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 09.03.01 Информатика и вычислительная техника
Направленность (профиль): Программная инженерия и компьютерные науки

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА БАКАЛАВРА

Гемуева Александра Булатовича

Тема работы:

РАЗРАБОТКА И РЕАЛИЗАЦИЯ СЕТЕВОГО ПРОТОКОЛА ВЗАИМОДЕЙСТВИЯ ИСПОЛНИТЕЛЬНЫХ СИСТЕМ ФРАГМЕНТИРОВАННОГО ПРОГРАММИРОВАНИЯ

«К защите допущена»
Заведующий кафедрой,
д.т.н., профессор
Малышкин В.Э. /……
(ФИО) / (подпись)
«…» мая 2024г.

Руководитель ВКР
д.т.н, профессор,
зав. каф. ПВ ФИТ
Малышкин В. Э. /……
(ФИО) / (подпись)
«…» мая 2024г.

Соруководитель ВКР
к.т.н,
ст. преп. каф. ПВ ФИТ
Перепёлкин В.А. /……
(ФИО) / (подпись)
«…» мая 2024г.

Новосибирск, 2024

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 09.03.01 Информатика и вычислительная техника
Направленность (профиль): Программная инженерия и компьютерные науки

УТВЕРЖДАЮ

Зав. кафедрой Малышкин В.Э.
(фамилия, И., О.)
…….
(подпись)
«3» ноября 2023г.

ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ БАКАЛАВРА

Студенту Гемуеву Александру Булатовичу, группы 20209
(фамилия, имя, отчество, номер группы)
Тема работы: Разработка и реализация сетевого протокола взаимодействия исполнительных систем фрагментированного программирования
(полное название темы выпускной квалификационной работы)
утверждена распоряжением проректора по учебной работе от 3 ноября 2023 №0413
Срок сдачи студентом готовой работы 20 мая 2024 г.
Исходные данные (или цель работы): Разработка и реализация сетевого протокола взаимодействия исполнительных систем фрагментированного программирования.
Структурные части работы:
Обзор существующих средств организации взаимодействия параллельных программ, разработка протокола, описание реализации, тестирование.

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

Задание принял к исполнению
Гемуев А.Б. /……
(ФИО студента) / (подпись)
«3» ноября 2023г.

Соруководитель ВКР
к.т.н,
ст. преп. каф. ПВ ФИТ
Перепёлкин В.А. /……
(ФИО) / (подпись)
«3» ноября 2023г.

СОДЕРЖАНИЕ

Введение 5

1 Обзор средств организации взаимодействия параллельных программ..... 8

1.1 MPI 8

1.2 Charm++ 8

1.3 Apache Hadoop 9

1.4 Системы с распределенной общей памятью 9

1.5 Система LuNA 9

1.6 Вывод по обзору 10

2 Разработка протокола взаимодействия исполнительных систем фрагментированного программирования 11

2.1 Фрагментированный алгоритм 11

2.2 Проблемы организации взаимодействия 11

2.3 Форматы кодирования данных 17

2.4 Модель протокола 21

2.5 Базовый протокол взаимодействия 22

2.6 Анализ разработанного протокола 24

3 Реорганизация исполнительной системы LuNA для работы по разработанному протоколу 26

3.1 Архитектура сетевого модуля 26

3.2 Работа с технологией Google Flatbuffers 30

3.3 Работа с сетевыми модулями 32

3.4 Оптимизация доставки сообщений 33

3.5 Расширения протокола 35

4 Организация взаимодействия по разработанному протоколу между различными реализациями исполнительных систем LuNA 37

Заключение 39

Список использованных источников и литературы 40

Приложение А 43

Приложение Б 47

ВВЕДЕНИЕ

Написание программ, работающих на мультикомпьютерах, представляет собой сложную задачу, требующую понимания как технических аспектов программирования, так и особенностей работы распределенных систем. Разработчики должны иметь понимание алгоритмов параллельного программирования и распределенных вычислений. Это включает в себя знание методов разделения задач, синхронизации потоков и обмена данными между узлами. Для эффективной работы программы на мультикомпьютере необходимо учитывать архитектурные особенности и ресурсы каждого узла. Это требует оптимизации программного кода под конкретную аппаратную конфигурацию, а также управления нагрузкой и балансировки нагрузки по узлам. В связи с этим возникает потребность в существовании средств автоматизации для конструирования параллельных программ.

Сконструированная параллельная программа выполняется с помощью исполнительной системы. Универсальная и эффективная исполнительная система не может существовать, поскольку для решения конкретных классов задач существуют специфические эвристические подходы и алгоритмы. Например, система, оптимизированная для операций линейной алгебры, может предоставить значительное ускорение вычислений в этой области, но она может оказаться неэффективной для других типов задач. Точно так же система, использующая специализированные вычислители, такие как FPGA или GPU, может быть эффективной при выполнении определенных видов вычислений, но может оказаться неадаптированной к другим видам вычислений. Поэтому, для эффективного решения некоторых вычислительных задач может потребоваться решение подзадач с помощью различных исполнительных систем. Это, в свою очередь, требует организации их совместного взаимодействия. Поэтому обеспечение взаимодействия исполнительных систем является актуальной проблемой.

Обеспечение взаимодействия исполнительных систем в общем случае является сложной задачей. Каждая из систем имеет свою модель вычислений,

архитектуру, принципы работы и особенности реализации. Кроме того, со временем количество реализаций будет только увеличиваться.

Несмотря на то, что решение общей проблемы является труднодостижимым, можно разработать практически-значимый механизм взаимодействия для некоторого класса исполнительных систем. Например, можно разработать сетевой протокол для класса исполнительных систем LuNA [1].

Целью работы является разработка сетевого протокола для исполнительных систем LuNA. Для достижения поставленной цели необходимо выполнить следующие **задачи**:

1) Провести анализ проблем, возникающих при организации взаимодействия исполнительных систем фрагментированного программирования.
2) Разработать протокол, решающий рассматриваемые проблемы.
3) Реорганизовать реализацию системы LuNA на взаимодействие по разработанному протоколу.
4) Произвести тестирование разработанного протокола.

Научная новизна работы состоит в том, что был разработан протокол взаимодействия для исполнительных систем фрагментированного программирования, обладающий свойством расширяемости и подходящий для решения широкого класса задач.

Практическая ценность работы обусловлена тем, что благодаря разработанному протоколу стало возможным объединение различных исполнительных систем для создания более эффективных программ.

Работа изложена в четырех главах. Первая глава содержит обзор существующих решений для организации взаимодействия параллельных программ. Вторая глава посвящена разработке протокола. Третья глава содержит практические аспекты реализации. В четвертой главе производится применение разработанного протокола.

Приложение А является описанием схемы данных разработанного протокола.

Приложение Б является руководством программиста разработанного сетевого модуля.

1 Обзор средств организации взаимодействия параллельных программ

Для того, чтобы организовать взаимодействие исполнительных систем LuNA, необходимо ознакомиться с существующими методами организации взаимодействия параллельных программ, изучить их преимущества и недостатки, а также использовать и адаптировать некоторые из подходов и концепций.

1.1 MPI

MPI (Message Passing Interface) [2] — одна из самых распространённых технологий программирования для параллельных вычислительных систем. Существуют бесплатные и коммерческие реализации MPI почти для всех платформ и видов сетей, а также реализации для различных языков программирования.

MPI-программа состоит из множества параллельных взаимодействующих процессов, которые работают в своих адресных пространствах. Основным способом взаимодействия между процессами является явный обмен сообщениями. Модель передачи сообщений в MPI была расширена за счёт асинхронных и коллективных операций, объединения процессов в группы.

Несмотря на предоставляемые возможности, написание корректно работающих MPI-программ требует определённой квалификации в параллельных вычислениях, что препятствует использованию технологии широким кругом специалистов в своих предметных областях.

1.2 Charm++

Charm++ [3] — это система параллельного программирования на базе языка C++, основанная на модели мигрирующих объектов и обладающая мощной исполнительной системой. Ключевой особенностью Charm++ является декомпозиция: программист разделяет программу на большое количество объектов и определяет вычисления в терминах создания и взаимодействия между этими объектами, без прямого указания вычислителя, на котором находится тот или иной объект.

Вычислительная модель Charm++ основана на использовании множества объектов, называемых «чарами» (chares). Эти объекты распределяются по всем доступным процессорам системы и взаимодействуют друг с другом посредством асинхронных вызовов методов. Асинхронные вызовы методов реализованы с помощью обмена сообщениями между исполнительными системами.

1.3 Apache Hadoop

Apache Hadoop [4] — это платформа с открытым исходным кодом для распределенного хранения и обработки больших и несвязанных между собой данных. Apache Hadoop реализует модель MapReduce [5]. MapReduce — это парадигма распределенных вычислений, предназначенная для выполнения параллельных операций над крупными наборами данных в компьютерных кластерах.

В Hadoop межпроцессорные взаимодействия между узлами системы реализуются с помощью механизма RPC [6] на основе технологии Apache Avro [7]. Apache Avro позволяет строить JSON-схемы данных [8], при этом сериализованные данные хранятся в компактном двоичном формате.

1.4 Системы с распределенной общей памятью

Системы Unified Parallel C [9] и Coarray Fortran [10] основаны на использовании распределенной общей памяти [11]. В таких системах программист может писать код так, как если бы память была общей для всех процессоров. Однако на практике доступ к локальной и удаленной памяти имеет различную стоимость в плане времени доступа и пропускной способности. Игнорирование стоимости доступа к памяти приводит к написанию неэффективных программ.

1.5 Система LuNA

Для автоматизированного создания параллельных программ численного моделирования на мультикомпьютерах, в ИВМиМГ СО РАН разрабатывается язык и система LuNA, реализующая парадигму фрагментированного программирования.

В процессе выполнения программы исполнительная система LuNA обменивается сообщениями по сети. Однако в существующих реализациях формат сообщений задан неявно. Это означает, что вид сообщений, их содержание, порядок следования полей сообщения и бинарный формат зависят от конкретной реализации исполнительной системы. Такая неявная спецификация протокола влечет за собой ряд проблем. Во-первых, несовместимость реализаций исполнительных систем становится серьёзным барьером для их совместной работы. Различия в форматах сообщений и их обработке могут привести к непредсказуемому поведению приложений и потребовать дополнительных усилий для обеспечения их совместимости. Во-вторых, проблемы, связанные с кодированием бинарных данных, могут возникнуть из-за различий в оборудовании на гетерогенных мультикомпьютерах, например, различный порядок байтов при кодировании целых чисел. Это может привести к ошибкам интерпретации данных и некорректному выполнению программы. Кроме того, изменение схемы сообщений или добавление новых типов сообщений становится непростой задачей. Необходимость согласования изменений между различными реализациями исполнительных систем существенно затрудняет разработку и поддержку.

1.6 Вывод по обзору

Таким образом, существует большое количество способов организации взаимодействия исполнительных систем. Среди них можно выделить два класса – системы, построенные на основе обмена информационными сообщениями и системы, построенные на основе распределенной общей памяти. Стоит заметить, что в системах, построенных на основе распределенной общей памяти, неизбежно возникают проблемы, связанные с ее неоднородностью, такие как время доступа и пропускная способность. Поэтому более предпочтительным способом организации взаимодействия исполнительных систем является явный обмен сообщениями.

2 Разработка протокола взаимодействия исполнительных систем фрагментированного программирования

Используемые понятия и терминология.

- Сообщение – форма представления информации, имеющая признаки начала и конца, предназначенная для передачи через среду связи.
- Протокол – набор соглашений, которые определяют обмен данными между различными программами. Эти соглашения задают единообразный способ передачи сообщений и обработки ошибок.
- Сериализация – это процесс преобразования данных в бинарный формат, пригодный для хранения и передачи. Обратный процесс – десериализация.

2.1 Фрагментированный алгоритм

Введем неформально понятие фрагментированного алгоритма (ФА) [12] и связанных с ним сущностей.

Фрагментированный алгоритм состоит из множества фрагментов вычислений (ФВ), связанных информационными зависимостями. ФВ представляет собой некую вычислимую процедуру без побочных эффектов, принимающую и порождающую фрагменты данных (ФД). ФД содержит в себе некоторые данные и обладает свойством иммутабельности: после порождения ФД его значение не меняется.

Два ФВ находятся в отношении зависимости если и только если множество входных ФД одного ФВ пересекается с множеством выходных ФД другого ФВ. При этом отношение зависимости фрагментов вычислений является отношением частичного порядка.

Выполнение ФА сводится к выполнению ФВ в любом порядке, не противоречащем отношению зависимости, в т.ч. параллельно.

2.2 Проблемы организации взаимодействия

Для разработки протокола необходимо решить ряд проблем, возникающих при организации взаимодействия исполнительных систем. В каждом из подразделов 2.2.1 – 2.2.5 рассматривается проблема, после чего предлагается и обосновывается способ ее решения.

2.2.1 Синхронное и асинхронное взаимодействие

В системах, где используется синхронное взаимодействие между компонентами, возникает проблема зависания отправителя, пока получатель не обработает запрос [13]. Это может привести к задержкам или даже остановкам в работе системы, особенно если получатель обрабатывает запрос длительное время или зависает. Такое взаимодействие часто реализуется через блокирующие вызовы методов или ожидание подтверждения завершения операции, что делает поток выполнения предсказуемым, но вносит существенные недостатки в быстродействие системы.

Поскольку модель ФА не определяет порядок выполнения ФВ (кроме информационных зависимостей между ними), возникает недетерминизм порядка вычисления ФВ. В этом случае, если одна ИС, например, запрашивает ФД у другой ИС, ожидание получения ответа может составлять длительное время.

В качестве решения этой проблемы предлагается переход к асинхронному взаимодействию между исполнительными системами. В асинхронной модели отправитель продолжает свою работу, не дожидаясь завершения операции получателем. Это снижает риск задержек и остановок, поскольку отправитель не блокируется, система может продолжать функционировать даже при замедлении или сбоях одного из компонентов.

2.2.2 Поиск объектов и доставка сообщений

Семантика многих сообщений разрабатываемого протокола неизбежно связана с сущностями модели ФА. В частности, это означает, что для доставки сообщений необходимо разработать механизм определения местоположения двух основных видов объектов - фрагментов данных и фрагментов вычислений. Под местоположением ФД понимается ИС, где хранится значение данного ФД, в то время под местоположением ФВ подразумевается ИС, на которой ФВ исполняется. Сложность определения местоположения возникает из-за возможности миграции ФВ и перераспределении ФД между исполнительными системами, например, в результате работы алгоритмов динамической балансировки нагрузки по узлам.

Поскольку необходимо обеспечивать масштабируемости системы при работе на мультикомпьютерах с большим числом узлов, использование централизованных хранилищ в протоколе не представляется возможным. Во-первых, хранение полной информации о местоположении объектов на одном или нескольких узлах невозможно из-за ограничений по объему памяти. Во-вторых, дополнительные временные затраты на доступ к таким узлам будут слишком высокими. Следовательно, протокол должен внедрять децентрализованный масштабируемый механизм поиска объектов и доставки сообщений.

Для решения данной проблемы в работе [12] было введено определение локационной функции (локатор). Локационная функция определяется следующим образом: next = locate (current, id, t), где next – идентификатор следующей ИС, current – текущая ИС, id – идентификатор объекта, t – текущий момент времени. Предполагается, что локационная функция за конечное число шагов приведет к искомому объекту.

В работе предлагается использовать некоторое обобщение локационной функции. В рамках данного обобщения предполагается, что локационная функция задает некоторый алгоритм, который по состоянию исполнительной системы определяет идентификатор следующей ИС, на которую следует доставить сообщение.

2.2.3 Доставка сообщений в соответствии с сетевой топологией

Исполнительные системы могут быть распределены по узлам мультикомпьютера различным образом. При этом, из-за сетевой топологии обычно не поддерживается свойство прямого обмена данными между любыми двумя узлами. Из-за этого возникает ситуация, когда сообщение проходит через промежуточный узел и исполнительную систему. При этом промежуточная исполнительная система может не поддерживать локационную функцию, содержащуюся в сообщении. Помимо этого, семантика алгоритма локационной функции для вычисления следующего шага может требовать состояние ИС, на которую в данный момент направляется сообщение. Возможным способом решения данной проблемы является добавление в модель сообщения

идентификатора следующей ИС, вычисленный с помощью локационной функции. По данному идентификатору возможна маршрутизация сообщения через промежуточные исполнительные системы.

Необходимо подчеркнуть, что несколько исполнительных систем могут использовать общий сетевой модуль. Тем не менее все представленные рассуждения сохраняют свою применимость и в этом случае.

Рассмотрим доставку сообщения между исполнительными системами, изображенными на рисунке 1.

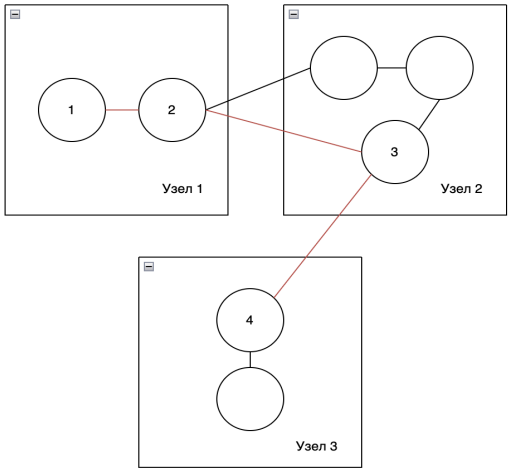

Рисунок 1 – Доставка сообщения между исполнительными системами

Изначально у ИС-1 имеется некоторое сообщение. При этом несущественно, сформировано это сообщение на ИС-1, или же получено от другой ИС в процессе доставки по локатору. Процесс доставки сводится к следующим шагам:

1) ИС-1 определяет с помощью локатора идентификатор следующей ИС, в данном случае ИС-4, после чего добавляет вычисленный идентификатор в сообщение.
2) Сообщение доставляется через ИС-2 и ИС-3. При этом данные исполнительные системы используют только идентификатор, добавленный в сообщение на предыдущем шаге.
3) Получив сообщение, ИС-4 достает из него локатор, с помощью которого определяет следующую исполнительную систему.

Решение проблемы изменения значения локационной функции сообщения во время доставки описано в разделе 3.4.

2.2.4 Обработка исключительных ситуаций

Во время работы исполнительных систем могут происходить исключительные ситуации. Например, исполнительная система получает сообщение, которое она не поддерживает или не способна обработать. Возможной стратегией разрешения ситуации является аварийное завершение программы, однако оно не всегда применимо.

Рассмотрим проблему на примере ситуации, когда ИС получает сообщение с некоторыми данными, формат которых она не поддерживает. Вместо аварийного завершения программы, ИС может попробовать повторно запросить данные в поддерживаемом формате. Стоит заметить, что посылать сообщение обратно также следует по локатору, поскольку ИС, обрабатывающая исключение, может измениться, например, если ФД был передан на другой узел.

В любом случае ИС, обрабатывающая ошибку, должна получить в необходимые данные для обработки. В простейшем случае такими данными может выступать уникальный идентификатор сообщения, вызвавшего ошибку.

2.2.5 Обнаружение завершения работы

Во время исполнения фрагментированного алгоритма ФВ порождает новые ФВ. В какой-то момент последний порожденный ФВ заканчивает свое выполнение, после чего система должна завершить свою работу. Проблема обнаружения завершения работы распределенного вычисления известна в англоязычной литературе как detecting termination of diffusing computation.

Под диффундирующим вычислением [14] понимается вычисление, исполняемое взаимодействующими процессами и обладающее следующими свойствами:

– Каждый процесс бывает либо активным, либо пассивным.
– Если процесс получает сообщение, он становится активным.
– Только активный процесс может посылать сообщения.

– Активный процесс может в любой момент по своему желанию стать пассивным.

Диффундирующее вычисление остановилось, если все процессы пассивные и нет сообщений в пути, поскольку они могут изменить состояние получателя из пассивного в активный. В качестве взаимодействующих процессов выступают ИС. Активное состояние ИС определяется наличием хотя бы одного незавершенного ФВ. При этом стоит заметить, что если ФВ мигрирует на другую ИС, то до завершения миграции он считается незавершенным на исходной ИС.

Данная проблема является хорошо разработанной. Например, данную проблему решают алгоритмы Дейксты-Шолтона [15] и алгоритм Ханга [16]. В работе предлагается использовать адаптацию алгоритма Дейксты-Шолтона, изложенную в [12]. Алгоритм предполагает, что исполнительные системы логически объединены в кольцо.

Опишем алгоритм в виде набора свойств:

– В каждый момент времени ровно одна ИС содержит счетчик. Изначально значение счетчика равно нулю.
– Каждая ИС содержит флаг, изначальное значение флага равно false.
– Как только ИС, содержащая счетчик переходит в пассивное состояние, значение счетчика инкрементируется, флагу присваивается false, после чего счетчик передается следующей ИС в кольце.
– Если со времени последней передачи флага ИС переходила в активное состояние, значение флага устанавливается в true.
– Если в момент получения счетчика ИС находится в активном состоянии, или значение флага установлено в true, значение счетчика обнуляется.
– Если значение счетчика достигло 2N, где N - количество исполнительных систем, работа системы завершена.

Счетчик достигает значения 2N тогда и только тогда, когда в системе больше не осталось работы. При этом счетчик совершает не более 3N пересылок.

Достоинством данного алгоритма являются низкие накладные расходы для его реализации. Действительно, если ИС загружены, счетчик не передается в принципе. Недостатком является тот факт, что для обнаружения завершения работы счетчик должен завершить до трех полных проходов по множеству исполнительных систем.

2.3 Форматы кодирования данных

Для обеспечения взаимодействия между процессами необходимо определить формат кодирования данных. Формат кодирования данных определяет необходимые атрибуты, такие как поддерживаемые типы данных и способ организации сообщений. Существует два класса форматов кодирования: текстовые и бинарные. Текстовые форматы, такие как XML и JSON являются распространенным способом кодирования данных в алгоритмах прикладного уровня. Главным преимуществом текстовых форматов является возможность их чтения человеком. Однако такие форматы не подходят для организации передачи больших объемов данных, поскольку размер закодированных сообщений кратно, или даже на порядки превышает размер аналогичных сообщений, закодированных с помощью бинарных форматов. Поэтому для кодирования данных необходимо использовать двоичные форматы.

На данный момент существуют десятки технологий сериализации, и чтобы выбрать подходящую, необходимо сформулировать ряд требований.

2.3.1 Наличие IDL

Многие технологии сериализации, помимо форматов кодирования данных предоставляют язык описания интерфейсов, известный в англоязычных источниках как IDL (Interface description language) [17]. С помощью IDL описывают структуры и типы данных без привязки к конкретным языкам программирования. IDL широко используется для организации удаленного вызова процедур (RPC). Однако даже без использования RPC в организации взаимодействия между ИС, IDL является удобным инструментом для описания моделей сообщений.

2.3.2 Спецификация и API

Формат бинарного кодирования технологии должен быть стабильным и специфицированным. Во-первых, это позволяет поддерживать библиотечное API для различных языков программирования. Во-вторых, спецификация формата необходима, например, для самостоятельной реализации сериализации и десериализации сообщений на вычислителях или платформах, которые напрямую не поддерживаются технологией.

2.3.3 Поддержка операций без копирования

Поддержка операций без копирования, также известных как zero-copy [18], означает возможность чтения и записи данных в определенных частях сообщения в сериализованном виде. Это требование необходимо для достижения приемлемых нефункциональных свойств в обработке сообщений исполнительной системой. Рассмотрим это требование на примере следующей ситуации.

Во время работы ИС может получить сообщение, которое необходимо переслать на другую ИС в соответствии с вычислением локационной функцией. При этом сообщение может иметь значительный размер и сложную структуру, однако ИС требуется прямой доступ только для извлечения данных локационной функции. Соответственно, при применении zero-copy операции есть возможность сэкономить время обработки на разборе сообщения, а также количество используемой памяти, например, при отправке сообщения из того же буфера, в который оно было получено.

2.3.4 Поддержка архитектурно-независимого способа кодирования данных

Одной из наиболее распространенной проблемой совместимости между разными платформами является порядок следования байтов при кодировании целых чисел, также известный как endianness. Примером решения проблемы выступает фиксация формата кодирования при обмене и обработки. Технология сериализации должна предоставлять подобный механизм.

2.3.5 Сравнение технологий сериализации

В таблице 1 отображено сравнение технологий ASN.1 [19], Google Protobuf [20], Google Flatbuffer [21], Apache Avro [7].

Таблица 1 – Сравнение технологий сериализации

Технология	Наличие IDL	Реализации для языков программирования	Поддержка zero-copy операций	Платформенная независимость
ASN.1	Да	Проприетарные	OER	Да
Google Protobuf	Да	Для многих ЯП	Да	Да
Google FlatBuffers	Да	Для многих ЯП	Нет	Да
Apache Avro	Да	Для многих ЯП	Нет	Да

Из сравнения видно, что только технологии ASN.1 и Google FlatBuffers поддерживают операции без копирования.

Несмотря на то, что ASN.1 является стандартизованным и популярным форматом, поддерживающим операции без копирования при использовании бинарного формата OER, реализации ASN.1 для большинства языков программирования являются проприетарными, что является существенным препятствием для применения этой технологии. На основе сравнения было принято решение использовать технологию Google Flatbuffers, т.к. она удовлетворяет всем предъявленным требованиям.

2.3.6 Технология Google FlatBuffers

Таблицы являются основным способом определения объектов в FlatBuffers. Они состоят из имени и списка полей. Каждое поле имеет имя, тип и, опционально, значение по умолчанию. Если значение по умолчанию не указано

в схеме, оно будет равняться нулю для скалярных типов или значению NONE для остальных. Никакая мета-информация о схеме, например, названия таблиц и полей не появляются в бинарном представлении.

Поля в таблице могут быть обязательными и опциональными. Существенной особенностью формата FlatBuffers является кодирование опциональных полей. Если значение опционального поля совпадает со значением по умолчанию, такое поле не попадает в бинарное представление, что позволяет существенно уменьшить размер сериализованных данных. Например, некоторым сообщениям не нужен обратный локатор, поэтому имеет смысл сделать его поле опциональным. При этом отсутствующий локатор не увеличит размер сообщения по сравнению с тем, если бы поле обратного локатора в принципе отсутствовало в схеме.

Каждое поле в таблице имеет целочисленный идентификатор. Основной целью идентификации полей является поддержка прямой и обратной совместимости схемы данных. По умолчанию идентификатор не указывается, в этом случае все поля в таблице нумеруются последовательно, начиная с нуля. FlatBuffers поддерживает явную нумерации полей, но при условии, что все поля в таблице будут пронумерованы. Нумерация полей накладывает ограничения на изменение схемы данных - никакой идентификатор не может быть использован дважды. Удалить поле из схемы нельзя, однако можно пометить его как устаревшее – deprecated, и перестать записывать в него данные. Если поле являлось опциональным, оно не будет представлено в закодированном представлении.

Google FlatBuffers поддерживает следующие встроенные типы данных:

- Знаковые и беззнаковые целые числа размера 8, 16, 32 и 64 бит.
- Форматы с плавающей точкой размера 32 и 64 бит.
- Массивы произвольных типов.
- Строки в кодировке UTF-8 [22].

Для обеспечения платформенной независимости, целые числа в FlatBuffers всегда кодируются в формате big-endian.

Для представления именованных констант, FlatBuffers поддерживает тип Enum. При этом существует возможность указания размера целочисленного типа, лежащего в основе двоичного представления. Union предоставляет механизм для хранения в поле таблицы одного из нескольких табличных типов. При кодировании Union превращается в идентификатор типа, заданного в Union, и непосредственно данных конкретного типа.

2.4 Модель протокола

При реализации сетевых моделей часто применяют механизм разделения данных по уровням. Протоколы вышестоящих уровней, с помощью механизма инкапсуляции передают свои данные с помощью протоколов нижнего уровня. Модели OSI и TCP/IP являются популярными моделями, применяющими данный принцип [23]. Вместе с этим сообщения сетевых протоколов часто разделяют на две части – заголовок и тело. Заголовок сообщения содержит мета-информацию, необходимую для работы протокола. Тело сообщения содержит данные вышестоящего уровня.

В разделе 2.2.3 рассматривается ситуация, когда сообщение не может быть доставлено на ИС, являющуюся следующим шагом локатора при помощи прямой отправки данных. Поскольку процесс доставки сообщения в общем случае разбивается на два этапа: определение следующей ИС с помощью локатора и маршрутизации сообщения через топологию, в модели взаимодействия ИС предлагается выделить два уровня.

Заголовки нижнего уровня содержат информацию об источнике и приемнике данного сообщения, которые включают в себя идентификаторы исполнительных систем и узлов.

В заголовках верхнего уровня содержится прямой и обратный локаторы, а также уникальный идентификатор данного сообщения. Тело сообщения верхнего уровня содержит данные, специфичные для данного сообщения.

На рисунках 2 и 3 изображены схемы данных сообщений нижнего и верхнего уровней соответственно.

```
table MessageComm {
    message_info: [byte];
    destiny_rank: int32;
    destiny_node: int32;
    source_rank: int32;
    source_node: int32;
}
```

Рисунок 2 – Схема сообщения нижнего уровня на IDL

```
table MessageInfo {
    locator: Locator;
    message: Message;
    backward_locator: Locator;
    uid: uint64;
}
```

Рисунок 3 – Схема сообщения верхнего уровня на IDL

Такая модель протокола имеет определенные преимущества. Нижний уровень скрывает в себе все промежуточные пересылки и сетевую топологию, что позволяет достичь иллюзии полносвязной сети при обмене сообщениями верхнего уровня. Тем не менее, для достижения приемлемых нефункциональных свойств все равно необходимо обеспечивать преимущественно локальные обмены данных между узлами.

2.5 Базовый протокол взаимодействия

В данном подразделе опишем множества сущностей и сообщений, составляющие базовую версию протокола. При необходимости, данные множества могут быть расширены для реализации частных алгоритмов и оптимизаций, применяемых в прикладных задачах.

2.5.1 Представление идентификатора объекта

Для идентификации объектов (ФВ и ФД) введем понятие идентификатора – ID. В качестве ID используется массив целых чисел.

2.5.2 Представление ФД

ФД могут быть представлены как единичными значениями элементарных типов, так и массивами и более сложными структурами. Например, в случае, если ИС работают в одном адресном пространстве, ФД может быть представлен как

некий указатель. Также ФД может представлять идентификатор некоторого ресурса, например, файла.

В качестве способов представления ФД были выделены следующие варианты:

- Представление ФД в качестве массива элементарных типов.
- Представление ФД в качестве идентификатора ресурса в локальном хранилище, доступным нескольким исполнительным системам.

Для единообразного представления в сообщениях, ФД в IDL задан с помощью Union-типа. Такой способ определения не ограничивает множество возможных представлений ФД.

2.5.3 Представление ФВ

В существующей реализации исполнительной системы LuNA каждый ФВ представлен в виде агента, состояние которого определяется номером текущего исполняемого фрагмента кода. Поскольку ФА накладывает на ФВ лишь функциональное ограничение, может существовать множество других реализаций ФВ. Поэтому, аналогично ФД, ФВ представлен в IDL с помощью Union-типа.

2.5.4 Сообщение миграции ФВ

ФВ порождается на одной ИС, но исполняться может на нескольких, например, в результате работы алгоритмов динамической балансировки нагрузки по узлам. Для миграции ФВ введем сообщение типа Migrate, которое включает в себя ФВ и ID.

2.5.5 Сообщение прямой доставки ФД

Для доставки ФД без предварительного запроса введем сообщение Post. Сообщение Post состоит из DF, ФД, а также дополнительной информации. В дополнительной информации может содержаться информация о ФД, необходимая, например, алгоритмам сборки мусора. Рассмотрим следующие примеры:

– Количество потреблений ФД неизвестно, удаление ФД происходит по прямому запросу. В этом случае сообщение не содержит дополнительной информации.
– Количество потреблений ФД известно, тогда дополнительная информация содержит целое число.
– Удаление ФД происходит после потребления определенным ФВ. В этом случае дополнительная информация содержит ID данного ФВ.

2.5.6 Сообщение доставки ФД по запросу

В данном варианте одна из исполнительных систем запрашивает данные, посылая сообщение типа Request, другая исполнительная система отвечает на запрос сообщением Response. Сообщение Request включает ID запрашиваемого ФД. Сообщение Response содержит ФД и его ID. Важным замечанием является то, что прямым локатором для сообщения Response будет являться обратный локатор сообщения Request.

Стоит отметить тот факт, что благодаря доставке сообщений при помощи локаторов, даже в случае миграции запрашивающего ФВ на другую ИС до получения ФД, запрашивающий ФВ все-равно получит ответ.

2.5.7 Сообщение явного удаления ФД

Для явного ФД по прямому запросу введем сообщение типа Destroy. В простейшем случае оно должно содержать лишь ID удаляемого ФД. Заметим, что при необходимости, в сообщение Destroy также можно добавить условие для удаления ФД.

2.5.8 Сообщение Idle

Сообщение Idle необходимо для алгоритма обнаружения завершения работы, описанного в разделе 2.2.5. Сообщение включает в себя передаваемый между исполнительными системами счетчик.

2.6 Анализ разработанного протокола

Предложенная модель протокола позволяет при необходимости добавлять поддержку новых прикладных и системных алгоритмов. Для этого необходимо лишь расширить множество верхнеуровневых сообщений протокола, а также

поддержать их обработку в исполнительных системах. Существенно, что исполнительные системы могут поддерживать не все сообщения протокола, а лишь те, которые используются ими для решения конкретной задачи. При этом протокол не ограничивает масштабируемость системы, поскольку не требует централизованного хранилища информации.

3 Реорганизация исполнительной системы LuNA для работы по разработанному протоколу

3.1 Архитектура сетевого модуля

В этом разделе описана разработанная архитектура сетевого модуля исполнительной системы LuNA. На рисунке 4 изображена UML-диаграмма классов модуля.

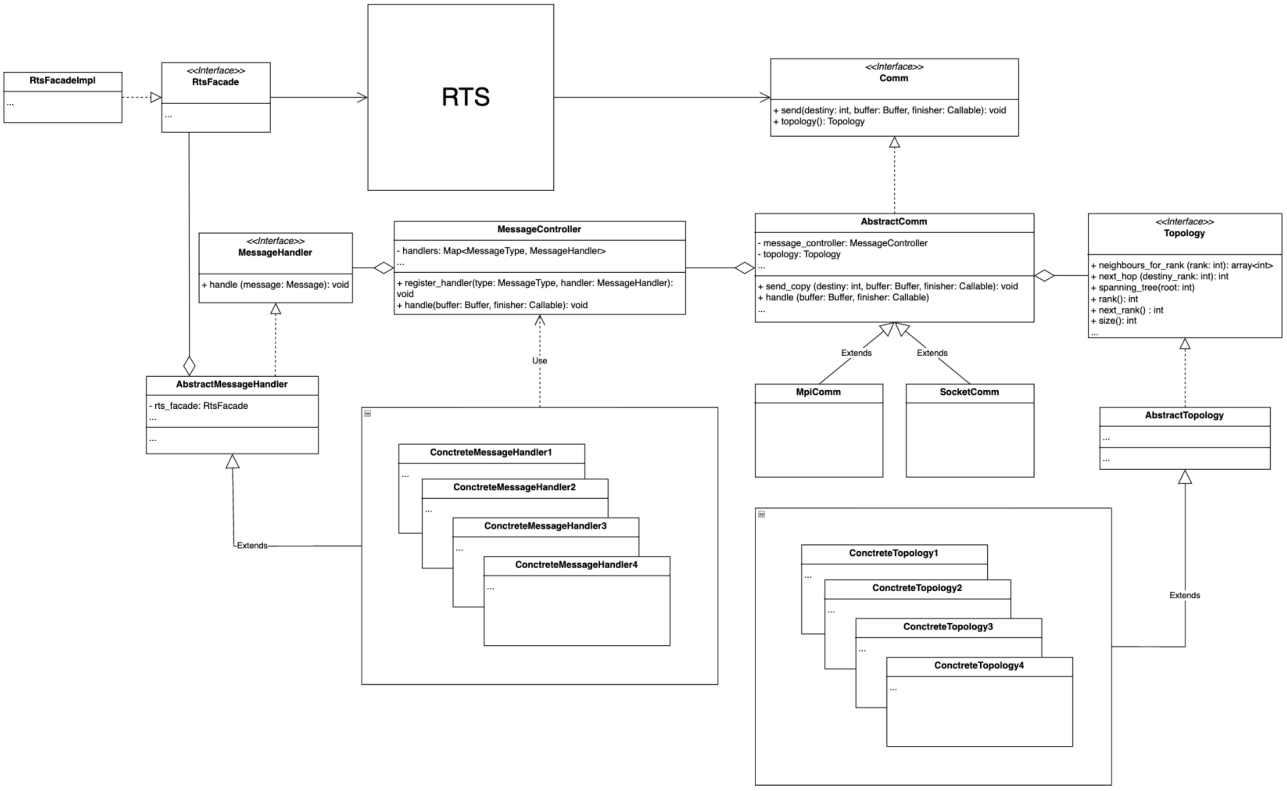

Рисунок 4 – UML-диаграмма классов сетевого модуля

3.1.1 Топология

Topology предоставляет интерфейс для работы с фактической сетевой топологией между исполнительными системами.

Интерфейс содержит следующие методы:

- neighbours_for_rank(rank: int): array<int> – получение идентификаторов исполнительных систем, являющихся соседями ИС с рангом rank
- next_hop(destination: int): int – получение идентификатора ИС, являющейся следующим звеном при пересылке сообщения нижнего уровня
- rank(): int – ранг ИС
- size(): int – количество ИС

Основной реализацией интерфейса Topology является GraphTopology. Класс GraphTopology строит топологию на основе полученного при инициализации неориентированного графа, узлами которого являются исполнительные системы, а ребрами – прямые сетевые соединения между ними.

В случае, если топология сети не меняется, пересчет маршрута с помощью метода next_hop при каждой отправке сообщения коммуникатором добавляет лишние накладные расходы. Чтобы избавиться от лишних накладных расходов, была создана реализация CachedTopologyDecorator, принимающая при создании другой объект интерфейса Topology. При первом вызове next_hop она делегирует работу по вычислению следующего звена, после чего запоминает результат.

3.1.2 Коммуникатор

Коммуникатор (Comm) предоставляет интерфейс для передачи сообщений по сети. Интерфейс коммуникатора содержит следующие методы:

- send(destination, buffer, finisher): void
- topology(): Topology
- finalize(): void

Чтобы послать сообщение верхнего уровня, ИС использует метод send. Параметр destination – следующий идентификатор ИС, вычисленный на основе локационной функции. Параметр buffer – сериализованное представление сообщения верхнего уровня. Контракт коммуникатора не подразумевает его интерпретацию и обработку. Для отправки по сети, коммуникатор помещает сериализованное представление сообщения верхнего уровня в сообщение нижнего. Поскольку коммуникатор может не иметь возможность послать сообщение исполнительной системе с идентификатором destination, коммуникатор осуществляет локальную пересылку своему соседу по топологии. Аналогично коммуникатор поступает при получении сообщения, в котором идентификатор получателя не соответствует идентификатору текущей исполнительной системы.

3.1.3 Реализация коммуникатора на основе MPI

Преимуществом реализации коммуникатора на MPI является возможность использования исполнительными системами механизмов стандарта, например, групповых операций. Существенным недостатком стандарта MPI является отсутствие возможности обработки ошибок. Например, при выходе из строя процесса исполнительной системы или обрыве сетевого соединения, реализации MPI аварийно завершают работу. Также MPI не предоставляет явного механизма настройки взаимодействия MPI-процессов. Из-за этого существует вероятность расхождения топологии MPI-процессов с топологией исполнительных систем. Несмотря на это, система будет функционировать, однако для эффективного взаимодействия необходима согласованность.

При реализации MPI-коммуникатора использовались асинхронные операции типа точка-точка. Для отправки сообщений используется пул потоков, прием сообщений также осуществляется несколькими потоками.

3.1.4 Реализация на основе TCP

TCP является протоколом транспортного уровня сетевой модели OSI/ISO. Протокол TCP предоставляет гарантию надежной доставки и следования сообщений. Основной целью реализации коммуникатора на основе TCP являлась потребность в большем контроле, чем предоставляет MPI. Например, в случае использовании TCP есть возможность самостоятельно определить физическую топологию соединений между коммуникаторами исполнительных систем. Другой возможностью является реализация различных стратегий восстановления после сбоя, например, в случае обрыва соединения или выхода из строя экземпляра ИС. В работе была реализована только стратегия аварийного завершения программы.

Инициализация соединения между исполнительными системами происходит на основе интерфейса Topology. Также стоит заметить, что в данной реализации связь между исполнительными системами поддерживается лишь с помощью одного TCP-соединения, что не позволяет отправлять несколько сообщений параллельно.

3.1.5 Другие возможные реализации коммуникатора

Помимо представленных двух реализаций коммуникатора, существует возможность создать и другие реализации. В качестве транспорта для доставки сообщений могут быть использованы, например, очереди сообщений [24].

3.1.6 Обработчики сообщений

MessageHandler предоставляет интерфейс для обработки сообщений верхнего уровня. Каждая реализация интерфейса MessageHandler обрабатывает лишь один вид сообщений. Чтобы реализации интерфейса MessageHandler могли взаимодействовать лишь с необходимыми компонентами исполнительной системы, был введен интерфейс RtsFacade. Реализации коммуникатора не могут напрямую зависеть от реализаций MessageHandler, поскольку обработчики сообщений транзитивно зависят от компонентов RTS, что нарушает принцип инверсии зависимостей (DIP) Чтобы инвертировать данную зависимость, был создан класс MessageController (контроллер), в котором регистрируются обработчики. Регистрация обработчиков не происходит в коммуникаторе, чтобы соблюсти принципы единственной ответственности (SRP) и разделения интерфейсов (ISP) [25].

Поток данных обработки сообщения устроен следующим образом: коммуникатор получает сообщение верхнего уровня в виде массива байтов, передает этот массив в контроллер, после чего контроллер вызывает необходимый обработчик.

3.1.7 Оценка полученной архитектуры

Архитектура сетевого модуля спроектирована с помощью парадигмы ООП, а также с использованием таких практик, как паттерны проектирования и принципов SOLID. Полученная архитектура позволяет добавлять новые обработчики сообщений, новые способы доставки сообщений, а также менять внутреннюю остальные компоненты ИС без внесения изменений в сетевой модуль.

3.2 Работа с технологией Google Flatbuffers

3.2.1 Генерация схемы данных, компиляция и сборка

Схема данных технологии Flatbuffer описывается с помощью IDL в файлах с расширением .fbs. Чтобы использовать сериализацию и десериализацию данных, Flatbuffer предоставляет утилиту flatc (Flatbuffer compiler). Утилита flatc получает на вход схему данных и генерирует на ее основе код для выбранного языка программирования. В случае использования C++, утилита преобразует схему в набор заголовочных файлов, в котором содержаться функции для работы с сообщениями.

Для генерации кода из схемы, компиляции, получения зависимостей и сборки исполнительной системы LuNA был модифицирован Makefile, используемый системой сборки make. Помимо изменения Makefile, была добавлена сборка системы с помощью утилиты cmake.

3.2.2 Сериализация объектов

Каждый сериализуемый класс описывается в схеме данных, на основе которой генерируется структуры для представления в коде программы. Для единообразной обработки сериализуемых классов был создан шаблонный интерфейс FlatbufferSerializable, изображенный на рисунке 5.

```
template<typename T>
class FlatbufferSerializable {
public:
    using OffsetType = T;

    virtual ~FlatbufferSerializable() = default;

    using BuilderType = flatbuffers::FlatBufferBuilder;

    virtual flatbuffers::Offset<OffsetType> addToBuffer(BuilderType &builder) const = 0;

    virtual void deserializeFromFlatbuffer(const OffsetType *buffer) {
        throw std::runtime_error("deserialize_from_flatbuffer not implemented");
    }
};
```

Рисунок 5 – Интерфейс FlatbufferSerializable

Параметром шаблона является тип данных сериализуемого класса, получаемый при генерации кода. Была реализована сериализация следующих классов:

- class CF : public FlatbufferSerializable<protocol::CF>
- class Locator : public FlatbufferSerializable<void>
- class DF : public FlatbufferSerializable<protocol::DFStraight>
- class Id: public FlatbufferSerializable<protocol::ID>

Сериализация с помощью FlatBuffers состоит из двух этапов. На первом этапе создается экземпляр FlatBufferBuilder. С помощью данного объекта добавляются данные, на основе которых будет построено сообщение. Из-за особенностей представления бинарных данных в виде смещений, для добавления сущности в FlatBufferBuilder сначала необходимо добавить все данные, из которых состоит сущность. Данное свойство определяет процедуру сериализации в методе addToBuffer:

- объект вызывает addToBuffer у всех своих композитов, получая их смещения;
- объект добавляет себя в FlatBufferBuilder на основе смещений из предыдущего шага.

На втором этапе FlatBufferBuilder возвращает буфер с сериализованным представлением сообщения.

3.2.3 Десериализация объектов из полей типа Union

Для использования сущностей, полученных в сообщении, их нужно восстановить из бинарного представления, что требует идентификации типа. В случае использования Union, поле сообщения может содержать данные объектов разных классов. Если при этом количество классов, представляющих данные в Union-типе велико, возникает сложность в единообразной обработке сообщения. Решением данной проблемы является использование единого интерфейса для всех классов сущности, а десериализация класса производится с помощью фабричного метода. В частности, для восстановления локаторов, был создан

класс LocatorFactory. При инициализации процесса ИС в нем регистрируются все классы локаторов.

3.3 Работа с сетевыми модулями

3.3.1 Инициализация сетевых модулей

Перед началом работы исполнительных систем, их сетевые модули должны пройти процедуру инициализации, которая включает в себя установку необходимых соединений. Для инициализации сетевых модулей были написаны скрипты tcp_comm_init.sh и mpi_comm_init.sh, производящие настройку коммуникаторов на основе TCP и MPI соответственно.

Скрипт tcp_comm_init.sh принимает на вход файл, содержащий конфигорационную информацию. В этом файле перечислены адреса и порты коммуникаторов, а также топологию их подключения.

3.3.2 Корректное завершение работы

Когда одна из ИС обнаруживает факт завершения работы системы (в результате выполнения вычислительной задачи или в результате ошибки), она уведомляет об этом другие исполнительные системы с помощью отправки сообщения Stop. При этом может возникнуть ситуация, когда из-за преждевременной остановки ИС может разрушиться топология. Рассмотрим пример с топологией, изображенной на рисунке 6.

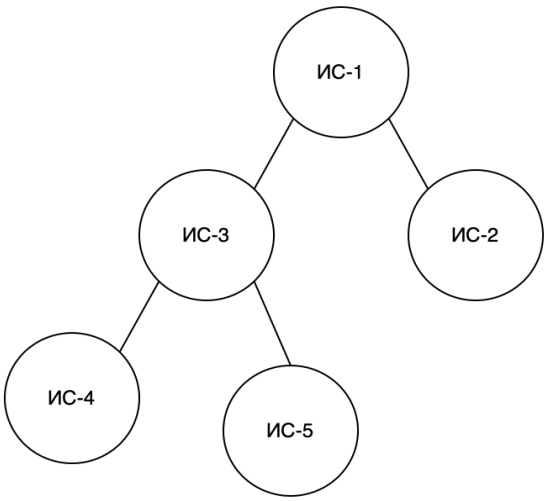

Рисунок 6 – Топология связи исполнительных систем

ИС-1 инициирует процедуру остановки системы, отправляя другим исполнительным системам сообщение Stop. При этом из-за асинхронной отправки, сообщения могут доставляться в любом порядке. Пусть ИС-3 получает сообщение Stop первой и завершает свою работу. Поскольку ИС-3 является точкой сочленения, граф топологии перестает быть связным. Из-за этого ИС-4 и ИС-5 не получат сообщение Stop.

Есть множество способов решения данной проблемы. Например, получив сообщение Stop, ИС может отправить Stop всем своим соседям. В работе была реализована процедура завершения на основе построения остовного дерева и подтверждения остановки.

Без ограничения общности будем считать, что остановку системы начинает ИС-1.

Процедура остановки состоит из нескольких этапов.

- ИС-1 строит остовное дерево из графа топологии. Корнем дерева является ИС-1.
- ИС-1 начинает постфиксный обход дерева в глубину. При посещении вершины, она отправляет ей сообщение Stop. При этом сообщение доставляется с помощью PathLocator (1). PathLocator (1) содержит путь в остовном дереве от корня до посещаемой вершины. Обратным локатором сообщения выступает PathLocator (2), который содержит инвертированный путь PathLocator (1).
- Когда посещаемая вершина получает Stop, она отвечает сообщением Ack. В сообщении Ack указывается uid сообщения Stop. В качестве локатора используется обратный локатор сообщения Stop.
- Получая подтверждение остановки, ИС-1 продолжает обход дерева.
- Процедура завершается после посещения всех вершин.

3.4 Оптимизация доставки сообщений

Для доставки сообщений были добавлены классы ConstantLocator и PathLocator, реализующие интерфейс Locator. ConstantLocator всегда возвращает идентификатор исполнительной системы, переданный объекту класса при его

создании. PathLocator принимает при инициализации путь из идентификаторов исполнительных систем, которые необходимо посетить. Целевой ИС PathLocator является последний идентификатор пути. PathLocator добавлен для реализации процедуры остановки системы.

Упомянутые реализации интерфейса Locator являются статическими в том смысле, что выдаваемый ими идентификатор следующей ИС не зависит от состояния исполнительных систем.

Рассмотрим ситуацию, когда сообщение верхнего уровня доставляется с помощью локатора, учитывающего текущую загруженность узлов. Пусть четыре исполнительные системы связаны топологией, изображенной на рисунке 7.

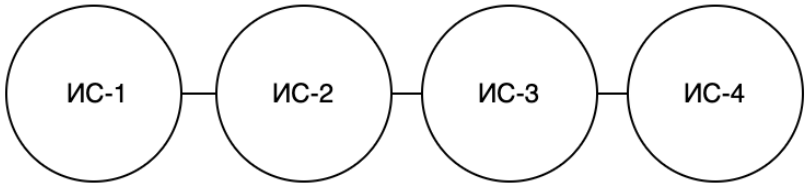

Рисунок 7 – Топология связи исполнительных систем

1) Сообщение порождается на ИС-1, локатор вычисляет идентификатор ИС-4.
2) Поскольку ИС-1 и ИС-4 не имеют прямого соединения, сообщение доставляется через ИС-2 и ИС-3. При этом ИС-2 и ИС-3 используют только заголовки нижне-уровневой части сообщения и не вычисляют значение локатора.
3) Когда сообщение достигает ИС-3, уровень загруженности узлов меняется, и теперь локатор указывает на ИС-2, однако сообщение продолжает доставляться на ИС-4.
4) ИС-4 получает сообщение, но теперь локатор указывает на ИС-2. Поэтому ИС-4 посылает сообщение на ИС-2.

Несмотря на то, что сообщение в конечном счете будет доставлено конечному получателю, возникает несколько лишних пересылок. Этого можно избежать, если выполняются следующие условия:

– ИС-3 поддерживает локатор в пересылаемом сообщении.

– Семантика локатора поддерживает перерасчет пути во время доставки.

В случае выполнения этих условий, ИС-3 может пересчитать значение локатора, после чего отправить сообщение сразу на ИС-2.

3.5 Расширения протокола

В данном разделе описаны дополнительные возможности, расширяющие базовый протокол.

3.5.1 Локальное хранилище ФД

В случае, если исполнительные системы находятся на одном вычислительном узле, вместо прямой передачи некоторого ресурса по сети, например ФД, они могут разместить его в локальном хранилище. Для извлечения ФД из хранилища необходимо передать информацию об его местоположении. В работе было реализовано локальное хранилище на основе механизма разделяемой памяти (shared memory), предоставляемом операционной системой Linux.

Такой подход к передачи ФД имеет несколько преимуществ, особенно если ФД состоит из большого количества данных:

– Поскольку ФД не изменяется после создания, один и тот же ФД могут использовать сразу несколько экземпляров исполнительных систем, экономя при этом потребление памяти.
– Передача ФД через локальное хранилище может существенно снизить количество данных, передаваемых через сеть или с помощью IPC.

Локальное хранилище представлено интерфейсом LocalStorage и реализацией интерфейса SharedMemoryStorage.

Для реализации передачи DF через хранилище на основе разделяемой общей памяти в схему данных была добавлена таблица DF_SHARED_MEMORY.

3.5.2 Обработка неподдерживаемого формата

Чтобы поддержать сценарий обработки исключительной ситуации при получении сообщения, описанного в разделе 2.2.4, было создано сообщение ErrorMessage. ErrorMessage содержит поле Error, имеющее тип Union, в котором передается информация для обработки исключения, а также поле error_uid, в

котором содержится уникальный идентификатор сообщения, обработка которого завершилась неудачей.

Рассмотрим применение данного механизма на примере обмена ФД. Предположим, что необходимо доставить ФД из ИС-1 в ИС-2. Это возможно сделать прямой доставкой с помощью сообщения Post, или доставкой по запросу с помощью пары сообщений Request/Response.

В случае получения ИС-2 неподдерживаемого формата ФД (в сообщениях Post или Response), ИС-2 формирует ErrorMessage с типом ошибки UnsupportedDfType. UnsupportedDfType состоит из ID полученного ФД, списка поддерживаемых форматов ФД и части данных оригинального сообщения. ИС-2 посылает сформированное сообщение с ошибкой по обратному локатору. В общем случае, отправляемое сообщение может получить ИС отличная от ИС-1, например, в случае миграции ФД. Пусть это будет ИС-3.

Когда ИС-3 получает сообщение ErrorMessage с типом ошибки UnsupportedDfType, ИС-3 формирует сообщение Post или Response на основе данных ошибки, помещая в него ФД в запрашиваемом формате.

Процесс обработки неподдерживаемого формата для сообщения Post изображен на рисунке 8.

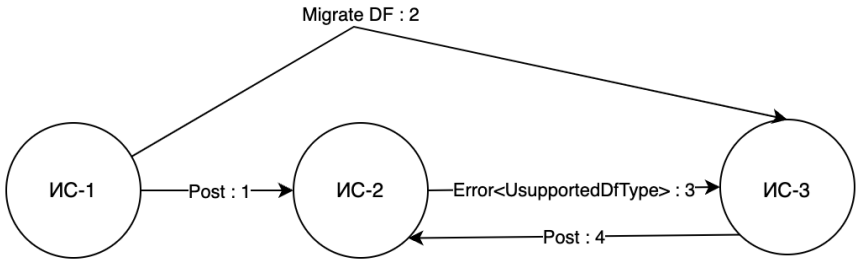

Рисунок 8 – Обработка неподдерживаемого формата ФД для сообщения Post

3.5.3 Балансировка нагрузки

Чтобы поддержать динамическую балансировку нагрузки методом Work Stealing [26], было добавлено сообщение Steal. Сообщение Steal содержит идентификатор ИС, а также именованную константу STEAL_TYPE, допускающую значения STEAL и STEAL_REVOKE.

4 Организация взаимодействия по разработанному протоколу между различными реализациями исполнительных систем LuNA

Рассмотрим задачу, состоящую из вычисления матриц с их последующим перемножением. Инициализация матриц выполняется с помощью исполнительной системы класса А, а перемножение с помощью исполнительной системы Б.

Пусть экземпляры исполнительной системы А и вычисленные ими матрицы пронумерованы от 0 до N. Тогда перемножение должно производится в порядке, эквивалентному последовательному перемножению матриц с номерами 0...N.

На рисунке 9 изображено расположение исполнительных систем.

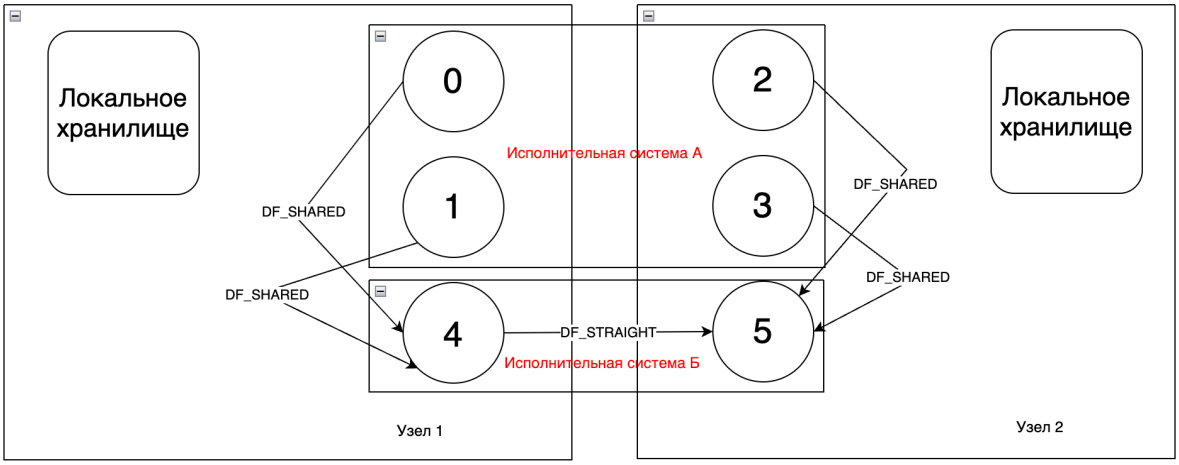

Рисунок 9 – Расположение экземпляров исполнительных систем

В качестве исполнительной системы А выступает реализация, описанная в третьей главе. Исполнительная система Б имеет другую реализацию, которая поддерживает сообщения Post, Idle, Stop, Ack; ФД в форматах DF_STRAIGHT, DF_SHARED; локаторы ConstantLocator, CyclicLocator, PathLocator.

Для эффективной передачи ФД между исполнительными системами в рамках одного узла используется механизм, описанный в разделе 3.5.1.

Исполнение программы происходит следующим образом:

– На экземплярах исполнительных систем 0 – 3 производится вычисление вещественных матриц с сохранением результатов в локальные хранилища.

– ИС-0 и ИС-1 передают вычисленные матрицы с помощью сообщения Post на ИС-4. Т.к. ФД передаются внутри одного узла, ФД передается в формате DF_SHARED с указателем на данные в локальном хранилище. Аналогично происходит передача ФД от экземпляров ИС-2 и ИС-3 до ИС-5.
– ИС-4 производит перемножение матриц, полученных от ИС-0 и ИС-1, после чего отправляет ФД с результатом на ИС-5 с помощью сообщения Post в формате DF_STRAIGHT.
– ИС-5 перемножает матрицы, полученные от ИС-2, ИС-3 и ИС-4.

Стоит заметить, что порядок перемножения матриц зависит от порядка получения данных на ИС-4 и ИС-5. В описании исполнения программы опущена передача сообщений Idle, Stop, Ack. Результаты тестирования отображены в таблице 2.

Таблица 2 – Сравнение переданных по сети данных экземплярами исполнительных систем

Размер матриц	Количество переданных по сети данных (байт)					
	ИС-0	ИС-1	ИС-2	ИС-3	ИС-4	ИС-5
128x128	1472	1244	672	460	131932	816
256x256	1472	1244	672	460	525148	816
512x512	1472	1244	672	460	2098012	816

По результатам тестов видно, что большая часть данных передается с помощью ИС-4. Это объясняется тем, что ИС-4 передает данные в ИС-5 без использования локального хранилища. Кроме того, объем данных, передаваемых другими исполнительными системами, не зависит от размера обрабатываемых матриц, что демонстрирует эффективное взаимодействие исполнительных систем, работающих в рамках одного узла.

ЗАКЛЮЧЕНИЕ

В ходе проведенной работы был осуществлен анализ существующих средств обеспечения взаимодействия параллельных программ, а также выявлены проблемы, возникающие при организации взаимодействия исполнительных систем фрагментированного программирования. Был разработан протокол взаимодействия исполнительных систем, решающие рассмотренные проблемы.

Исполнительная система LuNA была реорганизована для работы по разработанному протоколу. В процессе реорганизации были внесены изменения в архитектуру сетевого модуля.

Проведено тестирование протокола при решении вычислительной задачи с использованием различных реализаций исполнительных систем, а также оптимизации передачи данных через разделяемую память узла.

На защиту выносятся следующие положения:

1) Разработан протокол взаимодействия исполнительных систем фрагментированного программирования.
2) Разработана архитектура сетевого модуля.
3) Протокол реализован в составе системы LuNA.

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».

_____ _____

ФИО студента *Подпись студента*

« ____ »_____20 __г.

(заполняется от руки)

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ И ЛИТЕРАТУРЫ

1. Перепелкин В. А. Система LuNA автоматического конструирования параллельных программ численного моделирования на мультикомпьютерах // Проблемы информатики. – 2020. – №. 1. – С. 66-74.
2. MPI Documents. Официальная документация стандарта MPI [Электронный ресурс]. – URL: https://www.mpi-forum.org/docs/ (дата обращения 17.05.2024).
3. The Charm++ Parallel Programming System. Официальная документация Charm++ [Электронный ресурс]. – URL: https://charm.readthedocs.io/en/latest/charm++/manual.html (дата обращения 17.05.2024).
4. Apache Hadoop. Документация [Электронный ресурс]. – URL: https://cwiki.apache.org/confluence/display/HADOOP2 (дата обращения: 17.05.2024).
5. Dean J., Ghemawat S. MapReduce: simplified data processing on large clusters // Communications of the ACM. – 2008. – Т. 51. – №. 1. – С. 107-113.
6. Remote procedure call [Электронный ресурс]. – URL: https://en.wikipedia.org/wiki/Remote_procedure_call (дата обращения: 17.05.2024).
7. Apache Avro. Документация [Электронный ресурс]. – URL: https://avro.apache.org/docs/ (дата обращения: 17.05.2024).
8. ECMA International. The JSON Data Interchange Format [Электронный ресурс]. – URL: https://www.ecma-international.org/publications-and-standards/standards/ecma-404 (дата обращения: 17.05.2024).
9. W. Carlson, J. Draper, D. Culler [и др.]. – Introduction to UPC and Language Specification // CCS-TR-99-157, IDA Center for Computing Sciences, 1999.
10. Chivers I., Sleightholme J. Coarray Fortran. In: Introduction to Programming with Fortran. — Springer Cham, 2015.

11. Almasi G. PGAS (Partitioned Global Address Space) languages // Encyclopedia of Parallel Computing. – 2011. – С. 1539-1545
12. Перепёлкин, Владислав Александрович. Язык и система LuNA автоматического конструирования параллельных программ численного моделирования на мультикомпьютерах : диссертация на соискание ученой степени кандидата технических наук : 05.13.11 / Перепёлкин, Владислав Александрович ; [Место защиты: ИВМиМГ СО РАН]. – Новосибирск, 2023.
13. Message Passing in Distributed System [Электронный ресурс]. – URL: https://www.geeksforgeeks.org/message-passing-in-distributed-system/ (дата обращения: 17.05.2024).
14. Диффундирующие вычисления [Электронный ресурс]. – URL: https://neerc.ifmo.ru/wiki/index.php?title=Диффундирующие_вычисления (дата обращения: 17.05.2024).
15. Dijkstra E. W., Scholten C. S. Termination detection for diffusing computations //Information Processing Letters. – 1980. – Т. 11. – №. 1. – С. 1-4.
16. Huang S. T. Termination detection by using distributed snapshots //Information Processing Letters. – 1989. – Т. 32. – №. 3. – С. 113-119.
17. IDL (Interface Description Language) [Электронный ресурс]. – URL: https://www.telecomtrainer.com/idl-interface-description-language/ (дата обращения: 17.05.2024).
18. Zhang T. et al. Achieving Zero-copy Serialization for Datacenter RPC // 2023 IEEE International Performance, Computing, and Communications Conference (IPCCC). – IEEE, 2023. – С. 304-312.
19. Dubuisson O. ASN.1 Communication Between Heterogeneous Systems. – Morgan Kaufmann, 2000. – С. 456-457.
20. Protocol Buffers. Документация // Официальный сайт [Электронный ресурс]. – URL: https://protobuf.dev/ (дата обращения: 17.05.2024).

21. FlatBuffers. The Memory Efficient Serialization Library [Электронный ресурс]. – URL: https://flatbuffers.dev/index.html (дата обращения: 17.05.2024).
22. Yergeau F. RFC2279: UTF-8, a transformation format of ISO 10646. – 1998.
23. Таненбаум Э., Уэзеролл Д. Компьютерные сети. 5-е изд – С. 57-64.
24. IBM. Очереди сообщений [Электронный ресурс]. – URL: https://www.ibm.com/topics/message-queues (дата обращения: 17.05.2024).
25. Martin R. C. Agile software development: principles, patterns, and practices. – Prentice Hall PTR, 2003. – С. 95-135.
26. Чмиль А.В. Разработка динамического балансировщика нагрузки для runtime-системы LuNA // Информационные технологии : Материалы 58-й Междунар. науч. студ. конф. 10–13 апреля 2020 г. / Новосиб. гос. ун-т. – Новосибирск : ИПЦ НГУ, 2020. – С. 48.

ПРИЛОЖЕНИЕ А

Схема данных протокола, описанная с помощью IDL технологии Google Flatbuffers.

```
namespace protocol;

table DF_STRAIGHT{
   bytes: [ubyte];
}

table DF_SHARED_MEMORY{
   key: string;
   offset: uint64;
   size: uint64;
   node_id: int;
}

union DF: uint16{
   DF_STRAIGHT = 1,
   DF_SHARED_MEMORY = 2
}

table ID{
   ids: [int32];
}

table StoreItem{
   id: ID;
   df: DF;
}

table CF{
   next_block: int32;
   args: [DF];
   ids: [ID];
   store: [StoreItem];
   caption: string;
}

table DF_WRAPPER{
   df: DF;
}

table CyclicLocator {
 destiny_rank: int;
}

table PathLocator {
   path: [int32];
}
```

```
table ConstantLocator {
   target: int32;
}

union Locator {
 CyclicLocator,
 PathLocator,
 ConstantLocator
}

table Migrate {
 cf: CF;
}

table Post {
   id: ID;
   df: DF;
   count: int32;
}

table RemotePtrResponsePayload {
   remote_pointer: uint64;
}

union ResponsePayload {
   RemotePtrResponsePayload
}

table Response {
   df_id: ID;
   df: DF;
   payload: ResponsePayload;
}

table RemotePtrRequestPayload {
   remote_pointer: uint64;
}

union RequestPayload {
   RemotePtrRequestPayload
}

table Request {
   df_id: ID;
   backward_locator: Locator;
   payload: RequestPayload;
}

table Stop {
   backward_locator: Locator;
}

table IdleMarker {
```

```
   marker: int32;
}

table Ack {
   message_uid: uint64;
}

enum STEAL_TYPE : byte {
   STEAL = 0,
   STEAL_REVOKE
}

table Steal {
   type: STEAL_TYPE;
   source: int32;
}

table Destroy {
   df_id: ID;
}

table Push {
   df_id: ID;
   df: DF;
   cf_id: ID;
}

table IdInfo {
   id: ID;
   numbers: [int32];
}

table SubroutineInvoke {
   subroutine_name: string;
   id_infos: [IdInfo];
}

table UnsupportedDfType {
   id: ID;
   supported_df_types: [uint16];
}

union Error {
   UnsupportedDfType
}

table ErrorMessage {
   error: Error;
   error_uid: uint64;
}

union Message: uint16 {
 Post = 1,
```

```
 Request = 2,
 Response = 3,
 Stop = 5,
 Ack = 6,
 IdleMarker = 7,
 Steal = 8,
 Destroy = 9,
 Push = 10,
 SubroutineInvoke = 11,
 Migrate = 12,
 ErrorMessage = 13
}

table MessageInfo {
   locator: Locator;
   message: Message;
   backward_locator: Locator;
   uid: uint64;
}

table MessageComm {
   message_info: [byte];
   destiny_rank: int32;
   destiny_node: int32;
   source_rank: int32;
   source_node: int32;
}
```

ПРИЛОЖЕНИЕ Б

Сетевой модуль исполнительной системы «LuNA»

Руководство оператора

Листов 6

Новосибирск 2024

СОДЕРЖАНИЕ

Аннотация 49

1 Назначение программы 50

1.1 Функциональное назначение программы 50

1.2 Эксплуатационное назначение программы 50

1.3 Состав функций 50

2 Условия выполнения программы 51

2.1 Минимальный состав аппаратных средств 51

2.2 Требование к персоналу 51

3 Выполнение программы 52

3.1 Загрузка и запуск программы 52

3.2 Выполнение программы 52

3.3 Завершение работы программы 52

АННОТАЦИЯ

В данном документе представлено руководство для оператора по использованию сетевого модуля исполнительной системы LuNA.

В разделе «Назначение программы» содержатся сведения о предназначении программы и информация, необходимая для понимания её функций и эксплуатации.

Раздел «Условия выполнения программы» включает требования, необходимые для её корректного выполнения.

В разделе «Выполнение программы» описана последовательность действий оператора, обеспечивающая загрузку, запуск, выполнение и завершение работы программы.

Оформление программного документа «Руководство оператора» произведено по требованиям ЕСПД: 19.101-77, 19.105-78, ГОСТ 19.505-79.

1 Назначение программы

1.1 Функциональное назначение программы

Сетевой модуль предназначен для обеспечения взаимодействия исполнительных систем LuNA, работающих на различных вычислительных узлах, в соответствии с разработанным протоколом.

1.2 Эксплуатационное назначение программы

Сетевой модуль предназначен для использования разработчиками фрагментированных программ в системе LuNA для решения крупных задач численного моделирования и вычислительной математики.

1.3 Состав функций

Программа обеспечивает возможность выполнения перечисленных ниже функций:

1) Инициализация сетевых соединений между исполнительными системами;
2) Обмен сообщениями между исполнительными системами в рамках исполнения фрагментированной программы;
3) Обнаружение завершения работы системы.

2 Условия выполнения программы

2.1 Минимальный состав аппаратных средств

Программа предназначена для использования на персональном компьютере или вычислительном кластере. Минимальный перечень технических средств, обеспечивающих работу программы:

1) Оперативная память объемом не менее 8 Гб;
2) Жёсткий диск объёмом не менее 128 Гб;
3) Операционная система Linux Ubuntu версии не менее 20.04 LTS.

2.2 Требование к персоналу

Конечный пользователь программы (оператор) должен иметь практические навыки работы с интерфейсом командной строки, а также уметь запускать LuNA-программы.

3 Выполнение программы

3.1 Загрузка и запуск программы

Сетевой модуль запускается одновременно с исполнительной системой LuNA с помощью скрипта, написанного на языке Python. Для настройки сетевых соединений необходимо указать имя конфигурационного файла через параметр --comm-config. В этом файле определяются адреса и порты исполнительных систем, а также топология подключения.

3.2 Выполнение программы

После запуска LuNA-программы исполнительные системы обмениваются сообщениями через сетевой модуль по разработанному протоколу.

3.3 Завершение работы программы

Сетевой модуль определяет факт завершения программы с помощью средств разработанного протокола и уведомляет об этом другие компоненты исполнительной системы.