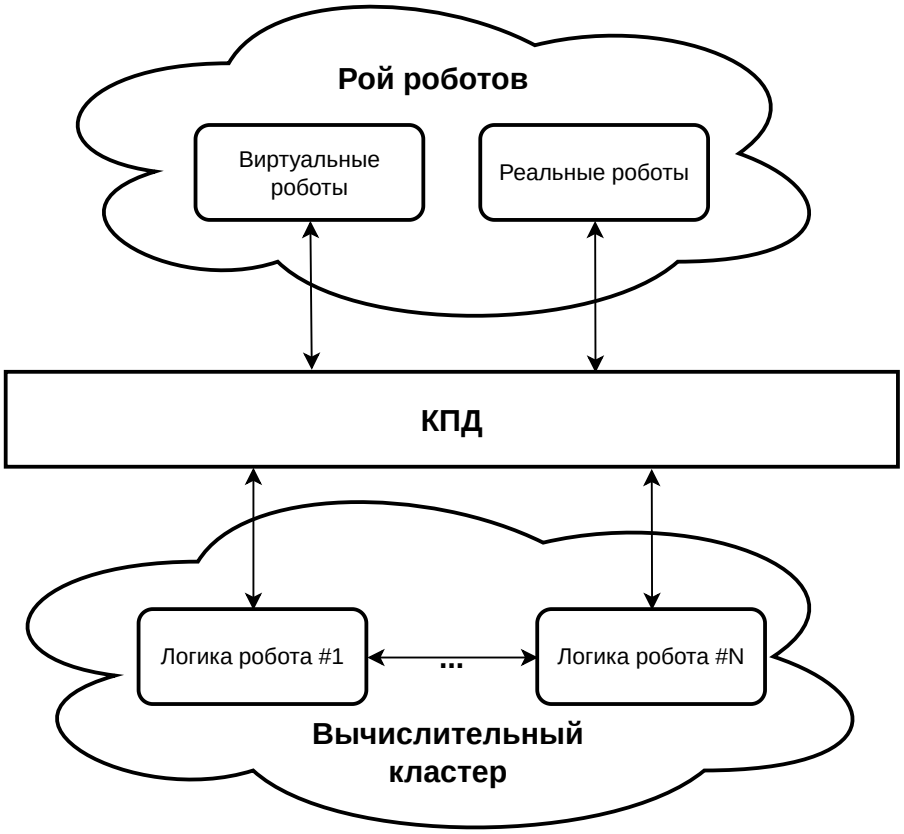

Рис. 3: Верхнеуровневая архитектура гибридного симулятора

В то же время, отдельно от вышеописанного кластера запущена

некоторая система, симулирующая поведение реальных роботов, либо же имеются реальные роботы. Роботы содержат минимум логики, а та, что имеется, отвечает лишь за передачу текущих параметров соответствующему вычислительному процессу кластера, и за передвижения, в зависимости от получаемой от кластера информации.

Общение же осуществляется по каналу передачи данных (КПД). К данному КПД могут подключаться как роботы, так и процессы вычислительного кластера. Ограничения на протокол общения не задается, основное требование к КПД состоит лишь в возможности совершать общение по сети.

2.4. Детальная архитектура

Детальный вид архитектуры гибридного симулятора с конкретными программными компонентами представлен ниже (Рис. 4). Система состоит из одного симулятора Webots с запущенной симуляцией роботов, кластера из K JADE контейнеров (один из которых является главным контейнером, который выступает центральным звеном при организации общения между агентами платформы JADE) и узлов ROS. Помимо этого, в данной системе необходимо использовать сервис rosbridge, так как в JADE отсутствует нативная поддержка ROS, и необходимо использовать промежуточное звено для того, чтобы агенты JADE могли осуществлять коммуникацию с роботами Webots.

Каждый робот в Webots и агент в JADE имеет свой уникальный id, причем, для корректной работы системы требуется, чтобы на каждого робота Webots существовал свой агент в кластере JADE. В данном случае не важно, в каком контейнере расположен тот или иной агент, важно лишь, чтобы у него и соответствующего ему робота был одинаковый id.

Как уже упоминалось ранее, основная часть логики робота, отвечающая за мышление и принятие решения о следующем шаге, вынесена в JADE. В роботе же оставлено минимум функциональности, отвечающей только лишь за то, чтобы получить данные с датчиков и передать

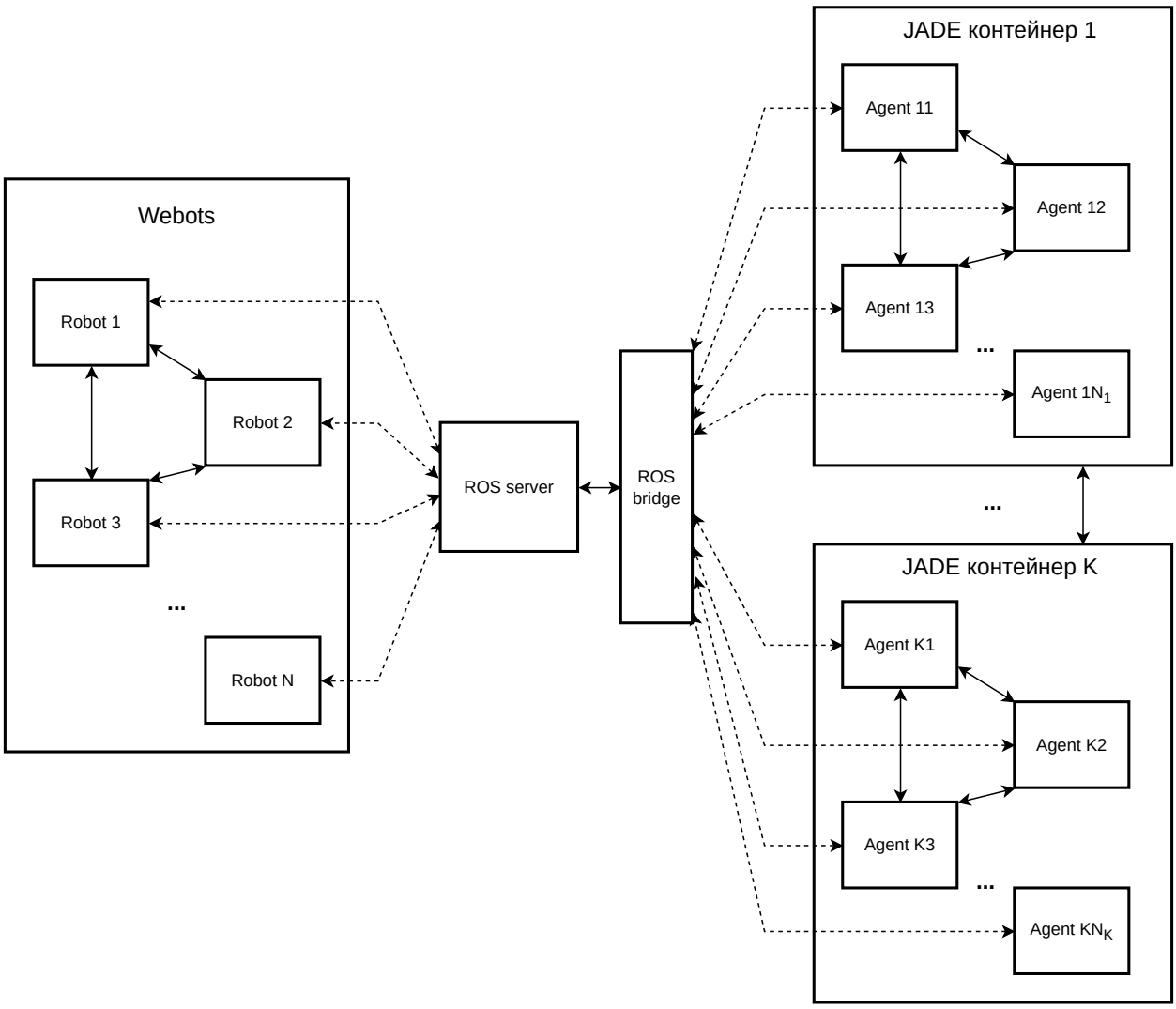

Рис. 4: Детальная архитектура гибридного симулятора

их соответствующим агентам. Кроме того, в роботе находится вся логика, отвечающая за перемещение. Это сделано для того, чтобы максимально облегчить и ускорить логику контроллера роботов Webots. Таким образом, увеличение числа роботов в симуляции будет меньше сказываться на скорости работы системы.

Для общения был выбран подход publisher/subscriber. При таком подходе, на каждом шаге симуляции роботы будут слать актуальные параметры агентам и получать новый результат, используя топики. Для данной системы были выбраны следующие наименования топиков:

- /webots_to_jade/<robot_id>/robot_info - информация для мультиагентной платформы, содержащая параметры конкретного ро-

бота

- /jade_to_webots/<robot_id>/result - результат работы мультиагентного алгоритма

2.5. Алгоритм работы системы

Алгоритм работы данного гибридного симулятора можно представить в следующих шагах:

1. Запускаем сервис roscore
2. Запускаем сервис rosbridge
3. Запускаем main контейнер JADE, указываем URL rosbridge
4. Запускаем остальные контейнеры JADE платформы, указываем URL main контейнера и rosbridge
5. Агенты внутри запущенных JADE контейнеров будут выполнять следующее:
    5.1. поведение, отвечающее за принятие решения и унаследованное от TickerBehaviour: считывает параметры робота из топика, отправляет необходимую для конкретного алгоритма информацию агентам-соседям, рассчитывает новый результат алгоритма и отсылает его в топик
    5.2. поведение, принимающее сообщения от соседей и унаследованное от CyclicBehaviour: принимает сообщения ACLMessage с информацией от агента-соседа и сохраняет её для последующего использования в мультиагентном алгоритме
6. Запускаем симулятор Webots, указываем URL ros master
7. При запуске Webots роботы будут выполнять следующее:
    7.1. с периодичностью в T сек. поочередно запускается код контроллеров каждого робота

7.2. в рамках контроллера происходит считывание параметров с датчиков робота и их передача в соответствующий топик

7.3. далее происходит считывание из другого топика с целью получить результат работы мультиагентного алгоритма по данным, отправленным на предыдущем шаге

3. Экспериментальные результаты

3.1. Тестовая задача

В качестве мультиагентной задачи, в ходе которой будет проверена эффективность построенного гибридного симулятора, была выбрана задача о групповом движении роя роботов на источник света в среде с препятствиями (Рис. 5). Роботы должны, кооперируясь, двигаться на луч света, при этом огибая препятствия, стоящие у них на пути.

У тестового робота есть колеса, чтобы была возможность перемещаться в пространстве. Размер каждого робота примерно равен 0.5 метров в длину и 0.3 метра в ширину. Кроме того, робот в виртуальной среде имеет следующий набор устройств:

- DistanceSensor – позволяет измерять расстояние до объектов в направлении, заданном датчику
- LightSensor – позволяет измерять уровень светового излучения в направлении, заданном датчику
- Compass – позволяет определять текущее направление движения робота
- Emitter – позволяет имитировать радиоволновый излучатель для передачи сообщений между роботами внутри симуляции. Отправка сообщений осуществляется по заранее заданной частоте. Как альтернатива, может отправлять сообщения по всем частотам
- Receiver – позволяет принимать сообщения от других роботов, посылаемых с помощью устройства Emitter. Принимает сообщения на определенной, заранее заданной частоте. Как альтернатива, может принимать сообщения со всех частот сразу

Тестовая среда состоит из прямоугольного поля размером 15 x 8 метров, окруженного стенами. По центру поля расположены два цилиндрических препятствия, которые роботы должны будут обходить

Рис. 5: Групповое движение роя в направлении источника света с обходом препятствий [5]

при движении. С трех сторон за границей данного поля расположены излучатели света, причем, излучатель, расположенный на севере, светит в два раза сильнее, чем излучатели на востоке и западе. Таким образом передний излучатель является конечной целью для роя роботов, в то время как боковые источники света играют роль отраженного от пола, стен и препятствий света. Такая манипуляция со светом нужна затем, что в Webots на момент написания работы не существует функциональности по отражению света, но для симуляции, приближенной к реальным, мы хотим, чтобы часть света от источника была доступна световым датчикам даже в условиях, когда между роботами и источником света находится препятствие. На юге, друг за другом, в шеренгах по 25 устройств расположен рой роботов.

Для решения задачи используется модификация алгоритма протокола локального голосования. Согласно предложенному в [5] решению, роботы обмениваются друг с другом информацией о текущем курсе движения и об уверенности в этом курсе. После этого, высчитывает-

ся новое направление движения, принимая во внимание полученную от ближайших соседей информацию и показания с собственных датчиков.

3.2. Тестовое оборудование

Тестирование построенной системы осуществлялось на различных вариациях инфраструктур и на различных машинах. В тестировании принимали участие следующие машины:

- Laptop #1
  - процессор – AMD Ryzen 5 4600U (6 x 2.1GHz / 12 потоков)
  - видеокарта – интегрированная AMD Radeon Graphics
  - оперативная память – 16 ГБ (2 выделено под GPU)
  - хранение данных – SSD «SSSTC CL1-4D512»
- Laptop #2
  - процессор – AMD Ryzen 7 5800H (8 x 3.2GHz / 16 потоков)
  - видеокарта – Nvidia GeForce GTX 1650
  - оперативная память – 16 ГБ
  - хранение данных – SSD «SAMSUNG MZVLB1T0HBLR-000L2»

3.3. Описание проводимых опытов

В ходе экспериментов последовательно были созданы миры Webots с 25, 50 и 100 роботами (Рис. 6, 7). Размер группы является очень важным показателем для конкретного тестового случая, так как с ростом группы увеличивается вычислительная нагрузка системы. Кроме того, тот или иной мультиагентный алгоритм может с разной степенью успешности работать в зависимости от размера роя.

В каждом случае проводился ряд опытов, не менее 10, в рамках которых замерялось, за какое время роботы пересекут финишную черту в сторону источника света. В качестве такой черты была выбрана одна

из линий, расположенная за препятствиями, то есть, чтобы её пересечь, роботам нужно было миновать стоящие перед ними препятствия, что означало бы, что они корректно справились со своей задачей. Кроме того, каждая среда тестировалась на двух архитектурах с целью дальнейшего сравнения оригинального подхода и предложенного гибридного.

В симуляторе Webots шаг симуляции был установлен в 100ms. Данный шаг был подобран эмпирически - было выявлено, что при таком шаге роботы всё ещё обладают достаточной реакцией для реагирования на изменяющийся мир, при этом выбор этого шага, вместо установленных по-умолчанию 32ms., позволит снизить нагрузку на всю систему.

Кроме того, важной чертой является расположение симулятора Webots. В каждом сценарии он должен находиться на одной и той же машине, чтобы дальнейшее сравнение проведенных опытов с классическим подходом было допустимым.

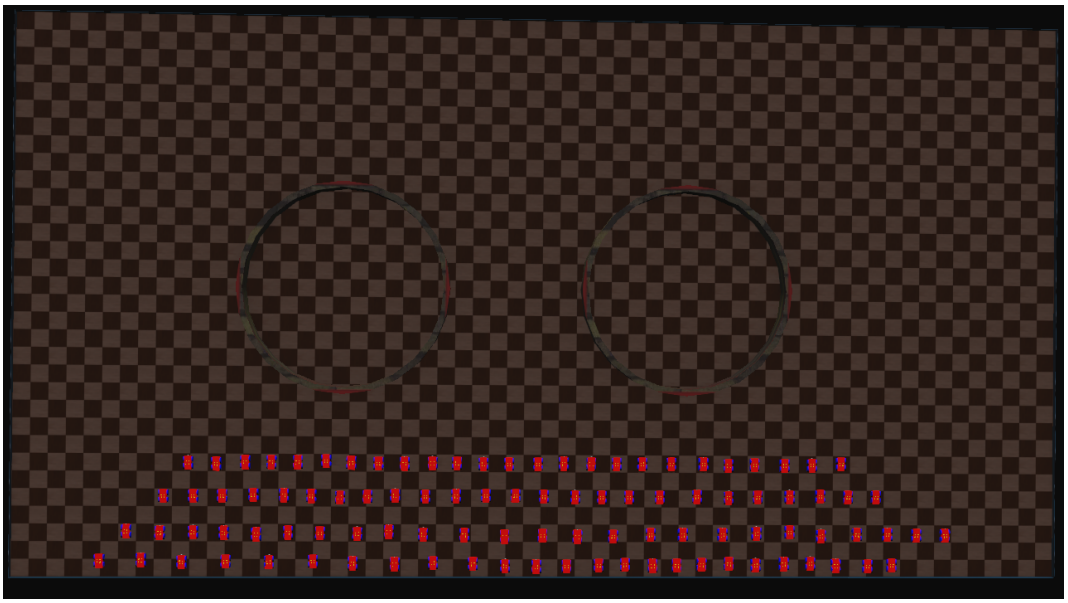

Рис. 6: Симуляция с роем в 100 роботов на старте

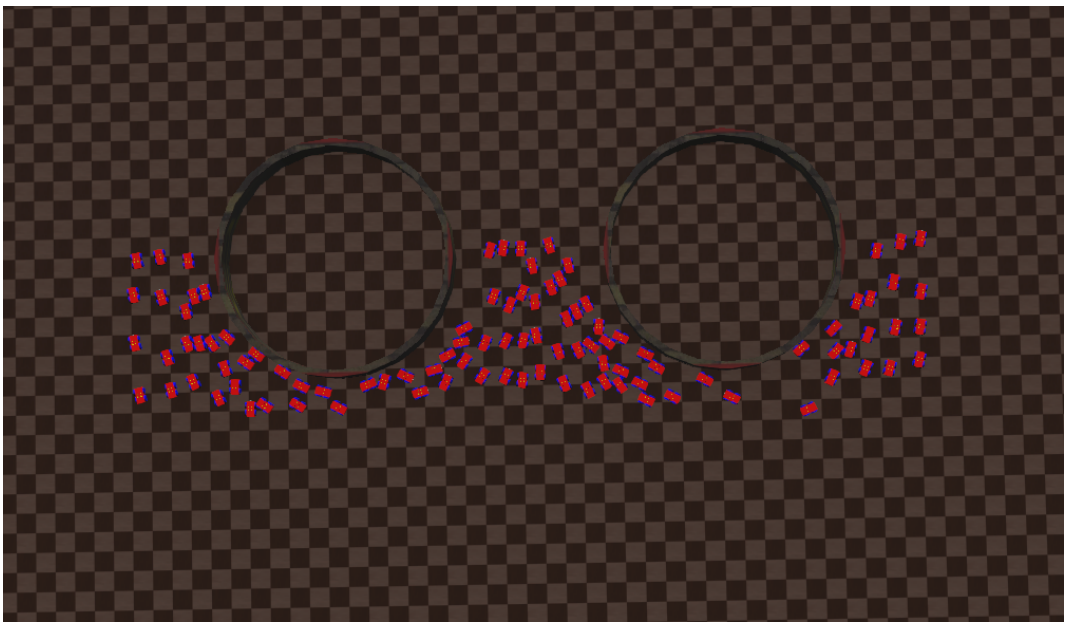

Рис. 7: Движения роя из 100 роботов в симуляции

3.3.1. Описание метрик

Для оценки полученного решения были проведены замеры следующих величин:

1. виртуальное время симуляции, за которое решается мультиагентная задача

2. реальное время, за которое решается мультиагетная задача

3. частота посылки роботам очередного решения мультиагентного алгоритма, идеальное значение которого равно шагу симуляции; стандартное отклонение значений данного параметра

4. частота посылки агентам текущих параметров робота; стандартное отклонение значений данного параметра

Таким образом, распределение компонентов системы внутри конкретной инфраструктуры должно происходить таким образом, чтобы минимизировать время работы алгоритма, при этом добиваясь как можно более низкой задержки в общении мультиагентной системы с симулятором. Под задержкой (delay) понимаем разницу между идеальной скоростью предоставления решения мультиагентной задачи, равному

шагу симуляции time step, и действительной частотой отправки сообщений (publish freq) внутри системы:

$$delay = publish\_freq - time\_step$$

Аналогичная задержка может быть высчитана и для данных, посылаемых симулятором в мультиагентную систему, однако, из-за особенностей выполнения логики контроллеров в Webots, не является возможным повлиять на неё.

3.3.2. Сценарий #0

Перед тем, как переходить к тестированию построенного решения, необходимо замерять эффективноть работы обычного подхода, когда симуляция просто запущена в симуляторе Webots. Опыты проводились на машине Laptop #1.

3.3.3. Сценарий #1

В рамках первого сценария все компоненты предложенного решения были запущены на одной машине, Laptop #1.

При запусках симуляции в данном сценарии наблюдались периодические проявления так называемого thread starvation, так как часть агентов, несмотря на регулярное получение данных о роботах, не отправляли никакого результата в Webots. Такие проявления начались в момент, когда началось тестирование алгоритма в рое из 50 роботов.

Было отмечено, что проведении опытов со 100 роботами в рамках такого сценария сильно проседала производительность. Из-за дополнительных расходов на коммуницирование внутри экосистемы ROS, алгоритм работал даже дольше, чем оригинальный алгоритм в симуляторе Webots. Полученный результат окажется полезным, как доказательство преимущества использования распределенной архитектуры в последующих сценариях.

3.3.4. Сценарий #2

В сценарии #2 были использованы две машины. На Laptop #1 были подняты: Webots, ros master, rosbridge и JADE main контейнер. На другой же машине, Laptop #2, был поднят второй контейнер JADE. Агенты распределены между контейнерами таким образом, чтобы на Laptop #1 находилось 30% агентов, а на Laptop #2 - 70%.

Общение между машинами осуществлялось по локальной сети через роутер с использованием Wi-Fi. По сравнению с первым сценарием, нагрузка на Laptop #1 была меньшей, однако добавилась небольшая задержка коммуникации компонентов, связанная с передачей данных по сети.

3.3.5. Сценарий #3

В сценарии #3 были использованы две машины. На Laptop #1 был поднят симулятор Webots, ros master и rosbridge. На Laptop #2 же был поднят контейнер, содержащий всех агентов системы.

В это сценарии симулятор Webots может выкладываться на полную, так как основная часть системной нагрузки вынесена на другой вычислительный узел.

3.4. Результаты опытов

При проведении опытов были получены следующие результаты (Рис. 8), описывающие скорость работы системы в различных сценариях. Как можно увидеть из графиков, в случае установки всех компонентов решения на одной машине, скорость работы симуляции может быть даже хуже, чем исходная скорость работы в симуляторе Webots. Однако, если распределить агентов по разным машинам, или вовсе вынести их с машины, где запущен Webots, в результате получим лучшую скорость работы, чем мы можем получить, запуская обычную симуляцию.

Кроме того, были сделаны замеры скорости общения компонентов системы ROS (Рис. 9, 10). В идеальном сценарии, частота посылки со-

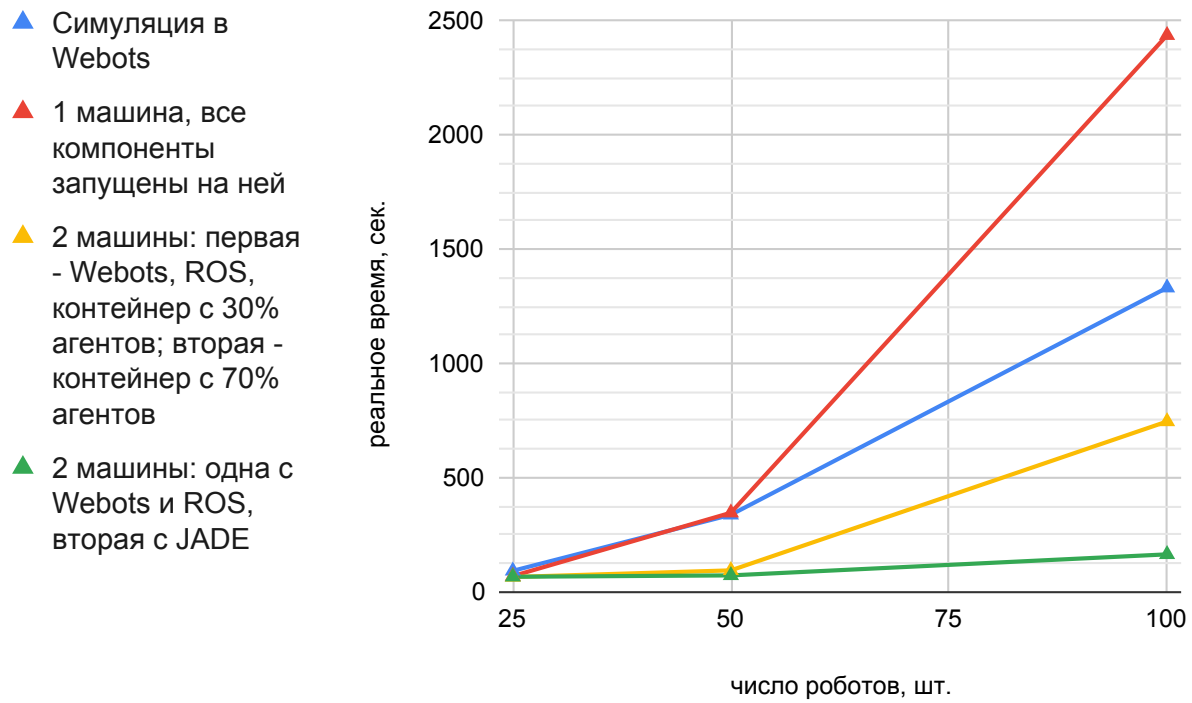

Рис. 8: Реальное время, затраченное на симуляцию, в зависимости от количества роботов в рое

общений в системе должна быть равной выбранному шагу симуляции, который также равен и шагу, с которым выполняется логика в агенте JADE. Выдвинутые в разделе 3.3.1 метрики важны для оценки эффективности системы, с той точки зрения, что при использовании распределенных вычислений точность нашего алгоритма абсолютно точно будет уменьшаться, так как в классической симуляции в Webots на момент подсчета очередного решения мультиагентного алгоритма роботы всегда обладают последними актуальными данными, чего может не происходить, если агенты и роботы JADE в распределенной сети будут общаться между собой с большой задержкой.

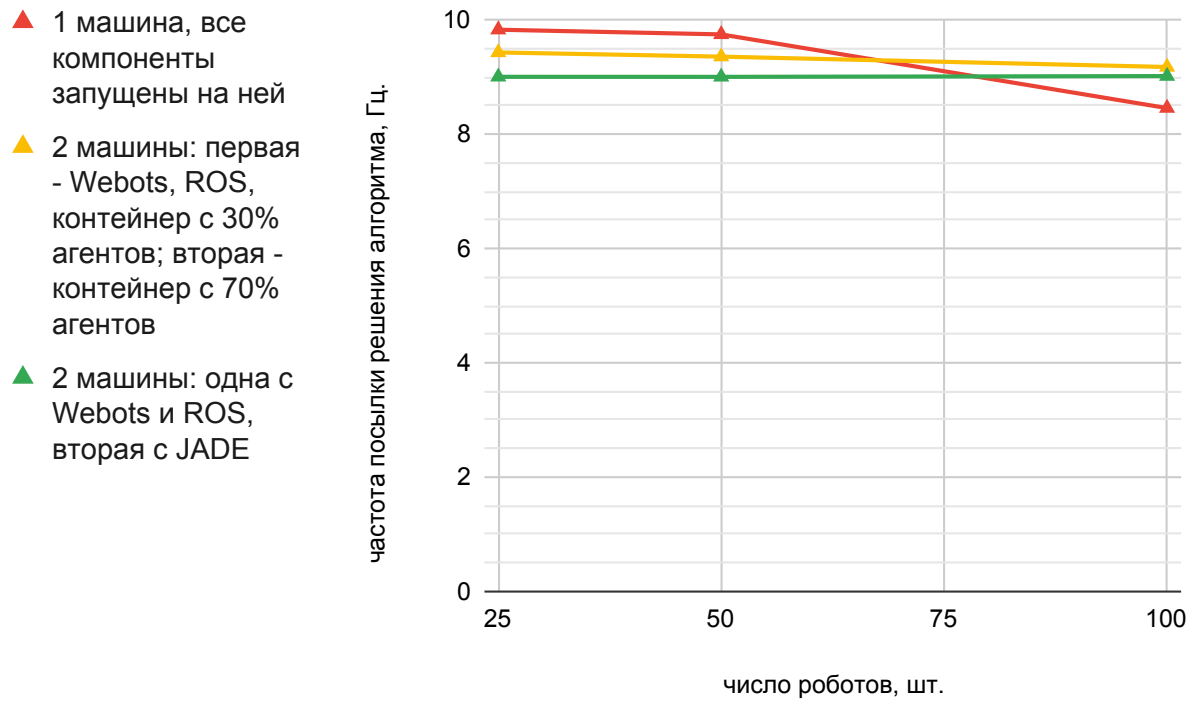

Рис. 9: Частота посылки результата работы алгоритма от агентов JADE, в зависимости от количества роботов в рое

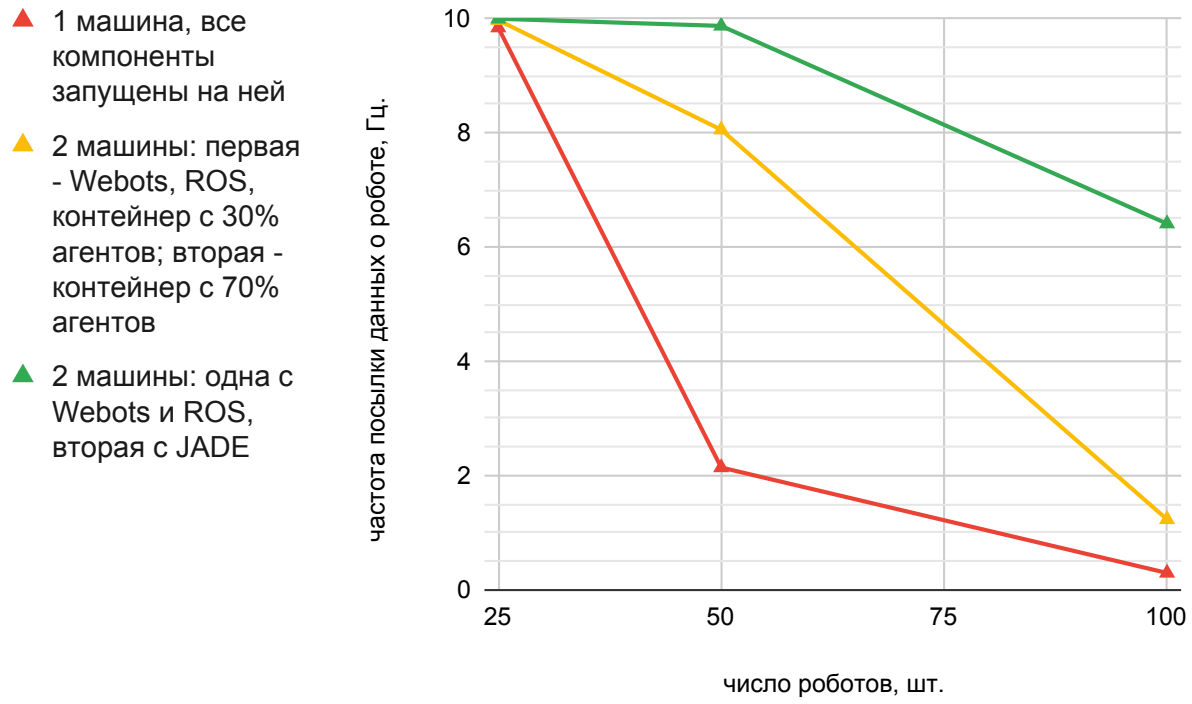

Рис. 10: Частота посылки параметров от роботов Webots, в зависимости от количества роботов в рое

Для рассмотренных сценариев использовалось значение шага равное 100ms., или, другими словами, в идеальном случае компоненты должны общаться друг с другом со скоростью 10 сообщений в секунду (10Hz.). Как видно из графиков, скорость посылки сообщений от агентов JADE, в случае использования распределенной инфраструктуры, слабо деградирует с ростом числа роботов в рое. Однако, того же нельзя сказать о частоте посылки сообщений роботами Webots - с ростом числа роботом в симуляции, она значительно замедляется, что приводит к снижению частоты общения в системе.

3.5. Итог

На основе представленных данных можем сделать вывод, что предложенная гибридная архитектура более оптимальна по времени для задач, в которые вовлечен большой рой роботов. Как видно по замерам скорости работы системы, в самом быстром сценарии реальное затраченное время на работу всей системы было снижено примерно в 8 раз. Причем, наибольшей эффективности и точности работы система достигает в случае, когда симулятор Webots запущен на отдельном компьютере и ничто не мешает ему работать. Вычислительные же узлы JADE лучше распределять таким образом, чтобы суммарное число задействованных для него потоков было наибольшим.

При этом, данная архитектура проигрывает обычной Webots симуляции в случае, если у нас в распоряжении находится лишь один слабый компьютер, либо когда количество роботов достаточно мало.

Заключение

В ходе работы были получены следующие результаты:

- Был проведён обзор предметной области
- Были определены требования для предлагаемого решения
- Был реализован прототип гибридного симулятора с визуализацией и физическими процессами в Webots и мультиагентным асинхронным взаимодействием в JADE. Компоненты коммуницируют между собой в экосистеме ROS
- Проведена апробация симулятора с использованием виртуальных роботов в симуляторе Webots. Мультиагентный алгоритм был протестирован на группах в 25, 50 и 100 роботов. Проведены тесты с использованием разного физического размещения узлов системы и разным количеством контейнеров JADE. Была показана эффективность гибридного симулятора при размещении контейнеров JADE на высокопроизводительных узлах в условиях увеличения размера роя роботов

Планы для дальнейшего развития:

- Провести апробацию построенной системы с использованием реальных роботов

Список литературы

[1] ARGoS. — https://www.argos-sim.info/. — Accessed: 22.05.2022.

[2] Amelina Natalia, Granichin Oleg, Kornivetc Aleksandra. Local voting protocol in decentralized load balancing problem with switched topology, noise, and delays // 52nd IEEE Conference on Decision and Control / IEEE. — 2013. — P. 4613–4618.

[3] Cardoso Rafael C, Ferrando Angelo. A review of agent-based programming for multi-agent systems // Computers. — 2021. — Vol. 10, no. 2. — P. 16.

[4] Körber Marian, Lange Johann, Rediske Stephan et al. Comparing Popular Simulation Environments in the Scope of Robotics and Reinforcement Learning. — 2021. — URL: https://arxiv.org/abs/2103.04616.

[5] Emergent intelligence via self-organization in a group of robotic devices / Konstantin Amelin, Oleg Granichin, Anna Sergeenko, Zeev V Volkovich // Mathematics. — 2021. — Vol. 9, no. 12. — P. 1314.

[6] FIPA. Agent management specification // Foundation for Intelligent Physical Agents. — 2003.

[7] Feature and performance comparison of the V-REP, Gazebo and ARGoS robot simulators / Lenka Pitonakova, Manuel Giuliani, Anthony Pipe, Alan Winfield // Annual Conference Towards Autonomous Robotic Systems / Springer. — 2018. — P. 357–368.

[8] Gazebo. — https://gazebosim.org. — Accessed: 22.05.2022.

[9] Ivaldi Serena, Padois Vincent, Nori Francesco. Tools for dynamics simulation of robots: a survey based on user feedback // arXiv preprint arXiv:1402.7050. — 2014.

[10] Java Agent DEvelopment framework. — https://jade.tilab.com/. — Accessed: 01.05.2022.

[11] MuJoCo - Advanced physics simulation. — https://mujoco.org/. — Accessed: 22.05.2022.

[12] ROS: an open-source Robot Operating System / Morgan Quigley, Ken Conley, Brian Gerkey et al. // ICRA workshop on open source software / Kobe, Japan. — Vol. 3. — 2009. — P. 5.

[13] Robot Operating System (ROS). — https://ros.org. — Accessed: 09.05.2022.

[14] Robot simulator CoppeliaSim. — https://www.coppeliarobotics.com/. — Accessed: 22.05.2022.

[15] Task allocation algorithm for the cooperating group of light autonomous unmanned aerial vehicles / Konstantin Amelin, Natalia Amelina, Oleg Granichin, Victor V Putov // IFAC Proceedings Volumes. — 2013. — Vol. 46, no. 30. — P. 152–155.

[16] Tsardoulias Emmanouil, Mitkas Pericles. Robotic frameworks, architectures and middleware comparison // arXiv preprint arXiv:1711.06842. — 2017.

[17] Webots: robot simulator. — https://cyberbotics.com/. — Accessed: 01.05.2022.

[18] Weyns Danny, Georgeff Michael. Self-adaptation using multiagent systems // IEEE software. — 2009. — Vol. 27, no. 1. — P. 86–91.

[19] A comparison of humanoid robot simulators: A quantitative approach / Angel Ayala, Francisco Cruz, Diego Campos et al. // 2020 Joint IEEE 10th International Conference on Development and Learning and Epigenetic Robotics (ICDL-EpiRob) / IEEE. — 2020. — P. 1–6.

[20] The first twenty years of agent-based software development with JADE / Federico Bergenti, Giovanni Caire, Stefania Monica, Agostino Poggi // Autonomous Agents and Multi-Agent Systems. — 2020. — Vol. 34, no. 2. — P. 1–19.

[21] Ревякина В.Я. Амелин К.С. Прототип системы децентрализованного группового управления на основе протокола локального голосования без маршрутизации данных // Материалы XXIII конференции молодых ученых "Навигация и управление движением". — 2021. — P. 116–118.

Приложения

Приложение 1: код программной реализации

Ссылка на github репозиторий с программной реализацией: https://github.com/Ielay/diploma_hybrid_simulation