МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«НОВОСИБИРСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
(НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ, НГУ)

Факультет информационных технологий
Кафедра параллельных вычислений

Направление подготовки 09.04.01 Информатика и вычислительная техника
Направленность (профиль): Технология разработки программных систем

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА МАГИСТРА

Закирова Владимира Константиновича

Тема работы:

РАЗРАБОТКА СРЕДСТВ ОБЕСПЕЧЕНИЯ ОТКАЗОУСТОЙЧИВОСТИ В СИСТЕМЕ LUNA НА ОСНОВЕ КОНТРОЛЬНЫХ ТОЧЕК

«К защите допущена»
Заведующий кафедрой,
д.т.н., профессор
Малышкин В. Э./……..
(ФИО) / (подпись)
«31» мая 2023г.

Руководитель ВКР
д.т.н., профессор
зав. каф. ПВ ФИТ НГУ
Малышкин В. Э./……...
(ФИО) / (подпись)
«20» мая 2023г.

Соруководитель ВКР
ст. преп. каф. ПВ ФИТ
Перепёлкин В. А/……..
(ФИО) / (подпись)
«20» мая 2023г.

Новосибирск, 2023

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«НОВОСИБИРСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
(НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ, НГУ)

Факультет информационных технологий

Кафедра параллельных вычислений
(название кафедры)

Направление подготовки: 09.04.01 ИНФОРМАТИКА И ВЫЧИСЛИТЕЛЬНАЯ ТЕХНИКА
Направленность (профиль): Технология разработки программных систем

УТВЕРЖДАЮ

Зав. кафедрой Малышкин В. Э.
(фамилия, И., О.)

…….
(подпись)

«16» января 2023г.

ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ МАГИСТРА

Студенту Закирову Владимиру Константиновичу, группы 21223
(фамилия, имя, отчество, номер группы)

Тема: Разработка средств обеспечения отказоустойчивости в системе LuNA на основе контрольных точек
(полное название темы выпускной квалификационной работы магистра)

утверждена распоряжением проректора по учебной работе от 29.10.21 № 0288,
скорректирована распоряжением проректора по учебной работе от 16.01.2023 № 0005
Срок сдачи студентом готовой работы 20 мая 2023 г.
Исходные данные (или цель работы) разработать механизм контрольных точек для системы LuNA. Реализовать предложенное решение.
Структурные части работы обзор родственных работ, механизм контрольных точек, реализация механизма контрольных точек, тестирование.

Руководитель ВКР
зав. каф. ПВ ФИТ,
д. т. н., профессор
Малышкин В. Э./……..
(ФИ О) / (подпись)

«16» января 2023г.

Задание принял к исполнению
Закиров В. К./……..
(ФИО студента) / (подпись)

«16» января 2023г.

Соруководитель ВКР
ст. преп. каф. ПВ ФИТ,
Перепелкин В. А/……..
(ФИ О) / (подпись)

«16» января 2023г.

СОДЕРЖАНИЕ

Введение **4**
1 Обзор родственных работ **8**
2 Механизм контрольных точек **11**
2.1 Общее представление о системе LuNA 11
2.2 Как работает исполнительная система в LuNA 12
2.3 Постановка задачи 16
2.4 Формат контрольной точки 17
2.5 Предлагаемая идея решения 20
2.6 Остановка работы исполнительной системы 22
2.6.1 Остановка подсистемы выполнения задач 23
2.6.2 Остановка коммуникационной подсистемы 24
2.7 Сохранение контрольной точки и возобновление работы 25
2.8 Итоги главы 25
3 Реализация механизма контрольных точек **26**
3.1 Архитектура LuNA в терминах классов 26
3.2 Остановка подсистемы выполнения задач 30
3.3 Остановка коммуникационной подсистемы 31
3.4 Сохранение контрольной точки и возобновление работы 32
3.5 Требования к системе и их поддержка 33
3.5.1 Избавление от хранения функциональных объектов 35
3.5.2 Избавление от хранения указателей на агенты на нескольких узлах 36
3.6 Тестирование 37
3.7 Итоги главы 38
Заключение **40**
Список использованных источников и литературы **42**
Приложение А **45**
Приложение Б **47**
Приложение В **55**

ВВЕДЕНИЕ

Актуальность темы исследования. Использование суперкомпьютеров для задач численного моделирования широко распространено, потребность в этом испытывают такие области как: вычислительная биология, медицина, вычислительная химия, вычислительная гидродинамика, криптография и.т.д. Исполнение задачи численного моделирования на суперкомпьютерах предоставляет очевидные преимущества по сравнению с исполнением на персональных компьютерах. Эти преимущества вытекают из большой вычислительной мощности суперкомпьютеров и возможности одновременно выполнять задачи большого количества пользователей. Но из-за сложности таких систем возникают и проблемы, узлы могут выходить из строя. Также в связи с множеством пользователей, использующих суперкомпьютер, появляется необходимость в целесообразном выделении ресурсов на определенные задачи. Нет надобности предоставлять всю вычислительную мощь одному человеку, поэтому в таких системах перед выполнением принято запрашивать необходимую для выполнения программы память, количество процессоров и время их работы. Если запросить недостаточно памяти или процессорного времени и программа превысит эти лимиты, то выполнение прервется, не дойдя до полезного результата. Стоит отметить, что есть задачи, на выполнение которых может потребоваться несколько дней и даже больше, учитывая то, что узлы могут выходить из строя, важную роль при исполнении задач численного моделирования на суперкомпьютерах играют средства отказоустойчивости. Одним из таких средств являются контрольная точка - сохраненное состояние системы, позволяющее продолжить выполнение программы позже без изменения поведения вычислений [1].

Перед выполнением задачи на суперкомпьютере ее необходимо запрограммировать, используя средства для написания параллельных программ. Разработка же параллельных программ требует соответствующей квалификации. Проблему отсутствия квалификации решают системы автоматизации программирования - это совокупность программных средств,

предназначенная для автоматизации написания программ. Некоторые из таких систем позволяют обеспечить отказоустойчивость автоматически. Например, контрольная точка в таких системах может автоматически сохраняться по мере работы программы, а в случае сбоя нужно будет взять последнюю из них и восстановить работу. Стоит отметить, что разработка механизма контрольных точек (подразумевается механизм сохранения контрольных точек и восстановления с контрольных точек) - трудоемкая задача, требующая для каждого отдельного класса задач решать ее заново, так как универсальных методов ее решения может и не найтись.

LuNA [2] - система автоматизации программирования, позволяющая автоматически конструировать параллельные программы. LuNA использует идею фрагментированного программирования - подхода к автоматизации конструирования параллельных программ, при котором параллельная программа состоит из фрагментов во время исполнения, и эти фрагменты могут перераспределяться между вычислительными узлами и исполняться в разном порядке [3]. В LuNA отсутствует механизм контрольных точек, хотя модель вычислений данной системы позволяет оснастить систему автоматическим механизмом контрольных точек, то есть механизмом, который для любой запущенной задачи сможет генерировать контрольную точку универсальную для данной системы формата.

Объектом исследования данной работы является исполнительная подсистема системы LuNA. **Предметом исследования** является разработка механизма контрольных точек в системе LuNA.

Целью данной работы является создание механизма контрольных точек для системы LuNA.

Для достижения поставленной цели был сформирован следующий список задач:

1. Исследовать особенности исполнения фрагментированных программ, а также распределения ресурсов в системе LuNA

2. Разработать формат хранения контрольной точки для LuNA программ

3. Разработать и реализовать алгоритмы сохранения контрольной точки и восстановления с контрольной точки для системы LuNA

Степень разработанности проблемы отказоустойчивости в распределенных системах. Работы по обеспечению отказоустойчивости в распределенных системах ведутся до сих пор, хотя методы такого обеспечения прорабатываться уже долгое время. Связано это с тем, что в разных задачах, решаемых на суперкомпьютерах, есть разные потребности и специфика устройств, которые придется учитывать при разработке механизма контрольных точек. Системы автоматизации используют разные модели вычислений, что также принуждает рассматривать каждую из них в контексте автоматического обеспечения отказоустойчивости посредством контрольных точек, по отдельности. Отметим, что существуют универсальные методы, позволяющие сохранить состояние системы, например, метод представляющий собой моментальные снимки состояния системы. В таком случае будет сохранено состояние памяти и регистров процессора, что позволит перезапустить систему в случае сбоя. Но данный метод привязан к среде, в которой был сохранен снимок и конкретной конфигурации запуска, также он имеет максимально возможный объем данных, которые нужно сохранить. Хотя результаты работ в этой области позволяет почерпнуть данные для обеспечения отказоустойчивости в распределенной системе с уникальной моделью вычислений, но полностью перенять существующий метод не представляется возможным.

Практическая значимость. Разработанных механизм сохранения и восстановления с контрольной точки может быть использован для обеспечения отказоустойчивости в системе LuNA.

Научная новизна работы. Во-первых, разработан формат хранения контрольной точки для системы LuNA, на основе модели вычислений фрагментированных алгоритмов, позволяющий универсально представить

фрагментированную программу. Во-вторых, произведена модернизация алгоритмов и архитектуры исполнительной подсистемы в LuNA, для реализации механизма контрольных точек. После модернизации в системе, запущенной на нескольких узлах, можно сохранять контрольную точку без необходимости согласовывать данные между узлами во время процесса сохранения. В-третьих, был адаптирован алгоритм Ченди-Лэмпорта [4,5] под поставленные условия. Позволяет сохранить состояние фрагментированной программы, используя все узлы для записи в диск.

1 Обзор родственных работ

Необходимо выяснить, какие подходы в создании контрольных точек существуют в системах программирования, и подходят ли они для LuNA. Механизм контрольных точек должен быть автоматическим, в том плане, что пользователь может не принимать участия в работе этого механизма, но, когда случится сбой, сможет воспользоваться результатами его работы и восстановить работу системы.

Одним из базовых методов обеспечения отказоустойчивости является репликация, например, рабочие узлы могут поддерживать согласованное состояние со своей репликой и в случае отказа реплика сможет заменить оригинальный узел [6].

Еще одним из базовых методов повышения отказоустойчивости является сохранение состояния системы для дальнейшего восстановления. Сохранять состояние системы можно разными способами, например, сделать моментальный снимок состояния системы или разработать контрольную точку [1] для данной системы. Оба этих варианта являются эффективными в определенных условиях, но механизм контрольных точек более предпочтителен для распределенных систем, это будет пояснено далее.

Одним из аналогов системы LuNA, обладающих механизмом контрольных точек, является Charm++ [7], он предлагает ряд возможностей обеспечения отказоустойчивости благодаря своему механизму контрольных точек перезапуска. Базовой единицей параллельных вычислений в Charm++ является так называемый chare. Вычисления в Charm++ состоят из большого количества chares, распределенных по доступным процессорам системы и взаимодействующих друг с другом посредством асинхронных вызовов методов. Основная идея, лежащая в основе этого, проста: контрольная точка приложения подобна переносу его chares объектов с процессоров на диски, а перезапуск - обратному. Благодаря таким утилитам миграции, как PUP'ing , пользователи могут решать, какие данные сохранять в контрольных точках и как их сохранять. PUP(Pack/Unpack) - это общий способ описания данных в объекте и

использования этого описания для сериализации. При этом реализованы две схемы обеспечения отказоустойчивости, с завершением работы системы после сбоя и восстановления с контрольной точки после, или автоматического восстановления после сбоя. Данный механизм предполагает участие программиста для, во-первых, определения что сохранять, во-вторых, когда инициировать создание точки сохранения [8, 9].

Существуют подходы к автоматизации конструирования параллельных программ на основе частных моделей вычислений, например, MapReduce [10], используемый в [11, 12]. В данной системе есть главный “мастер” экземпляр программы, который будет управлять работой системы и хранить все структуры данных, описывающие состояние системы. Механизм контрольных точек может быть применен здесь без трудностей реализации, как подмечают и сами авторы статьи, так как нужно лишь сохранить состояния структур “мастера”. Но в использовании контрольных точек в данной системе мало смысла, так как “мастер” узел всего один и вероятность сбоя очень низка, таким образом обеспечение отказоустойчивости в данной системе заключается в хранении уникальной информации на одном узле, который может восстановить остальные узлы.

Есть подходы основанные на естественной отказоустойчивости алгоритма [13]. Параллельный алгоритм обладает естественной отказоустойчивостью, если он способен получить правильный ответ, несмотря на сбой некоторых задач во время вычисления. Дело не в том, что вычисления выполняются другими задачами, а в том, что природа алгоритма такова, что существует естественная компенсация за потерянную информацию, например, итерационному алгоритму может потребоваться больше итераций для сходимости, но он все равно сходится, несмотря на потерянную информацию. Применимость такого метода невысока так как не все задачи могут быть решены подобными алгоритмами.

Есть системы, использующие так называемый системный уровень контрольной точки, например, система Condor [14]. Это метод представляет

собой моментальные снимки состояния системы. Снимки, несомненно, являются рабочим способом повысить степень отказоустойчивости, вдобавок к этому реализация такого метода не является трудновыполнимой задачей. Но механизм контрольных точек обладает большей гибкостью. Контрольная точка в отличии от снимка не привязана ни к физическому оборудованию, ни к операционной системе. Если снимок был сделан в системе с операционной системой S, то и восстанавливать работы программы нужно будет в этой же системе S. У контрольной точки нет такой привязки, ее формат никак не зависит от программного и аппаратного окружения. Помимо этого моментальный снимок состояния системы нельзя восстановить на другом количестве рабочих узлов, что препятствует задачам балансировки нагрузки.

К сожалению, в существующих работах по системам фрагментированного программирования вопрос отказоустойчивости с применением контрольных точек не рассматривался в полной мере. Освящение этого вопроса происходило только в общих чертах. Вариант такого метода и будет рассмотрен в данной работе.

Рассмотренные варианты не предоставляют необходимый метод, удовлетворяющий заявленным требованиям. В некоторых работах механизм контрольных точек производится центральным узлом, что не подходит для распределенного хранения данных. В других предполагают участие программиста для выбора сохраняемых элементов. Рассмотренные варианты не применимы к данному случаю, проблема заключается в разных моделях вычислений таких систем, что не дает возможность сделать универсальный метод отказоустойчивости без недопустимых издержек. Одним из таких универсальных вариантов может быть снимок системы, но как было показано выше, данный метод не подходит под выставляемые требования.

2 Механизм контрольных точек

Данная глава устроена следующим образом. В разделе 2.1 дается информация о системе автоматизации программирования LuNA. В разделе 2.2 происходит постановка задачи в рамках описанной модели. Разделы 2.3.-2.7 посвящены основным результатам работы. В разделе 2.8 подводятся итоги по второй главе.

2.1 Общее представление о системе LuNA

Система LuNA использует идею фрагментированного программирования [3, 15, 16] для исполнения программ, она заключается в том, чтобы рассматривать программу как множество триплетов, пример расположен на рисунке 1, состоящих из входных данных, некоторого фрагмента кода без побочных эффектов и выходных данных. Пример фрагментированной программы можно увидеть на рисунке 2. Если выходные данные одного триплета совпадают с входными данными другого, это значит, что выходные данные одного триплета потребляются другим. Системе требуется отобразить эти триплеты на разные узлы и в местах, где значение находится на одном узле, а потребляется на другом, осуществить пересылку данных. При запуске исполнительной системы на нескольких узлах, на каждом из них будет запущен экземпляр исполнительной системы.

Исполнительная система в LuNA реализована на языке программирования C++ [17] с использованием MPI [18].

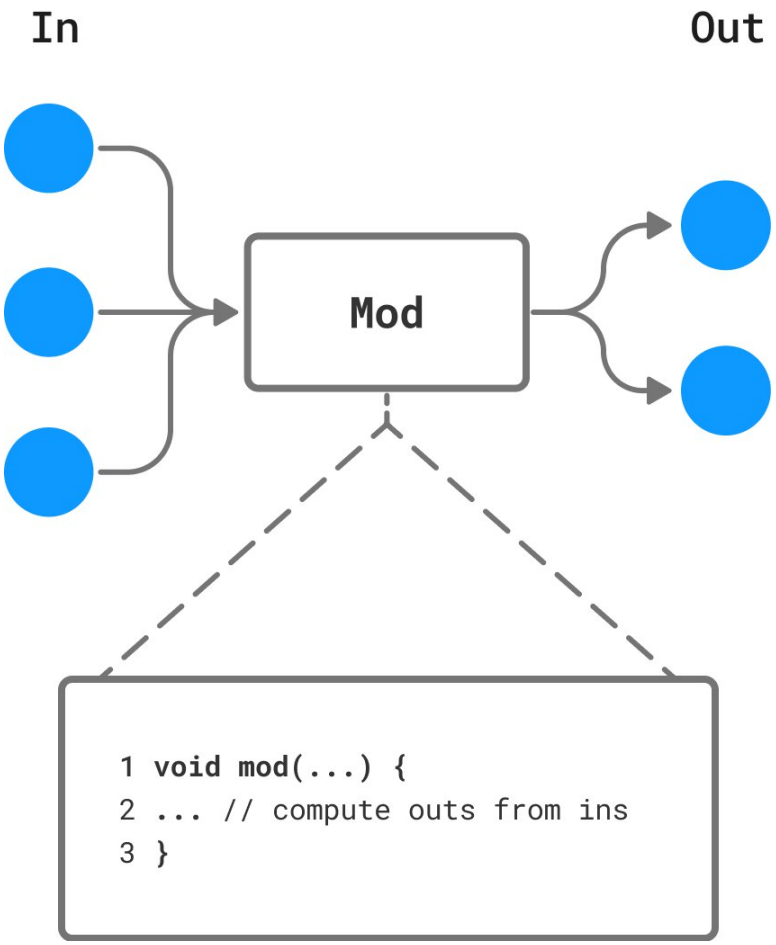

Рисунок 1 - Визуализация триплета

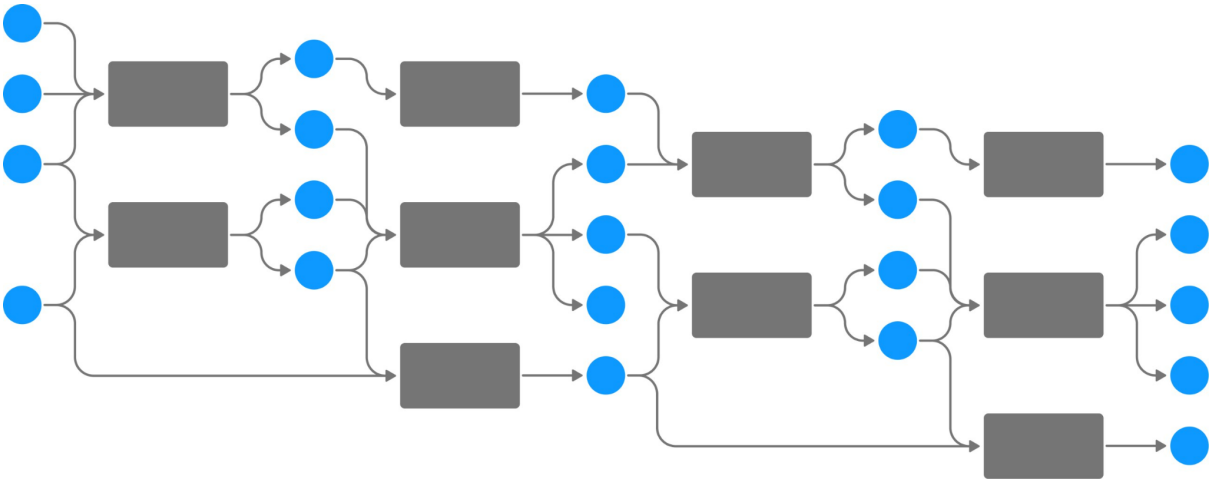

Рисунок 2 - Пример фрагментированной программы

2.2 Как работает исполнительная система в LuNA

Фрагмент вычислений (ФВ) - описанный выше триплет.

Фрагмент данных (ФД) - входные и выходные параметры фрагментов вычислений.

Идентификатор - структура, позволяющая уникально идентифицировать ФВ и ФД. Она представляет собой массив целых чисел.

Экземпляр исполнительной системы состоит состоит из трех модулей:

1. RTS (Run Time System)
2. ThreadPool
3. MpiComm

Для запуска исполнительной системы на некотором количестве рабочих узлов, триплет из этих трех модулей будет запущен на каждом из них.

RTS (Run time system) - это главный управляющий модуль исполнительной системы.

ThreadPool - модуль системы, отвечающий за выполнение задач. В этом модуле некоторое количество потоков (параметр, задаваемый программистом) находятся в постоянном ожидании поступления задачи на выполнение.

MpiComm - сетевой модуль исполнительной системы. Этот модуль отвечает за отправку и получение сообщений от других рабочих узлов. Этот модуль имеет два набора потоков. Первый набор занимается отправкой поступающих сообщений, а второй находится в постоянном ожидании сообщений от других рабочих узлов. Каждый набор имеет некоторое количество потоков (параметр, задаваемый программистом). Все коммуникации между рабочими узлами происходят через этот модуль.

Схему работы системы, запущенной на двух рабочих узлах, можно видеть на рисунке 3.

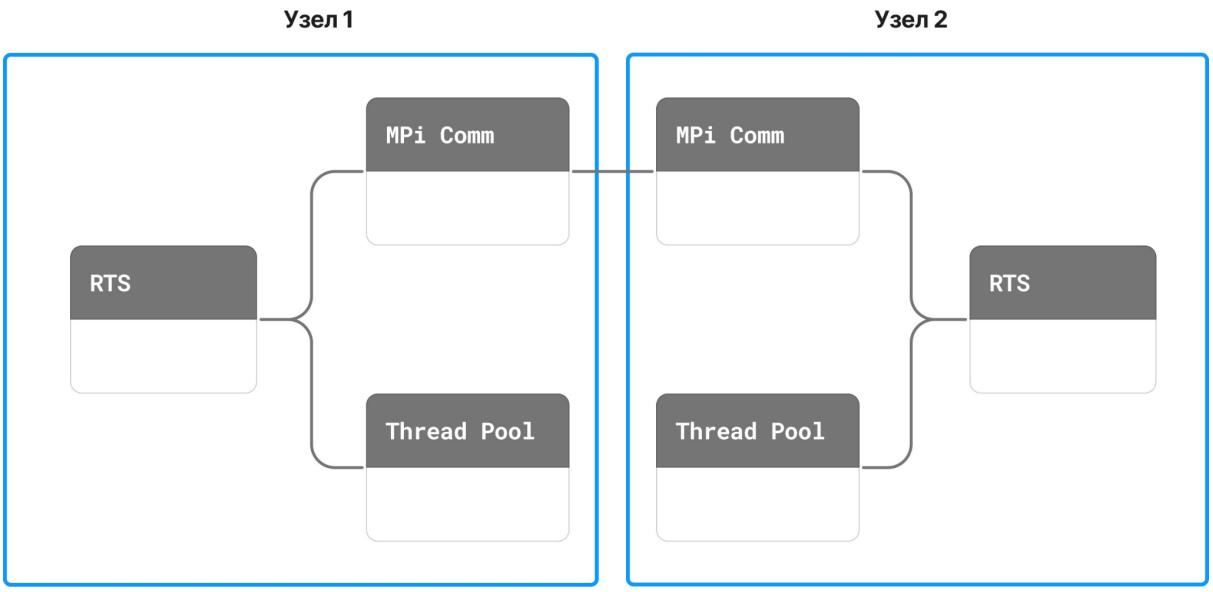

Рисунок 3 - Общая структура исполнительной системы, запущенной на двух рабочих узлах

Исполнительная система в LuNA использует мультиагентный подход. Так фрагментированная структура выражена распределенным по узлам множеством взаимодействующих агентов, каждый из которых управляется императивной программой. Каждому ФВ соответствует один агент. Программа агента похожа на обычную процедуру, которая отличается он нее лишь тем, что сама она разбита на несколько таких процедур, называемых блоками. Между блоками могут происходить остановки, позволяющие перенести агента на другой узел и продолжить выполнение там.

Каждый модуль RTS при старте должен инициализировать свою работу. RTS создает модули MPIComm, ThreadPool и запускает их работу. Для каждого ФВ создается соответствующий ему агент, можно сказать, что ФВ реализуется одним агентом. Созданные агенты передаются в ThreadPool для исполнения. Далее происходит исполнение переданного агента, его выполнение породит новых агентов (и соответствующие им ФВ). Отношение порождения агентов можно изобразить в виде ориентированного графа. В состоянии агента прописано, какой следующий блок ему нужно выполнить, эту информацию выставляет породивший его агент. Первый агент создается только на одном узле (узел с идентификатором 0 в MPI) и вводится в работу описанным выше способом.

Каждый выполненный блок возвращает некоторый код, представленный целочисленным типом данных. Этот код дает указание агенту для дальнейших действий. Представленные коды:

- Код EXIT. Система удалит агента.
- Код MIGRATE. Система сериализует состояние агента и отправляет его другому узлу. Агент же на этом узле будет удален системой.
- Код CONTINUE. Агент продолжит свою работу на текущим узле.
- Код WAIT. Означает, что в исполняемом блоке произошел запрос на ФД. В таком случае агент запрашивает данные с определенным Идентификатором либо с текущего узла, либо с другого узла (отправляя соответствующее сообщение на другой узел, используя MPIComm). Ожидание подразумевает помещение текущего агента в список ожидающих данные с определенным Идентификатором. Когда данные поступают на узел, будет произведена проверка всех ожидающих агентов. Все агенты, ожидающие полученные данные, получат их. Как только все ожидаемые данные будут получены, агент снова будет готов к исполнению и отправиться в модуль ThreadPool.

Фрагмент данных состоит из следующих составных частей:

1. Размер массива данных в байтах.
2. Массив данных. Данные хранятся как массив нетипизированных данных, в зависимости от типа данных они должны будут интерпретироваться как определенный тип.
3. Тип данных. Данные могут быть, напимер, следующих типов: неопределенный тип данных, целочисленное число, вещественное число, и.т.д. Под неопределенным типом данных подразумеваются данные, тип которых система не может идентифицировать, но система может передавать такие данные через сеть или сохранять их. Неопределенные данные могут понимать породившие их агенты или агенты, для которых они предназначаются.

Агент состоит из следующих частей:

1. Номер следующего блока для исполнения.

2. ФД, доступные с самого начала, которые поступают в момент создания.

3. Идентификаторы ФД, необходимые для начала вычислений.

4. Полученные данные. Это отражение идентификаторов полученных ФД в сами полученные ФД.

5. Набор из идентификаторов ожидаемых данных. Как только эта структура станет пуста, это будет означать, что все необходимые ФД получены и можно начинать исполнение.

Формат контрольной точки должен быть зафиксирован, чтобы разные реализации исполнительной системы смогли использовать контрольную точку, сохраненную, в другой ее версии. Исполнительная система в LuNA занимается тем, что исполняет параллельную программу, написать такую исполнительную систему универсальной не представляется возможным, наиболее эффективный вариант будет зависеть от запускаемой программы, в отдельных случаях лучшим решением будет использование нескольких исполнительных систем.

После сохранения состояния программы может случиться так, что запускать ее придется на другой конфигурации суперкомпьютера, ресурсов может стать как больше, так и меньше. Поэтому возникает необходимость в разработке метода распределения нагрузки под новые условия. Такой алгоритм позволит не только использовать точки восстановления в любой конфигурации вычислительной системы, но и позволит намеренно останавливать программу для переноса исполнения в другой среде. Разработка такого алгоритма выходит за рамки данной работы.

2.3 Постановка задачи

Нужно определить, какие данные в исполнительной системе LuNA необходимы для ее работы. Это позволит описать формат контрольной точки, который подойдет для сохранения состояния работы распределенной фрагментированной программы. Разработать механизм остановки системы LuNA, это предполагает остановку всех рабочих узлов, на которых работает

исполнительная система, в частности модуль RTS и их синхронизация. После этого нужно сохранить контрольную точку, согласно разработанному формату. Определить метод старта работы программы посредством контрольной точки.

2.4 Формат контрольной точки

Для сохранения контрольной точки исполнительной системы в LuNA предлагается формат описанный в данном разделе.

Формат контрольной точки исполнительной системы в LuNA - бинарный формат, состоящий таких секций как:

- Базовая часть - содержит служебную информацию для работы с форматом, например, номер версии.

Информация об узле(данный блок будет сохраняться для каждого рабочего узла):

- Агенты, находящиеся на данном рабочем узле.
- ФД, готовые к использованию.
- ФД, которые были запрошены агентами на данном рабочем узле.
- ФД, которые были запрошены агентами с других рабочих узлов.

Формат контрольной точки в качестве составных элементов включает в себя представление таких объектов, как: агенты, ФД, идентификаторы. Перед определением самого формата определим эти составные элементы. Формат структурирован, порядок следования элементов в формате строгий, с таком порядке данные будут записаны в файл контрольной точки.

Формат идентификатора:

1. Размер массива целочисленных (4 bytes int)
2. Массив целый чисел (4 bytes int каждый)

В таблице 1 представлен пример идентификатора, содержащего в себе массив из двух чисел {50, 36} в байтовом представлении.

Таблица 1 - представление идентификатора из двух чисел {50, 36} в шестнадцатеричной системе

Размер массива	Представление “50”	Представление “36”
00000002	00000032	00000024

Формат фрагмента данных(ФД):

1. Размер массива данных в байтах (4 bytes int)
2. Массив данных
3. Тип данных (1 byte char)

Для примера фрагмента данных можно посмотреть на таблице 2.

Таблица 2 - представление ФД, содержащего данные типа “TYPE UNDEFINED”, состоящие из 8 байт в шестнадцатеричной системе

Размер массива (16)	Массив данных из 8 байт	Тип данных, для примера “TYPE UNDEFINED”
00000010	439A4F0084B8C491	00000000

Формат агента:

1. Идентификатор следующего блока для исполнения (4 bytes int)
2. Количество стартовых ФД (4 bytes int)
3. Массив стартовых ФД
4. Количество ФД, которые необходимо получить (4 bytes int)
5. Массив идентификаторов ФД, которые необходимо получить
6. Количество полученных ФД (4 bytes int)
7. Массив пар, состоящий из идентификатора полученного ФД и самого ФД
8. Количество неполученных ФД (4 bytes int)
9. Массив идентификаторов неполученных ФД

Для примера агента можно посмотреть в приложение А.

Формат контрольной точки:

1. Номер версии (4 bytes int)
2. Количество сохраненных узлов (4 bytes int)
3. Массив размеров сохраненных узлов
4. Данные ниже сохраняются для каждого узла:
    a. Информация о всех агентах в системе:
        i. Количество агентов (4 bytes int)
        ii. Массив агентов
    b. Массив агентов для запуска системы:
        i. Массив индексов агентов(4 bytes int каждый)
    c. ФД, готовые к использованию, а именно:
        i. Количество готовых ФД (4 bytes int)
        ii. Массив данных, состоящий из триплета:
            1. Идентификатор ФД
            2. Сколько раз можно использовать данные (4 bytes int)
            3. ФД
    d. Информация о запрошенных ФД с этого узла:
        i. Количество запросов (4 bytes int)
        ii. Массив запрошенных ФД:
            1. Идентификатор запрошенного ФД
            2. Массив запросивших:
                a. Количество агентов, запросивших этот ФД (4 bytes int)
                b. Информация о запросивших:
                i. Порядковый номер агента, запросившего ФД (4 bytes int)
                ii. Номер узла, с которого произошел запрос (4 bytes int)
    e. Информация о запрошенных ФД с этого узла на другой узел:
        i. Количество запросов (4 bytes int)
        ii. Массив информации о запросивших:

1. Идентификатор ожидаемого ФД
2. Массив ожидающих агентов:
    a. Размер массива (4 bytes int)
    b. Массив порядковых номеров агентов, запросивших ФД (4 bytes int каждый)

f. Информация об использованных данных:
    i. Количество использованных ФД (4 bytes int)
    ii. Массив идентификаторов использованных ФД

Массив размеров сериализованных узлов указывается для того, чтобы каждый узел при восстановлении мог за константное время перейти в нужный участок файла с контрольной точкой и прочитать необходимые данные . Отметим, что указанные формат представляет собой сохранение состояние всех узлов в один файл, но он может быть представлен и по-другому. Каждый узел может сохранить свое состояние в отдельный файл, записав туда номер версии и самого себя.

2.5 Предлагаемая идея решения

Для решения был адаптирован алгоритм Ченди-Лэмпорта. Предполагается, что в системе есть рабочие процессе и процесс-наблюдатель. Процесс-наблюдатель инициирует создание снимка и собирает полученные данные. Оригинальная версия алгоритма выглядит следующим образом:

1. Процесс-наблюдатель (создающий снимок):
    a. Сохраняет свое состояние.
    b. Отправляет сообщение запроса снимка, содержащее токен моментального снимка, всем остальным процессам.
2. Процесс, впервые получивший сообщение с токеном снимка:
    a. Отправляет процессу наблюдателю его собственное состояние.
    b. Присоединяет токен снимка ко всем следующим сообщениям (для распространения).

3. Если процесс, получивший токен моментального снимка, следом получает сообщение, которое не несет токен моментального снимка, этот процесс пересылает полученное сообщение в процесс наблюдателя.

LuNA использует MPI-api для коммуникации между процессами, используя которое можно осуществить широковещательную рассылку сообщения, такое сообщение получают все процессы.

Широковещательное (broadcast) сообщение - сообщение, которое будет отправлено всем рабочим узлам, включая отправителя.

Предлагаемый алгоритм:

1. Один из процессов инициирует создание контрольной точки, отправляя широковещательное сообщение.
2. Процессы, получившие сообщения делают следующее:
    a. Запускают процесс остановки, чтобы убедиться, что процессы на время сохранения своего состояния не будут получать сообщения и, следовательно, не изменят своего состояния.
3. С помощью адаптированного алгоритма Дейкстры—Шольтена процесс-инициатор убедиться, что обмен сообщениями больше не происходит и отправит широковещательное сообщение с целью сохранения своего состояния.
4. Процессы, получившие сообщения, делают следующее:
    a. Сохраняют свое состояние.
    b. Оповещают об этом процесс-инициатор.
5. Процесс-инициатор после получения всех уведомлений возобновляет работу системы.

Идея решения заключается в трех этапах. Во-первых, остановить работу исполнительной системы. Остановка система (поставить систему на паузу) нужна для того, чтобы не добавлялись новые агенты и новые запросы, как было описано ранее, в процессе работы системы агенты создают новых агентов, программа агентов может производить запросы на ФД. Поэтому систему нужно остановить, чтобы модули RTS на всех узлах пришли в некоторое определенное

состояние. Во-вторых, произвести сохранение контрольной точки. В-третьих, продолжить работу системы. Если два узла инициировали создание контрольной точки одновременно, то для узла, управляющего сохранением, будет выбран узел с меньшим порядковым номером.

Вышеуказанный способ сохранения контрольной точки не единственный, рассмотрим другие методы ее сохранения. Программист или компилятор может произвести статическую разметку фрагментов вычислений, определяя какой набор фрагментов вычислений должен быть сохранен. После сбоя работы можно будет использовать последнюю полностью сохраненную точку восстановления. Минус такого подхода заключается в снижении гибкости работы с механизмом контрольных точек. Предлагаемый же в этой работе механизм не дает ограничений на вызов контрольной точки из кода программы, если разметка будет производиться автоматически, а также не накладывает на программиста обязанность пометить сотни фрагментов вычислений, если исполняется большая задача. Задачу разметки может решать компилятор, но тогда возникает задача создание такого алгоритма разметки.

2.6 Остановка работы исполнительной системы

Основная идея остановки работы системы заключается в том, что при остановке поступления новых задач на выполнение, задачи со временем закончатся. Для остановки подсистемы выполнения задач необходимо, во-первых, перестать добавлять новые задачи, во-вторых, нужно дождаться пока уже добавленные задачи на момент остановки закончатся - то есть дождаться их выполнения.

Для остановки был выбран следующий алгоритм действий:

1. Инициировать остановку и перестать добавлять задачи в модуле RTS.
2. Определить остановку подсистемы выполнения задач.
3. Определить остановку коммуникационной подсистемы.

2.6.1 Остановка подсистемы выполнения задач

Каждый модуль RTS после получения широковещательного сообщения об остановке перестает направлять новые задачи на исполнение в ThreadPool, начиная складывать их в буферную структуру (массив задач). Так как новые задачи в ThreadPool не поступают, то со временем уже находящиеся в нем задачи закончатся. Отметим, что при выполнении задачи могут порождаться новые задачи, но, как было сказано выше, они попадут в буферную структуру.

Модуль RTS уведомит модуль ThreadPool, что началась остановка, это нужно для того, чтобы ThreadPool запустил алгоритм определения отсутствия задач, когда все рабочие потоки находятся в ожидании новых задач. Ниже приведен алгоритм работы ThreadPool, жирным будет выделена добавленная часть. Для блокировки и снятия блокировки ниже используется примитив синхронизации мьютекс [19]. Для ожидания поступления задач используется условная переменная [20]. Алгоритм устроен так, что в цикле выполняются следующие шаги:

1. Блокировка.
2. Если задач нет:
    a. **Увеличить счетчик определения остановки.**
    b. **Если счетчик равен количеству потоков:**
        i. **ThreadPool остановлен, оповестить RTS.**
    c. Снять блокировку.
    d. Ожидать поступления задач.
    e. **Декрементировать счетчика определения остановки.**
3. Если задачи есть:
    a. Достать задачу.
    b. Снять блокировку.
    c. Выполнить задачу.

Потоки будут пробуждены от ожидания если добавится новая задача. Представленный выше алгоритм можно видеть на рисунке 4.

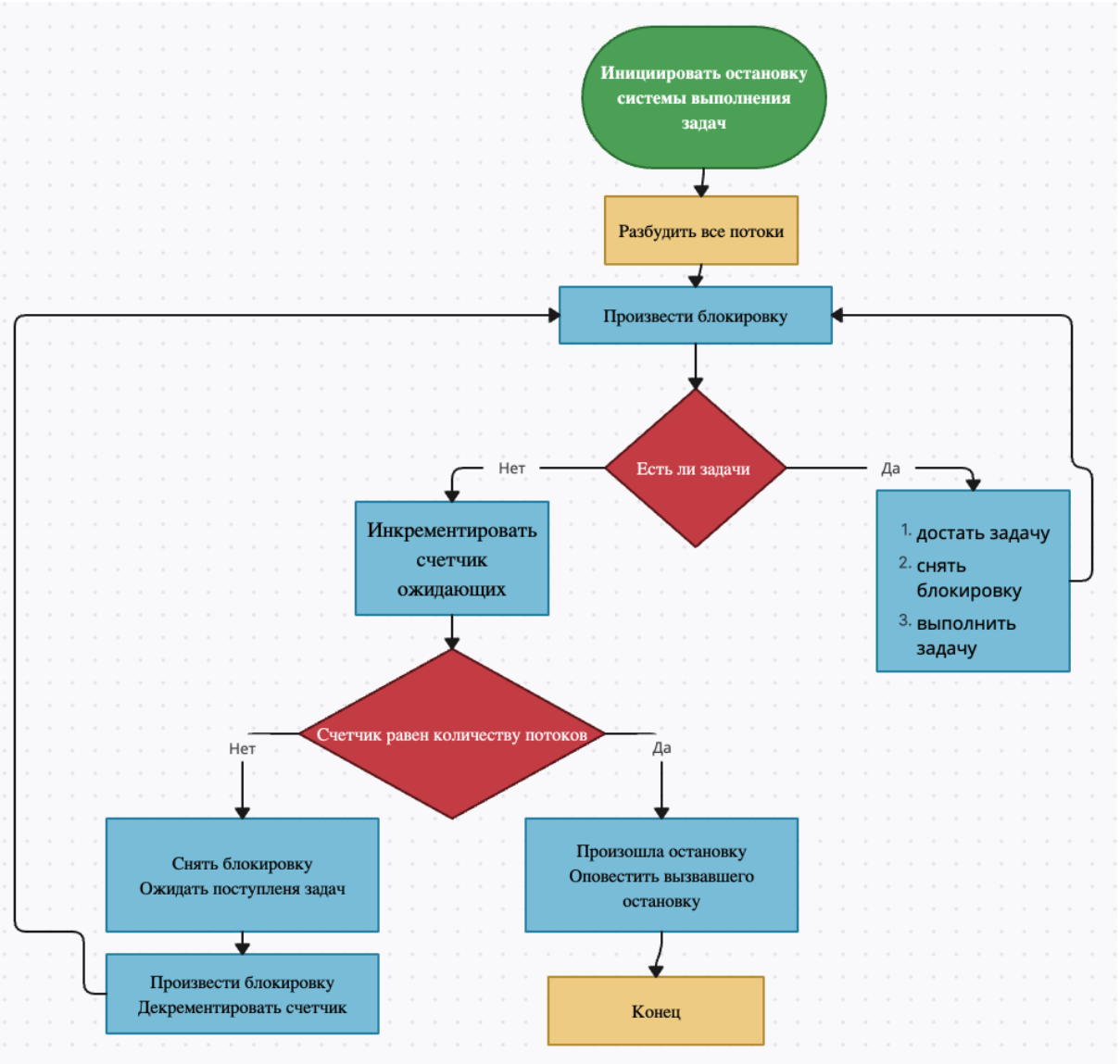

Рисунок 4 - Блок-схема алгоритма работы системы выполнения задач

2.6.2 Остановка коммуникационной подсистемы

Для определения остановки коммуникационного модуля MPIComm был использован алгоритм обнаружения завершения работы системы, а именно адаптация алгоритма Дейкстры—Шольтена к конкретной ситуации. Отметим что модуль, реализующий данный алгоритм уже есть в системе, отдельно он не разрабатывался. Алгоритм подразумевает, что все узлы мультикомпьютера логически организованы в кольцо (желательно, но не обязательно, чтобы узлы, соседние в кольце, были соседями и в сетевой топологии). В алгоритме

используется объект-счетчик, который располагается в каждый момент времени на одном из вычислительных узлов и может передаваться по сети. Также в алгоритме подразумевается, что каждый вычислительный узел может находиться в одном из двух состояний — «есть работа» и «нет работы». Первое означает, что на узле имеется как минимум одно исходящее сообщение, которое еще не доставлено. После этого MPIComm может оповестить RTS об остановке.

2.7 Сохранение контрольной точки и возобновление работы

Формат контрольной точки представлен как единая структура, но базовое сохранение не предполагает, что контрольная точка будет храниться в одном файлу. Каждый рабочий узел сохраняет свое состояние согласно формату в отдельный файл. Название файла при этом будет состоять из базовой части checkpoint и идентификатора. Идентификатор будет целочисленным числом, обозначающим номер узла в MPI.

Когда все узлы сохранят свое состояние, то отправят сообщение о готовности главному узлу. После этого можно возобновить работу, узел инициатор отправляет широковещательное сообщение о готовности всем узлам, после чего снимается блокировка в ThreadPool. Все задачи, сохраненные в буферной структуре, отправляются на исполнение в ThreadPool. Для возобновления работы системы с контрольной точки происходят аналогичные действия, но перед этим сохраненные в файле контрольной точки данные необходимо десериализовать и добавить в систему (Например, информацию о запросов ФД).

2.8 Итоги главы

В главе был рассмотрен механизм контрольной точки, предлагаемый для системы автоматизации программирования LuNA. Предлагаемое решение включает в себя формат представления контролькой точки, алгоритм паузы системы и алгоритм сохранения контрольной точки, использующий паузу системы.

3 Реализация механизма контрольных точек

В разделе 3.1 дается информация о системе автоматизации программирования LuNA в терминах классов. В разделах 3.2-3.4 даются подробности реализации механизма контрольных точек в LuNA. В разделе 3.5 рассмотрены особенности существующей реализации исполнительной системы в LuNA, которые не позволяли внедрить механизм контрольных точек в заявленных требованиях. Также в этот разделе рассказывается, как получилось модифицировать эти особенности. В разделе 3.6 рассматривается тестирование полученного решения. В разделе 3.7 подводятся итоги по третьей главе главе.

Для изложения особенностей реализации необходимо представить систему LuNA в терминах классов.

Для реализованной программы в приложении Б приведено руководство разработчика, а в приложении В – описание программы.

3.1 Архитектура LuNA в терминах классов

Рассмотрим схему работы исполнительной системы для понимания проделанной работы. А именно рассмотрим классы (в смысле ООП[21]) системы и то, за какую работу они отвечают, а также поясним общую схему работы системы. Отмечу, что общая структура связи указана на рисунке 3.

Как говорилось ранее, в исполнительной системе для взаимодействия между рабочими узлами используется модуль MPIComm, необходимые данные об этом модуле можно видеть на листинге 1. Сообщение между узлами отправляется с тегом - это идентификатор типа сообщения целочисленного типа, при получении сообщения другим узлом, действия обработчика будут зависеть от отправленного тега.

Листинг 1 - представление класса MPIComm

```
class MPIComm {
        void send(int tag, message);
        void bcast(int tag);
}
```

Для реализации механизма контрольной точки были добавлены следующие теги: TAG_PAUSE, TAG_MESSAGE_PAUSE_MARKER, TAG_MESSAGE_PAUSE_STOPPED_MARKER. Далее в разделе будет пояснение, для каких целей были добавлены данные теги.

Метод MPIComm::bcast отправляет широковещательное сообщение.

Класс IdleStopper, можно видеть на листинге 2, реализует алгоритм Дейкстры—Шольтена для определения завершения работы системы. Для этого экземпляр класса IdleStopper (далее модуль) отслеживает сообщения поступившие на отправку и отправленные сообщения, а также созданные и удаленные агенты. Для этого внутри класса хранится счетчик, который будет отслеживать количество невыполненных задач в системе. Для управления этим счетчиком у модуля IdleStopper есть метод change_load, который будет обновлять значение счетчика путем сложения своего состояния с пришедшим значением delta. Когда модуль IdleStopper определит, что задач на данном узле не осталось, то вызовет метод send_on_next_node, который отправит сообщение с тегом TAG_IDLE_MARKER, его примет исполнительная система (RTS, будет введено ниже) на другом узле и.т.д.. Обратный вызов (callback) send_on_next_node установит модуль RTS при инициализации. Когда модуль IdleStopper определит что задач больше не осталось ни на одном узле, то вызовет обратный вызов send_on_next_node, который при помощи модуля MPIComm отправит широковещательное сообщение всем узлам с тегом TAG_STOP. При получении этого сообщения узлы завершат свою работу. Обратный вызов on_all_idle_callback также установит модуль RTS при инициализации.

Листинг 2 - представление класса IdleStopper

```
class IdleStopper {
	void change_load(int delta);
	function send_on_next_node_callback;
	function on_all_idle_callback;
}
```

Класс CF - представляется собой агента.

Класс DF - представляется собой ФД.

Класс ID - представляет собой идентификатор агента и ФД.

Как говорилось в теоретическом разделе, в модуле ThreadPool, можно видеть на листинге 3, есть некоторое количество потоков, которые ожидают поступления задач и при поступлении начинают их исполнение. Эта схема устроена следующим образом: в каждом потоке вызывается метод routine, в котором потоки будут в бесконечном цикле проверять наличие задач на исполнение и, при их наличии, исполнять. Если задач нет, то поток переходит в спящее состояние, выйти из которого сможет, когда будет вызван метод ThreadPool::submit, который разбудит один из спящих потоков, более подробно это рассмотрено в разделе 2.6.1..

Листинг 3 - представление класса ThreadPool

```
class ThreadPool {
	void submit(CF);
	void routine();
}
```

Главным классом исполнительной системы является RTS, можно видеть на листинге 4, он инкапсулирует внутри себя всю управляющую логику и хранит объекты вышеупомянутых классов. Метод RTS::submit вызовет метод pool::submit, то есть производит перенаправление задачи в ThreadPool. Структура requests хранит данные о запрошенных ФД и о агентах, которые их запросили. Метод RTS::post вызывается, когда на узел поступил ФД, в таком

случает будет просмотрена структура requests и, если есть агенты, ожидающие эти данные, они будут отправлены ожидающим агентам.

Листинг 4 - представление класса RTS

```
class RTS {
    ThreadPool pool;
    MPIComm comm;
    IdleStopper stopper;
    pair<ID, array<callback>> requests;
    void submit(CF);
    void request(CF, DF)
    vod post(ID, DF);
}
```

Все агенты, представление агента можно видеть на листинге 5, хранят ссылку на экземпляр класса RTS, которому они принадлежат (система может быть запущена на нескольких узлах, в таком случае эта ссылка определяет к какому узлу относится агент). Когда агент потребуются данные, он вызовет метод CF::request, внутри которого будет вызван метод rts::request. Метод RTS::request произведет запрос ФД (у любого узла, включая запросивший). Если данные нужно запросить у самого себя, то будут обновлены структуры данных, отвечающие за информацию о хранении запросов на ФД. Если данные нужно запросить с другого узла, то отправится сообщение с другого узлу с тегом TAG_REQUEST.

Листинг 5 - представление класса CF

```
class CF {
    RTS& rts;
    void request(Id);
}
```

3.2 Остановка подсистемы выполнения задач

Были произведены следующие модификации в структурах классов, для реализации остановки работы системы:

- в класс RTS добавлен метод initiate_checkpoint_create
- в класс RTS добавлена переменная is_pause булевого типа
- в класс RTS добавлен метод `void make_checkpoint`
- добавлен тег TAG_PAUSE
- в класс ThreadPool добавлена переменная is_pause булевого типа
- в класс ThreadPool добавлен функциональный объект thread_pool_pause_callback

Инициировать создание контрольной точки может любой узел в системе, путем вызова соответствующего метода (initiate_checkpoint_create). Для тестовых целей его инициировал один из узлов в заданный момент времени. При инициации создания контрольной точки происходит следующее. Процесс, инициировавший создание контрольной точки, отправляет широковещательное сообщение с тегом TAG_PAUSE. При получении данного сообщения все рабочие узлы выставляют булеву переменную is_pause в значение true. И также выставляет булеву переменную is_pause в значение true для ThreadPool.

Что происходит при выставлении булевой переменной is_pause в значение true в RTS:

- Задачи, которые добавляются через submit не отправляются в ThreadPool::submit, теперь они складываются в буферную структуру данных

В модуле ThreadPool переменная is_pause отслеживается в методе ThreadPool::routine, если эта переменная выставлена в значение true, то потоки перестанут просто переходить в спящее состояние при отсутствии задач. Необходимо определить, что все рабочие потоки находятся в ожидающем состоянии, для этого перед тем как перейти в ожидающий режим все потоки должны инкрементировать специальную переменную и, когда ее значение станет равно количеству рабочих потоков, то можно будет сказать, что все потоки находятся в ожидании поступления новых задач. Первый поток, который

достигнет этого кода, должен разбудить все ожидающие задач потоки. Это необходимо, так как подсчет количества ожидающих потоков будет вестись только при выставлении переменной is_pause в значение true, поэтому для исключения ситуации, когда некоторые потоки уже находятся в ожидающем состоянии, необходимо разбудить все потоки, чтобы пересчитать их. Как только все потоки инкрементировали счетчик и перешли в режим ожидания, последний рабочий поток, инкрементирующий счетчик, вызовет thread_pool_paus_callback для оповещения модуля RTS о завершении остановки. Обратный вызов thread_pool_paus_callback вызовет RTS::make_checkpoint. Этот обратный вызов будет установлен при инициализации RTS. После исполнения thread_pool_paus_callback поток установит переменную is_pause в значение false и разбудит все потоки. Отметим, что до пробуждения всех потоков данный поток выполнит метод RTS::make_checkpoint, в котором произойдет сохранение контрольной точки. Поэтому во время сохранения контрольной точки модуль ThreadPool не будет выполнять задачи.

Отметим, что при выставлении переменной is_pause в ThreadPool, пробуждается один поток. Сделано это для исключения ситуации, когда все потоки уже находятся в ожидании новых задач. Так как для определения такой ситуации потоки необходимо пересчитать и придется разбудить один поток, который в свою очередь разбудит все остальные ожидающие потоки.

В начале метода RTS::make_checkpoint произойдет барьерная синхронизация со всеми узлами, после чего главный узел запустит алгоритм определения остановки работы системы, описанный в следующем разделе, и на всех узлах выполняющий поток перейдет в спящее состояние. Когда алгоритм закончит свою работу и разбудит выполняющий поток, произойдет сохранение контрольной точки, описанное в следующих разделах.

3.3 Остановка коммуникационной подсистемы

Для определения остановки передачи сообщений необходимо определить, что все загруженные для отправки сообщения отправлены. Для достижения

этой цели был использован модуль IdleStopper, для этого в класс RTS нужно добавить еще один объект этого класса (помимо уже имеющегося stopper, используемого для определения остановки работы системы) `IdleStopper pause`. Этот объект должен отслеживать сообщение которое передано на отправку и отправленное. В этот объект нужно установить следующие обработчики:

Когда задачи закончатся:

- отправить сообщение с тегом TAG_MESSAGE_PAUSE_MARKER на другой узел.

Когда модуль IdleStopper определит, что произошла остановка:

- отправит широковещательное сообщение всем узлам с тегом TAG_MESSAGE_PAUSE_STOPPED_MARKER

Сами объекты, как обработчики, будут созданы и установлены при инициализации создания контрольной точки.

При получении сообщения с тегом TAG_MESSAGE_PAUSE_STOPPED_MARKER можно сказать, что система остановлена, и можно произвести сохранение контрольной точки. Узлы, получившие данное сообщение, пробуждают ожидающий поток в методе RTS::make_checkpoint.

Для исключения конфликтов во время остановки (когда переменная RTS::is_pause имеет значение true) RTS::stopper не будет работать. Для этого в обработчике сообщения с тегом TAG_IDLE_MARKER при выставленной в значение true переменной RTS::is_pause отправка дальнейших сообщений остановится, пока переменна RTS::is_pause снова не будет иметь значения false (то есть до окончания работы алгоритма остановки).

3.4 Сохранение контрольной точки и возобновление работы

Для сохранения и восстановления с контрольной точки был создан модуль Checkpoint, можно видеть на листинге 6. В нем есть два метода, принимающие указатель на модуль RTS. Метод saveCheckpoint сохранит состояние переданного модуля RTS(формат сохранения описан в разделе 2.4). В конце

метода RTS::make_checkpoint будет вызван saveCheckpoint, после него будет следовать барьерная синхронизация.

Для возобновления работы в класс ThreadPool был добавлен метод `void submit_tasks()`, который при добавлении задач не пробуждает потоки.

После сохранения контрольной точки метод RTS::make_checkpoint должен возобновить работу системы. Для этого все задачи, которые были помещены в буферную структуру, отправляются на выполнение в модуль ThreadPool при помощи метода ThreadPool::submit_tasks (отметим, что этот модуль все еще находится в состоянии остановки и не будет выполнять задачи), далее буфер очищается, переменная RTS::is_pause выставляется в значение false. После этого рабочий поток возвращается обратно в ThreadPool, выставляет перемену is_pause в значение false и пробуждает все ожидающие потоки. После этого система продолжит работу, как-будто остановки не было.

Метод же startFromCheckpoint восстановит работу с контрольной точки. Для того чтобы запустить программу с контрольной точки в исполнительную систему был добавлен ключ командной строки start_from_checkpoint. Если передать этот ключ при запуске исполнительной системы, то модуль RTS в своем конструктору вызовет startFromCheckpoint, который попытается открыть файлы контрольных точек из директории запуска программы (если их нет, но исполнительная система завершит свою работу с ошибкой) и прочитать с них сериализованные данные.

Листинг 6 - представление класса Checkpoint

```
class Checkpoint {
	void saveCheckpoint(RTS*);
	void startFromCheckpoint(RTS*);
}
```

3.5 Требования к системе и их поддержка

Для сохранения контрольной точки необходимо следующее:

1. Все структуры данных должны быть сериализуемы.

2. Уникальные объекты в каждый момент времени должны храниться только на одном рабочем узле.

В реализации исполнительной системы LuNA эти правила не соблюдаются.

В одном из блоков агента может произойти запрос ФД, он запрашивает его, используя CF::request, который в свою очередь вызовет RTS::request, одним из аргументов которого является несериализуемый функциональный объект, инкапсулирующий внутри себя все необходимые данные и логику. В методе RTS::request происходит проверка, запрашиваемые данные находятся или будут находиться на текущем рабочем узле, или на другом. Если на текущем, то текущий запрос на фрагмент данных, а именно идентификатор данных и функциональный объект, помещается в структуру данных RTS::request. Когда данные будут готовы, поступят в метод RTS::post, то произойдет проверка, есть ли ожидающие эти данные агенты. Если же данные необходимо запросить с другого узла, то создастся еще один функциональный объект, внутри которого будет вызываться уже созданный, и отправится на другой узел. При получении такого запроса, он будет помещен в структуру RTS::requests другого узла.

Удобство такого подхода заключается в том, что, во-первых, для хранения необходимо всего одна структура данных _request, не важно с какого узла пришел запрос. Во-вторых, удобство чтения и понимания, так как при формировании фрагментов вычислений функционального объекта внутрь него инкапсулируется вся логика и необходимые данные, то мы можем рассматривать запрос на данные как что-то однообразное, вне зависимости с какого узла произошел запрос. Даже если агент будет храниться на узле 1, то нет никаких проблем добавить в него данные на узле 2.

Но такая реализация не подходит для достижения поставленных целей по следующим причинам:

1. В модуле RTS есть структуры данных, которые невозможно сериализовать

2. Несколько узлов могут ссылаться на уникальный агент, хотя физически хранится он только на одном узле

Для решения этих проблем были произведены изменения в способе отправки и хранения запросов, рассмотрим их далее.

Первоначальная сериализация агента была неполна, сохранение набора из идентификаторов ожидаемых данных не производилась, хотя для восстановления работы системы это необходимо. Сохранение указанных данных было добавлено.

3.5.1 Избавление от хранения функциональных объектов

При вызове метода RTS::request агент больше не создает функциональный объект, а передает туда самого себя. В методе RTS::request при получении агента он также складывается в структуру requests модуля RTS, но уже в модифицированную версию. Вместо функционального объекта будет храниться указатель на агента, а также идентификатор узла, на котором он был запрошен. Если данные были запрошены у другого узла, то на другой узел будет отправлен указатель на запросившего агента, при получении другой узел запишет в свою структуру requests этого агента и идентификатор запросившего узла. Теперь при получении ФД не будет вызываться функциональный объект, а будет вызвана функция с идентичной логикой внутри, куда будет передан агент. При таком подходе получилось избавиться от несериализуемых функциональных объектов.

Первоначальная структура хранения requests:

- pair<ID, array<callback>>

Новая структура хранения requests:

- pair<ID, array<pair<node_id, CF>>>

3.5.2 Избавление от хранения указателей на агенты на нескольких узлах

При сериализации агента должен сохраниться в единственном экземпляре в контрольной точке, поэтому сохраняемые структуры, которым требуется сослаться на агента, вместо этого должны сохранить некоторую ссылку на агента. Возникает проблема, что такие ссылки могут храниться на нескольких узлах, в таком случае для корректного восстановления необходимо восстанавливать узлы синхронным образом, сначала восстановить агенты на всех узлах, потом все остальные структуры со ссылками на них, также это внесло бы ряд проблем и дополнительных синхронизация при сохранении состояния. Для того, чтобы восстановление могло делаться асинхронно, было принято решение хранить указатели на агенты только на одном узле, для этого был произведен ряд модификаций в хранения запросов и методе коммуникации для запросов.

Если агенту необходим некоторый ФД с другого узла, то он сохраняет эту информацию в структуру deferred_requests, которая хранит ID ожидаемого ФД и ожидающего агента. После этого на другой узел отправляется запрос на эти данные. Другой узел помещает в поле requests нулевой указатель и номер запросившего узла. При поступлении данных на второй узел, они будет направлены на запросивший (первый узел). Первый узел при получении сообщения с ФД просмотрит структуру deferred_requests и оповестит ожидающих агентов.

Поясним, почему вместо deferred_requests не складывать полученные данные в requests. Все ФД для агента берутся из структуры posts, в которой хранится информация о том, сколько можно использовать определенный фрагмент данных. При запросе с другого узла, счетчик использования posts уменьшается на другом узле, следовательно, уменьшать его на запросившем узле не нужно. Поэтому первоначально callback для таких агентов вызывался сразу. Теперь же такие данные нужно также сразу обрабатывать, без изменения структуры posts, поэтому была добавлена отдельная структура.

3.6 Тестирование

Целью тестирования было проверить работоспособность остановки системы, когда система запущена на нескольких узлах. А также проверить работоспособность и время работы механизма сохранения и восстановления с контрольной точки на одном узле.

Первый тест. В первую очередь была протестирована остановка работы исполнительной системы. Для тестирования использовался решатель модельного трехмерного уравнения в единичном кубе [22], где вычисления осуществляются итерационно, каждая итерация решается конвейерным алгоритмом прогонки по каждому из трех направлений. Данный тест был выбран, так как в процессе работы он заполняет все присутствующие в системы структуры данных на всех узлах. Для этого при старте программы главный узел (узел с идентификатором 0 в MPI) инициировал начало остановки системы с разной задержкой, это происходило, когда система была запущена на разном количестве ядер (от 1 до 16). После остановки работа системы возобновлялась и завершалась корректно. По итогу тестирования можно заключить, что остановка системы работает корректно.

Второй тест. Для тестирования создания контрольной точки использовалась задача умножения матриц. В процессе своей работы тесты заполняет все структуры данных исполнительной системы, которые будут использованы для сохранения контрольной точки, поэтому тест является подходящим для проверки сохранения состояний всех структур исполнительной системы. Исполнительная система запускалась на одном узле, далее узел инициировал начало создания контрольной точки. После сохранения контрольной точки работа системы корректно завершалась. Далее система запускалась на одном узле, но уже с контрольной точки. По итогу тестирования можно сделать вывод, сохранение и восстановление с контрольной точки на одном узле работает корректно. Также с тестом умножения матриц были произведены замеры времени с начала инициации контрольной точки, до

восстановления работы системы, результаты которых вы можете видеть в таблице 3.

Таблица 3 - время для сохранения контрольной точки

Сохраняемые параметры	Время с начала инициации контрольной точки, до восстановления работы
Агентов - 3 Запросов - 2 Готовых данных - 5	0.002 сек. 0.379 сек. - время работы всей программы.
Агентов - 1893 Запросов - 185 Готовых данных - 202	0.006 сек. 0.026 сек. - время работы всей программы.
Агентов - 1820 Запросов - 401 Готовых данных - 457	0.01 сек. 0.026 сек. - время работы всей программы.
Агентов - 2004 Запросов - 900 Готовых данных - 1782	0.022 сек. 0.66 сек. - время работы всей программы.
Агентов - 11841 Запросов - 1290 Готовых данных - 2561	0.05 сек. 7.13 сек. - время работы всей программы.

Время работы самого самого теста в данном случае является малозначимой характеристикой, так как время сохранения зависит в большей степени от количества сохраняемых данных. Если в двух разных тестах количество сохраняемых данных будет идентично, то и время сохранения будет идентично и не зависеть от времени работы теста, даже если оно существенно отличается. Тестирование производилось на процессоре AMD Ryzen 7 5800H with Radeon Graphics.

3.7 Итоги главы

В главе рассмотрены особенности реализа механизма контрольных точек в системы автоматизации программирования LuNA. Приводится ряд модификаций, которые были произведены для удовлетворения требованиям

создания контрольной точки. А также поясняются особенности реализации самого алгоритма сохранения контрольной точки. Приводятся результаты тестирования, подтверждающие работоспособность алгоритма остановки для исполнительной системы, работающей на нескольких узлах. А также работоспособность сохранения и восстановления с контрольной точки на одном узле.

ЗАКЛЮЧЕНИЕ

В системе программирования LuNA был разработан механизм контрольных точек. Для единообразного представления контрольных точек был разработан формат контрольной точки. Разработанный метод был реализован в рамках исполнительной системы в LuNA, было произведено его тестирование. Разработанных механизм сохранения и восстановления с контрольной точки может быть использован для обеспечения отказоустойчивости в системе LuNA.

На защиту выносятся следующие положения:

- Формат контрольной точки
- Алгоритм сохранения и восстановления с контрольной точки в системах фрагментированного программирования
- Реализация данного алгоритма в исполнительной системе в LuNA для одного узла

Дальнейшие планы:

1. Добавить реализацию сохранения контрольной точки для нескольких узлов и возможность запускать контрольные точки на другом количестве узлов
2. Изучить способы по инициации создания контрольной точки: как часто нужно ее создавать
3. Добавить в языке LuNA возможность инициировать создание контрольной точки

Выпускная квалификационная работа выполнена мной самостоятельно и с соблюдением правил профессиональной этики. Все использованные в работе материалы и заимствованные принципиальные положения (концепции) из опубликованной научной литературы и других источников имеют ссылки на них. Я несу ответственность за приведенные данные и сделанные выводы.

Я ознакомлен с программой государственной итоговой аттестации, согласно которой обнаружение плагиата, фальсификации данных и ложного цитирования является основанием для не допуска к защите выпускной квалификационной работы и выставления оценки «неудовлетворительно».

Закиров Владимир Константинович _____

(фамилия, имя, отчество студента) (подпись студента)

« » г.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ И ЛИТЕРАТУРЫ

1. Schulz, M. (2011). Checkpointing. In: Padua, D. (eds) Encyclopedia of Parallel Computing. Springer, Boston, MA. https://doi.org/10.1007/978-0-387-09766-4_62
2. LuNA Wiki [Электронный ресурс]. - Режим доступа: https://gitlab.ssd.sscc.ru/luna/luna/-/wikis/home
3. Перепелкин В.А. Система LuNA автоматического конструирования параллельных программ численного моделирования на мультикомпьютерах. дис. ... канд. техн. наук. защищена 12.02.2023. - Режим доступа: https://icmmg.nsc.ru/sites/default/files/sovet/files/dissertaciya_perepyolkin.pdf
4. K. Mani Chandy, Leslie Lamport. Distributed Snapshots: Determining Global States of Distributed Systems. ACM Transactions on Computer Systems, Vol. 3, No. 1, February 1965. https://doi.org/10.1145/214451.214456
5. Алгоритм Ченди – Лэмпорта [Электронный ресурс]. - Режим доступа: https://wiki5.ru/wiki/Chandy%E2%80%93Lamport_algorithm
6. Sari, A. and Akkaya, M. (2015) Fault Tolerance Mechanisms in Distributed Systems. International Journal of Communications, Network and System Sciences, 8, 471-482. doi: 10.4236/ijcns.2015.812042.
7. Официальный сайт Charm++ [Электронный ресурс]. - Режим доступа: https://charmplusplus.org/
8. The Charm++ Programming Language Manual [Электронный ресурс]. - Режим доступа: https://mesonpi.cat.cbpf.br/nextcomp/downloads/charm++.pdf
9. Gengbin Zheng, Lixia Shi, Laxmikant V. Kale. FTC-Charm++: An In-Memory Checkpoint-Based Fault Tolerant Runtime for Charm++ and MPI. 2004 IEEE International Conference on Cluster Computing (IEEE Cat. No.04EX935). doi: 10.1109/CLUSTR.2004.1392606.
10. Jeffrey Dean, Sanjay Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. OSDI'04: Sixth Symposium on Operating System Design and Implementation, San Francisco, CA (2004), pp. 137-150.

11. Официальный сайт Disco [Электронный ресурс]. - Режим доступа: http://discoproject.org/.

12. White, T.. Hadoop: The definitive guide. O'Reilly Media, Inc, 2012. P 17-42.

13. Engelmann, C., Geist, A. (2005). Super-Scalable Algorithms for Computing on 100,000 Processors. In: Sunderam, V.S., van Albada, G.D., Sloot, P.M.A., Dongarra, J.J. (eds) Computational Science – ICCS 2005. ICCS 2005. Lecture Notes in Computer Science, vol 3514. Springer, Berlin, Heidelberg. https://doi.org/10.1007/11428831_39

14. T. Tannenbaum J. B. M. Litzkow and M. Livny. Checkpoint and Migration of Unix Processes in the Condor Distributed Processing System. Technical Report Technical Report 1346, University of Wisconsin-Madison, 1997.

15. Вальковский В.А., Малышкин В.Э. Синтез параллельных программ и систем на вычислительных моделях. — Новосибирск : Наука. Сиб. отд-ние, 1988. — 129 с.

16. В. Э. Малышкин, “Технология фрагментированного программирования”, Вестн. ЮУрГУ. Сер. Выч. матем. информ., 2012, № 1, 45–55.

17. Язык программирования C++ : книга / Бьерн Страуструп. Бином, 2017.— 11 с.

18. Документация Open MPI [Электронный ресурс]. - Режим доступа: https://www.open-mpi.org/doc/

19. Документация IBM, Мьютекс [Электронный ресурс]. - Режим доступа: https://www.ibm.com/docs/ru/informix-servers/12.10?topic=mutexes

20. Документация IBM, Условная переменная [Электронный ресурс]. - Режим доступа: https://www.ibm.com/docs/en/aix/7.1?topic=programming-using-condition-variables

21. Объектно-ориентированный анализ и проектирование с примерами приложений : книга / Гради Буч. Вильямс, 2008.— 67-73 с.

22. Ахмед-Заки Д. Ж., Лебедев Д. В., Малышкин В. Э., Перепелкин В. А. Автоматизация конструирования распределенных программ численного моделирования в системе LuNA на примере модельной задачи // журнал "Проблемы информатики", 2019, № 4, с.53-64. DOI: 10.24411/2073-0667-2019-00016

ПРИЛОЖЕНИЕ А

Пример представления агента в бинарном виде

Покажем представление агента в бинарном виде (в шестнадцатеричной системе). Для представления агента введем несколько ФД с их идентификаторами.

Таблица 4 - представление первого ФД

Размер массива (4)	Данные	Тип данных, для примера “TYPE INT”
00000004	00000001	00000001

Таблица 5 - идентификатор первого ФД

Размер массива	Представление “1”	Представление “1”
00000002	00000001	00000001

Таблица 6 - представление второго ФД

Размер массива (4)	Данные	Тип данных, для примера “TYPE INT”
00000004	00000002	00000001

Таблица 7 - идентификатор второго ФД

Размер массива	Представление “1”	Представление “2”
00000002	00000001	00000002

Таблица 8 - идентификатор третьего ФД

Размер массива	Представление “1”	Представление “2”
00000002	00000001	00000003

Сам третий ФД вводиться не будет, так как не учитывается в данных формата. Тогда представление агента можно увидеть в таблице 9.

Таблица 9 - пример агента в бинарном виде.

Идентификатор следующего блока	000000D3
Количество стартовых ФД	00000000
Массив стартовых ФД.	нет данных
Количество ФД, которые необходимо получить	00000003
Массив Идентификаторов ФД, которые необходимо получить Примечание: Идентификаторы таб. 5, 7, 8	00000002 00000001 00000001
	00000002 00000001 00000002
	00000002 00000001 00000003
Количество полученных ФД	00000002
Массив пар, состоящий из идентификатора полученного ФД и самого ФД Примечание: первые таб. 4, 5, вторые таб. 6, 7	00000002 00000001 00000001 00000004 00000001 00000001
	00000002 00000001 00000002 00000004 00000002 00000001
Количество неполученных ФД	00000001
Массив идентификаторов неполученных ФД	00000002 00000001 00000003

Примечание: так как значение количества стартовых ФД равно нулю, то массив стартовых ФД пустой, поэтому в его строке в таблице написано “нет данных”.

ПРИЛОЖЕНИЕ Б

СОХРАНЕНИЕ И ВОССТАНОВЛЕНИЕ С КОНТРОЛЬНОЙ ТОЧКИ ДЛЯ СИСТЕМЫ LUNA

РУКОВОДСТВО РАЗРАБОТЧИКА

Листов 8

Новосибирск 2023

СОДЕРЖАНИЕ

Аннотация 49
1 Назначение и условия программы 50
1.1 Назначение программы 50
1.2 Функции, выполняемые программой 50
1.3 Условия, необходимые для выполнения программы 50
2 Характеристика программы 51
2.1 Описание основных характеристик программы 51
2.2 Описание основных особенностей программы 51
3 Обращение к программе 52
3.1 Описание процедур вызова программы 52
4 Входные и выходные данные 53
4.1 Организация используемой входной информации 53
4.2 Организация используемой выходной информации 53
5 Лист регистрации изменений 54

АННОТАЦИЯ

В данном документе приведено руководство разработчика для сохранения и восстановления с контрольной точки в системе LuNA. Исходным языком программы является C++. Средство разработки – редактор исходного кода Visual Studio Code.

Основной функцией модуля является остановка системы, сохранение контрольной точки и восстановление с контрольной точки.

Оформление программного документа «Руководство разработчика» произведено по требованиям ГОСТ 19.504-79 «ЕСПД. Руководство программиста» и ГОСТ 19.105-78 «Единая система программной документации (ЕСПД). Общие требования к программным документам (с Изменением N 1)».

1 Назначение и условия программы

1.1 Назначение программы

Сохранение и восстановление с контрольной точки для системы LuNA.

1.2 Функции, выполняемые программой

Модуль производит остановку работы исполнительной системы, создает контрольную точку, также запускает работу исполнительной системы с контрольной точки.

1.3 Условия, необходимые для выполнения программы

Модуль работает в рамках исполнительной системы в LuNA, исполнительная система должна быть запущена на одном узле для сохранения контрольной точки.

2 Характеристика программы

2.1 Описание основных характеристик программы

Модуль позволяет производить остановку системы. Также сохранять контрольную точку и начинать работу с контрольной точки.

2.2 Описание основных особенностей программы

Модуль сохранит контрольную точку в виде файла.

3 Обращение к программе

3.1 Описание процедур вызова программы

В модуль RTS исполнительной системы LuNA был добавлен метод make_checkpoint, который произведет остановку исполнительной системы и использует модуль Checkpoint для сохранения контрольной точки. Контрольная точка будет сохранена в виде файла. Для начала работы исполнительно системы с контрольной точки необходимо передать ее при старте программы. Тогда модуль RTS при инициализации вызовет добавленный метода startFromCheckpoint, который использует модуль Checkpoint для восстановления с контрольной точки.

4 Входные и выходные данные

4.1 Организация используемой входной информации

Если контрольная точка сохраняется, то входных данных нет. Если необходимо начать работу с контрольной точки, то необходимо передать файл контрольной точки при запуске исполнительной системы.

4.2 Организация используемой выходной информации

Контрольная точка будет сохранена в файл.

5 Лист регистрации изменений

Таблица 10 - Лист регистрации изменении описания программы

Лист регистрации изменений									
Изм.	Номера листов (стр				Всего листов (страниц) в докум.	N докум.	Входящий N сопровод. докум. и дата	Подп.	Дата
	Измен.	Замен.	Нов.	Аннулир.					

ПРИЛОЖЕНИЕ В

СОХРАНЕНИЕ И ВОССТАНОВЛЕНИЕ С КОНТРОЛЬНОЙ ТОЧКИ ДЛЯ СИСТЕМЫ LUNA

ОПИСАНИЕ ПРОГРАММЫ

Листов 11

Новосибирск 2023

СОДЕРЖАНИЕ

Аннотация 57
1 Общие сведения 58
1.1 Обозначение и наименование программы 58
1.2 Программное обеспечение, необходимое для функционирования программы 58
1.3 Языки программирования 58
2 Функциональное назначение 59
2.1 Назначение программы 59
2.2 Сведения о функциональных ограничениях на применение 59
3 Описание логической структуры 60
3.1 Алгоритм программы 60
3.2 Используемые методы 60
3.3 Структура программы 60
3.4 Связи между составными частями программы 60
3.5 Связи программы с другими программами 60
4 Используемые технические средства 61
5 Вызов и загрузка 62
6 Входные данные 63
7 Выходные данные 64
8 Лист регистрации изменений 65

АННОТАЦИЯ

В данном документе приведено описание модуля сохранения и восстановления с контрольной точки для системы LuNA. Пользователь может использовать сгенерированную контрольную точку, для восстановления работы исполнительной системы.

Оформление программного документа «Описание программы» произведено по требованиям ГОСТ 19.402-78 «ЕСПД. Описание программы» и ГОСТ 19.105-78 «Единая система программной документации (ЕСПД). Общие требования к программным документам (с Изменением N 1)».

1 Общие сведения

1.1 Обозначение и наименование программы

Полное наименование модуля - Сохранение и восстановление с контрольной точки для системы LuNA

1.2 Программное обеспечение, необходимое для функционирования программы

Модуль функционирует в рамках исполнительной системы в LuNA, поэтому имеет схожие требования.

1.3 Языки программирования

Исходным языком программы является C++. Средство разработки – редактор исходного кода Visual Studio Code.

2 Функциональное назначение

2.1 Назначение программы

Программа предназначена предоставления возможности обеспечения отказоустойчивости LuNA программ.

2.2 Сведения о функциональных ограничениях на применение

В текущей реализации модуль не предназначена для сохранения и восстановления с контрольной точки, если запущена на нескольких рабочих узлах.

3 Описание логической структуры

3.1 Алгоритм программы

Модуль ставит на паузу работу исполнительной системы, производит сохранения контрольной точки в файл и продолжает работу. Модуль можно запустить для начала работы исполнительной системы с контрольной точки, для этого модуль прочитает файл контрольной точки и сохранит прочитанные данные в состояние исполнительной системы.

3.2 Используемые методы

Для остановки исполнительной системы модуль останавливает систему выполнения задач и систему отправки сообщений, после чего узел сохраняет свое состояние. После сохранения состояние работа исполнительной системы продолжается.

3.3 Структура программы

Модуль состоит из класса Checkpoint, в котором есть два метода: saveCheckpoint и startFromCheckpoint. Модуль находиться в файлах checkpoint.h и checkpoint.cc.

В модуль исполнительной системы RTS был добавлен метод make_checkpoint, который инициирует остановку системы и вызовет методы модуля Checkpoint для сохранения состояния.

3.4 Связи между составными частями программы

Связь между модулями программы осуществляется при помощи вызова функций и методов объектов.

3.5 Связи программы с другими программами

Модуль можно использовать только в рамках исполнительной системы LuNA.

4 Используемые технические средства

Режим работы – работает в качестве модуля исполнительной системы в LuNA.

5 Вызов и загрузка

Запуск модуля осуществляется путем вызова соответствующего метода при работе в исполнительной системе в LuNA.

6 Входные данные

Если контрольная точка сохраняется, то входных данных нет. Если необходимо начать работу с контрольной точки, то необходимо передать файл контрольной точки при запуске исполнительной системы.

7 Выходные данные

На выходе работы получится файл, содержащий в себе контрольную точку определенного формата.

8 Лист регистрации изменений

Таблица 11 - Лист регистрации изменении описания программы

Лист регистрации изменений									
Изм.	Номера листов (стр				Всего листов (страниц) в докум.	N докум.	Входящий N сопровод. докум. и дата	Подп.	Дата
	Измен.	Замен.	Нов.	Аннулир.					