МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ АГЕНТСТВО ПО ОБРАЗОВАНИЮ
ГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ» (НГУ)

Кафедра параллельных вычислений

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА БАКАЛАВРА

Тальников Андрей Васильевич

Разработка и реализация протокола безопасного обмена сообщениями для масштабируемой исполнительной системы фрагментированного программирования

Направление подготовки 230100.62 ИНФОРМАТИКА И ВЫЧИСЛИТЕЛЬНАЯ ТЕХНИКА

Руководитель

Перепёлкин В.А.
(фамилия , И., О.)

(уч.степень, уч.звание)

……..
(подпись, дата)

Автор

Тальников А.В.
(фамилия , И., О.)

ФИТ, 6204
(факультет, группа)

..……..
(подпись, дата)

Новосибирск, 2010 г.

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ АГЕНТСТВО ПО ОБРАЗОВАНИЮ
ГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ» (НГУ)

Кафедра параллельных вычислений

УТВЕРЖДАЮ

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

…….
(подпись, дата)

ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ БАКАЛАВРА

Студенту Тальникову Андрею Васильевичу.

Направление подготовки 230100.62 ИНФОРМАТИКА И ВЫЧИСЛИТЕЛЬНАЯ ТЕХНИКА

ФАКУЛЬТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ

Тема: Разработка и реализация протокола безопасного обмена сообщениями для масштабируемой исполнительной системы фрагментированного программирования.

Исходные данные (или цель работы): разработка и реализация безопасного асинхронного протокола обмена сообщениями для системы фрагментированного программирования

Структурные части работы: Исследование существующих протоколов доставки сообщений с учетом предъявляемых требований. Разработка протокола и доказательство его корректности. Реализация и внедрение в исполнительную систему фрагментированного программирования RFES4.

Содержание

ВВЕДЕНИЕ

1.1 Проблема

С каждым годом растут размеры и сложность задач, решаемых на суперкомпьютерах, а с ними растут и требования к вычислительным ресурсам, выделяемым на их решение. При этом программировать такие ресурсы, полностью используя их мощность, становится

гораздо сложнее. Общей тенденцией в развитии программных сред являются системы автоматизации программирования, созданные для достаточно узких областей так, чтобы существенно упростить процесс разработки задач и при этом не проиграть в производительности по сравнению с «низкоуровневым» решением.

Одной из таких технологий является технология фрагментированного программирования, разрабатываемая в отделе Математического обеспечения высокопроизводительных вычислительных систем института Вычислительной математики и математической геофизики Сибирского отделения Российской академии наук (МОВВС ИВМиМГ СО РАН) [1][2][3][4]. Проект по разработке и реализации протокола безопасного обмена сообщениями для исполнительной системы проходил в рамках технологии фрагментированного программирования. Фрагментированное программирование является подходом в параллельном программировании, нацеленным на автоматизацию обеспечения определённых динамических свойств параллельных программ, реализующих большие численные модели. Для решения таких задач применяются мультикомпьютеры разных типов: сети компьютеров, кластеры, GRID системы и подобные решения.

Как и в других системах параллельного программирования, фрагментированные программы состоят из меньших частей, динамически размещаемых на узлах распределенного вычислителя и взаимодействующих друг с другом по сети. В фрагментированном программировании такие части называются фрагментами. Задача обеспечения корректного взаимодействия фрагментов между собой – это одна из нетривиальных задач, возникающих при реализации подхода фрагментированного программирования. Эта проблема требует отдельного рассмотрения, которое проводится в данной работе.

1.2 Обзор существующих средств

Рассмотрим существующие средства, обеспечивающие межкомпонентную коммуникацию в условиях миграции этих компонентов между вычислительными узлами. Назовём такие компоненты агентами.

Во время миграции необходимо переместить агента на некоторый вычислительный узел и не потерять возможность коммуникаций других агентов с этим агентом. Для того чтобы отправить агента по сети, необходимо создать его образ, достаточный для восстановления состояния до миграции, отправить образ по сети и на узле-получателе восстановить агента по образу.

Существует три основных типа протоколов миграции, различающихся по способу сохранения коммуникаций между агентами: синхронная миграция, асинхронная миграция и квази-асинхронная миграция [5].

При синхронной миграции все агенты приостанавливаются на время миграции одного из агентов. После того, как агент восстановлен на целевом вычислительном узле, он удаляется со старого узла, а остальные агенты уведомляются о миграции и продолжают своё исполнение.

Асинхронная миграция подразумевает то, что коммуникации между агентами происходят с помощью сообщений. Этот подход позволяет избежать приостановки агентов на время миграции. Агент, начинающий миграцию, переходит в режим накопления полученных сообщений. Как и в других протоколах, его образ отправляется на целевой вычислительный узел, где он восстанавливается. Различие с синхронной миграцией заключается в том, что «старый» агент не удаляется, а остаётся и переотправляет полученные сообщения «новому».

Протокол квази-асинхронной миграции спроектирован таким образом, чтобы избежать двух проблем предыдущих типов миграции: приостановки во время исполнения и образования цепочек пересылки сообщений агенту. Агент уведомляет других агентов о том, что он начинает миграцию, после чего они не отправляют сообщения, адресованные этому агенту, а откладывают их. Несмотря на это, их исполнение продолжается. После того, как агент завершил миграцию, остальные агенты уведомляются об этом, отправляют ему отложенные сообщения и продолжают обычное исполнение.

В зависимости от того, какая сущность понимается под агентом, протоколы бывают уровня процессов операционной системы или программного уровня. Протоколы уровня процессов операционной системы совершенно прозрачны для пользовательских программ, а протоколы программного уровня требуют представления таких программ в специальном виде.

Если в операционной системе коммуникации между процессами осуществляются не с помощью сообщений, а синхронным способом, то возможна реализация только синхронного протокола миграции таких процессов. Чаще всего такие протоколы используют библиотеки создания контрольных точек (checkpoint) процессов и отката к этим точкам для реализации миграции. Одной из самых известных библиотек такого рода является BLCR [6]. Примерами операционных систем, поддерживающих синхронную миграцию процессов, являются LOCUS [7], MOSIX [8] и Sprite [9].

Существуют операционные системы, реализующие коммуникации между процессами с помощью сообщений. Миграции процессов в них могут быть синхронными, как в Accent [10] и V Kernel [11], и асинхронными, как в Demos/MP [12].

Для того чтобы уменьшить время простоя процессов во время синхронной миграции применяются различного рода оптимизации. В Accent страницы памяти, используемые процессом, не копируются во время миграции, а загружаются при

обращении. V Kernel осуществляет миграцию в две стадии. Сначала все страницы, используемые процессом, отправляются на целевой узел без остановки работы процессов. Затем все процессы приостанавливаются и на целевой узел отправляются только изменённые с прошлой отправки страницы.

Примером средства, реализующего миграцию агентов на программном уровне, является ChaRM [5]. Коммуникации между агентами в ChaRM осуществляются сообщениями. Для миграции агентов ChaRM использует квази-асинхронный протокол. Отправка сообщений на каждом узле происходит через C-Manager, который является элементом управления на каждом вычислительном узле. При отправке сообщения агенту указывается только его идентификатор, а реальное положение определяется экземпляром C-Manager. После миграции некоторого агента C-Manager на каждом вычислительном узле обновляет положение агента.

Существует библиотека Concordia [13], позволяющая создавать Java приложения в модели подвижных агентов (mobile agents) и сообщений. Миграция агентов в ней происходит по квази-асинхронному протоколу. В Concordia агенты объеденены в группы. Каждый агент группы обменивается сообщениями с другими агентами группы через объект группы. Во время миграции объект группы откладывает все сообщения агенту. Объект группы сам по себе не мигрирует, поэтому мигрирующие агенты не теряют связи с ним. Коммуникации реализованы с помощью библиотеки RMI и осуществляются через TCP/IP.

Протокол асинхронной миграции агентов представлен в языке программирования Charm++ [14]. Программы на Charm++ представлены в виде агентов, вызывающих методы друг друга через прокси-объекты. Миграция агентов происходит прозрачно для других агентов и используется для динамической балансировки загрузки вычислительных узлов.

1.3 Формулировка задачи

Рассмотрим исполнение фрагментированной программы на мультикомпьютере. Фрагментированная программа состоит из подпрограмм-агентов. Такие подпрограммы распределены по вычислительным узлам мультикомпьютера и исполняются параллельно. Совершив некоторые вычисления, агент может передать вычисленные данные другому агенту, или запросить у него некоторые данные. Каждый агент потребляет определённое количество вычислительных и сетевых ресурсов, а так же памяти. Для утилизации всех ресурсов вычислителя необходимо динамически перераспределять агентов между вычислительными узлами.

Подход фрагментированного программирования нацелен на реализацию ресурсоёмких вычислительных задач. Фрагментированные программы должны максимально эффективно использовать все ресурсы любого вычислителя, то есть обладать рядом свойств – переносимость, масштабируемость, настраиваемость на ресурсы, коммуникации на фоне вычислений и динамизм [Error: Reference source not found].

Была поставлена задача – разработать и реализовать протокол доставки информационных сообщений, обеспечивающих обмен данных между агентами и их миграции между вычислительными узлами в исполнительной системе фрагментированных программ RFES4.

В связи с особенностями фрагментированных программ, к реализуемому протоколу был предъявлен ряд требований.

Асинхронность. Отправитель сообщения при отправке не должен ждать фактической доставки сообщения до получателя. Асинхронные программы могут реализовывать коммуникации на фоне вычислений, то есть полезно использовать время простоя, вызванное сетевыми задержками.

Масштабируемость. Протокол должен сохранять достаточную производительность при росте размера задачи. Данное требование следует из масштабируемости самих фрагментированных программ.

Из масштабируемости вытекает децентрализация, то есть равноправность вычислительных узлов. Иначе, вычислительные узлы, на которые возложена наибольшая нагрузка, будут являться узким местом, особенно сказывающимся на производительности при увеличении размера задачи.

Эффективность. Под эффективностью понимается малая доля потребляемых протоколом ресурсов от ресурсов, потребляемых при исполнении самой задачи. В связи с ресурсоёмкостью фрагментированных программ производительность каждой используемой компоненты чрезвычайно важна.

Данные условия накладывают ограничения на способ доставки сообщений.

Например, доставка сообщений широковещательной рассылкой и доставка с пересылками не удовлетворяют требованию масштабируемости.

Для доставки широковещательной рассылкой отправляемое сообщение высылается на каждый вычислительный узел. Только тот вычислительный узел, на котором находится получатель, обрабатывает сообщение, остальные игнорируют его. При увеличении количества вычислительных узлов, нагрузка на сеть растёт линейно, что приводит к немасштабируемости такого протокола.

При доставке сообщений с пересылками вычислительный узел, принявший сообщение и не обнаруживший в локальном реестре агента-получателя, пересылает

сообщение вычислительному узлу, на котором предположительно находится агент-получатель. С ростом размеров вычислителя, затраты на пересылки могут значительно возрасти, что делает этот подход немасштабируемым.

Протокол синхронной миграции также не удовлетворяет требованию масштабируемости. Так как задача приостанавливается при каждой миграции, а с ростом размера задачи количество миграций возрастает, эффективность протокола ухудшается при увеличении размера задачи. По этой причине в качестве решения проблемы неприменимы такие средства, как Locus, MOSIX, Sprite, Accent и V Kernel.

Асинхронная миграция, представленная Demos/MP и Charm++, не удовлетворяет требованиям из-за доставки сообщений пересылками.

Условию масштабируемости не удовлетворяет и квази-асинхронная миграция, реализованная в ChaRM и Concordia. Количество координирующих сообщений между агентами при миграции одного из них имеет порядок O(n)+O(m), где n – количество агентов в задаче, а m – количество вычислительных узлов. Помимо этого, коммуникации в Concordia осуществляются через TCP/IP, что делает библиотеку Concordia неприменимой для кластеров и GRID.

Таким образом, протокол должен осуществлять доставку сообщений один-к-одному между агентами, мигрирующими между вычислительными узлами, и поддерживать целостность коммуникаций во время миграции агентов.

Фрагментированное программирование – инструмент для реализации больших численных моделей, требующий средство для обеспечения взаимодействий между фрагментами. Существующие решения в этой сфере не удовлетворяют предъявленным требованиям, что составляет научную новизну данной работы. Была поставлена задача: спроектировать, реализовать и внедрить новый протокол в систему исполнения фрагментированных программ RFES4.

2 Модель

2.1 Фрагментированное программирование

Фрагментированная программа состоит из подпрограмм и содержит в себе явный параллелизм программируемого алгоритма. Все вычисления программы делятся на фрагменты вычислений, данные – на фрагменты данных. Благодаря фрагментированной структуре, манипуляции над программой сводятся к манипуляциям над её фрагментами данных и вычислений. Например, динамическая балансировка загрузки процессоров осуществляется перепланированием запусков фрагментов вычислений с более загруженных процессоров на менее загруженные.

В фрагментированной программе также может содержаться дополнительная информация, описывающая её особенности. Такая информация применяется для оптимизации исполнения программы. Сложность фрагментов вычислений, количество памяти, потребляемой во время работы, и другая подобная информация может использоваться при планировании их запуска.

Формально, фрагментированная программа является кортежём <DF, FC, ρ>, описанием данных, вычислений и схемой организации вычислений.

Переменные алгоритма представлены *фрагментами данных*. Каждый фрагмент данных имеет уникальное имя и обозначает ячейку памяти. В отличие от переменных единственного присваивания, фрагменты данных являются переменными множественного присваивания: во время исполнения программы значение фрагмента данных может меняться произвольное количество раз. Фрагменты данных могут иметь сложную структуру, в том числе агрегировать другие фрагменты данных. Все фрагменты данных образуют множество DF.

Фрагменты вычислений изменяют (вычисляют) значения выходных фрагментов данных, преобразуя входные. Каждый фрагмент вычислений исполняется не более одного раза и имеет уникальное имя. Множество всех фрагментов вычислений называется FC.

Над фрагментами вычислений задаётся *отношение частичного порядка* $\rho = \{<c_i, d_i>\}$. Если пара $<c_i, d_i>$ принадлежит ρ, то фрагмент вычислений c_i должна исполниться раньше фрагмента вычислений d_i.

Исполнение фрагментированной программы – это запуск её фрагментов вычислений в порядке, не противоречащем заданному порядку ρ.

2.2 Исполнительная система RFES4

Для исполнения фрагментированных программ используется *исполнительная система* [1], запускающая фрагменты вычислений и распределяющая ресурсы.

RFES4 (Runtime Fragment Executive System 4) – это одна из таких масштабируемых исполнительных систем.

Модель программ для RFES4 является низкоуровневым приближением модели фрагментированных программ, основанной на событийно-ориентированной (event-driven) модели. В терминах этой модели легче проводить различного рода оптимизации исполнения, но сложнее программировать. Поэтому программы для RFES4 должны быть

сгенерированы *транслятором* по фрагментированной программе, записанной в «чистом» виде.

Программа в модели RFES4 является кортежём <A, M, N>. Агенты из множества A, связанные отношением соседства N, обмениваются сообщениями из множества M.

Каждый агент является обработчиком сообщений, пришедших ему от других агентов. Каждое сообщение инициирует запуск агента-получателя для обработки этого сообщения. Во время обработки своего сообщения агент может асинхронно послать сообщения другим агентам. Только агенты, связанные отношением соседства N, могут отправлять сообщения друг другу. После обработки сообщения, агент переходит в спящее состояние, ожидая следующего сообщения.

Агенты содержат в себе информацию, с которой они работают во время обработки сообщений, и которая сохраняется между запусками.

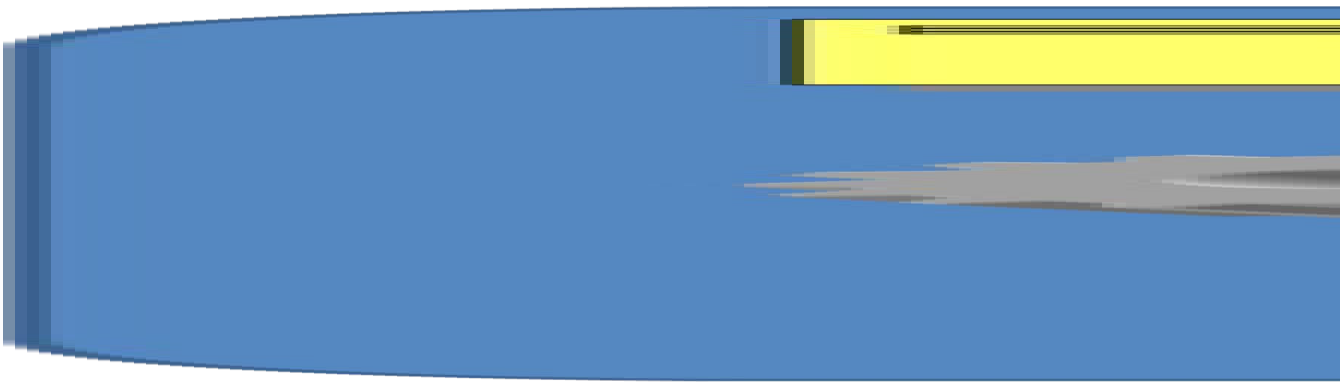

Рис. 1 Агент, обрабатывающий сообщение и спящий агент

По решению RFES4, некоторая группа агентов может быть выбрана для миграции на определённый вычислительный узел.

Для исполнения фрагментированная программа преобразуется к модели исполнительной системы. Группа фрагментов вычислений с одинаковым кодом представляется агентом, инкапсулирующим наиболее используемые этой группой фрагменты данных. Если некоторому фрагменту вычислений для запуска необходим нелокальный фрагмент данных, то агент, содержащий этот фрагмент данных, отправляет требуемое сообщение. Для описания порядка используются сообщения – они явно инициируют запуск агента. После того, как модель программы построена, она оптимизируется и отдаётся на исполнение RFES4.

Задача доставки сообщений между мигрирующими агентами и собственно организации миграции, потребовала реализации протокола.

2.3 Протокол

2.3.1 Структура

Для доставки сообщений была выбрана стратегия поддержки таблиц положений агентов, распределённых по вычислительным узлам, в когерентном состоянии.

В данной главе термин сообщение употребляется в двух смыслах – модельные сообщения между агентами и реальные сетевые сообщения между компьютерами. Сетевые сообщения используются, в частности, для реализации модельных сообщений. Модельные сообщения между агентами будем называть просто сообщениями, а межкомпьютерные сообщения – сетевыми сообщениями.

На каждом вычислительном узле запущен экземпляр *почтовой системы*, отправляющей сетевые сообщения и принимающей их. Каждая почтовая система содержит в себе собственную *таблицу положений* PT. Таблица PT представляет собой отображение идентификаторов агентов в номера узлов, на которых соответствующие агенты находятся в текущий момент, то есть их положение.

В PT содержится информация только о тех агентах, которым может произойти посылка сообщения. Так как посылка сообщений происходит только между соседями, PT состоит из положений соседей агентов рассматриваемого узла.

При отправке сообщения агенту A формируется сетевое сообщение, содержащее, помимо исходного сообщения, идентификатор A, и отправляется на вычислительный узел, соответствующий положению A в PT. При получении почтовой системой сетевого сообщения, содержащего сообщение агенту, агент-получатель извлекается из локального реестра агентов по прикреплённому идентификатору. Агент содержит в себе очередь входящих сообщений, в конец которой ставится полученное сообщение.

Для поддержания когерентности таблиц положений и для того, чтобы сообщения не терялись в условиях миграции получателей, применяется *заморозка* агентов – на время миграции все соседи агента в некотором смысле замораживаются.

Этот протокол является выгодной модификацией квази-асинхронного протокола. За счёт введения понятия соседства между агентами достигается требование масштабируемости, что будет показано в параграфе 2.3.5.

2.3.2 Заморозка

Введём формальное понятие заморозки агента его соседом. Если агент A заморожен своим соседом NA, то

- A может получать сообщения.
- A не может мигрировать.
- Все сообщения от A к NA откладываются.

В один момент агент может быть заморожен несколькими соседями.

2.3.3 Типы сетевых сообщений

В протоколе используется несколько типов сетевых сообщений:

- Сетевые сообщения, реализующие сообщения между агентами.
- Сетевые сообщения, содержащие мигрирующего агента.
- Системные сетевые сообщения, используемые при заморозке:
  - FREEZE
  - FREEZE ACK
  - UNFREEZE
  - UNFREEZE ACK

Сетевые сообщения, реализующие сообщения между агентами – это кортеж <S, R, M>. Они содержат идентификатор агента-получателя R, используемый при разборе очереди сообщений, идентификатор агента-отправителя S и собственно передаваемое сообщение M.

Сетевые сообщения, содержащие мигрирующего агента A, являются кортежём <IA, MQ, NPT>. Во-первых, это данные IA, необходимые для восстановления внутреннего состояния агента A на другом узле. MQ содержит сообщения для A, доставленные в очередь сообщений, но не обработанные им к моменту начала миграции, и NPT – положения соседей A на тот момент. Таким образом, целевой узел вместе с мигрировавшим агентом получает положения агентов, с которыми он, возможно, будет обмениваться сообщениями.

Сетевые сообщения FREEZE и UNFREEZE используются для заморозки и размораживания, а FREEZE ACK и UNFREEZE ACK в качестве ответов на них с подтверждением перехода в соответствующее состояние. Содержание этих сообщений рассматривается в параграфе 2.3.4, описывающем процедуру миграции агента.

2.3.4 Миграция

Рассмотрим миграцию агента.

В первую очередь, всем его соседям отправляются сообщения FREEZE. После получения всех подтверждений FREEZE ACK о переходе соседей в замороженное состояние, агент отправляется на другой узел. Когда целевой узел получает мигрирующего агента, всем его соседям отправляются сообщения UNFREEZE. По получению всех подтверждений UNFREEZE ACK о переходе в размороженное состояние от соседей, агент заканчивает перемещение.

При коллизии сообщений FREEZE, когда два соседа-агента начинают перемещения так, что получат сообщения FREEZE друг от друга, уже ожидая FREEZE ACK, один из агентов продолжит перемещение, а другой прекратит его. Выбор «победителя» производится на основе определённого линейного порядка на множестве фрагментов. Такие ситуации возникают из-за задержек в сети, если во время доставки сообщения FREEZE агент-получатель сам отправляет FREEZE.

Диаграмма состояний агента во время миграции выглядит следующим образом.

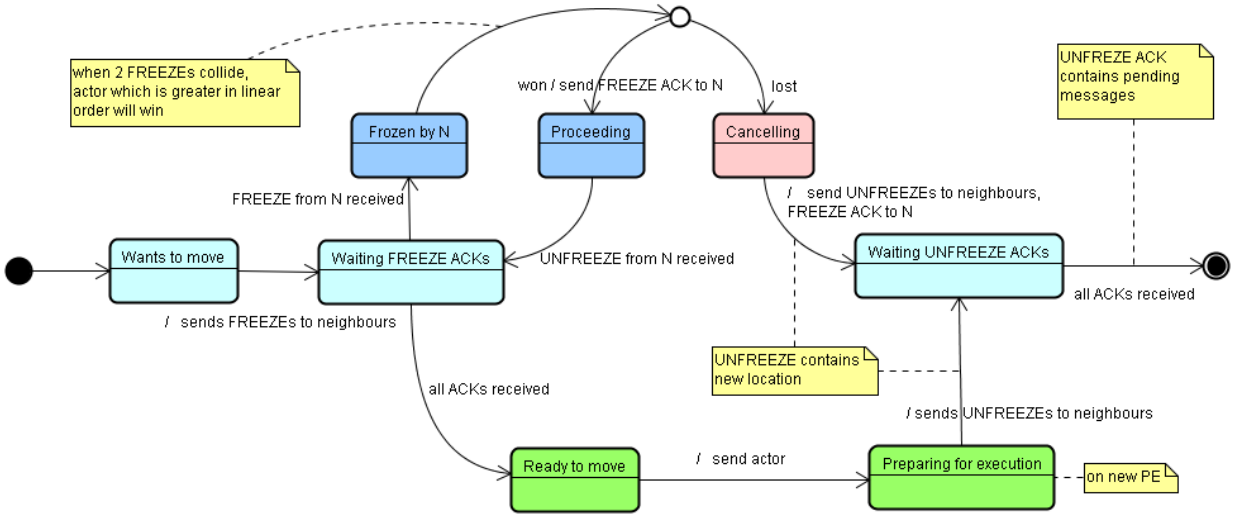

Рис. 2. Диаграмма состояний агента во время миграции.

На рисунке 2 скруглённые прямоугольники – это состояния мигрирующего агента. Выделены два специальных состояния: начальное и конечное, они изображаются закрашенным кругом и обведённым закрашенным кругом соответственно.

Состояния соединены стрелками, являющимися переходами между ними. Рядом со стрелкой может быть написано условие перехода и действия, выполняемые при переходе, в формате [[event][/action]]. В квадратных скобках здесь указаны необязательные элементы, то есть подпись «won / send FREEZE ACK to N» означает, что если агент выиграл, то он посылает FREEZE ACK агенту N и переходит по этой стрелке в следующее состояние, а подпись «/ send actor» означает безусловный переход, сопровождаемый посылкой агента.

Верхняя часть диаграммы отвечает за миграцию в случае коллизии сообщений FREEZE. Оба агента, участвующих в коллизии, сначала подтверждают заморозку с помощью FREEZE ACK. Проигравший агент размораживает всех соседей сообщениями UNFREEZE, а выигравший дожидается такого UNFREEZE и продолжает перемещение. Нижняя часть диаграммы описывает завершение миграции и непосредственно отправку агента.

Следует отметить, что миграция агента не может начаться, пока он находится в замороженном состоянии, или он ещё не разморозил всех соседей после предыдущей миграции.

Диаграмма состояний замораживаемого агента представлена на рисунке 3.

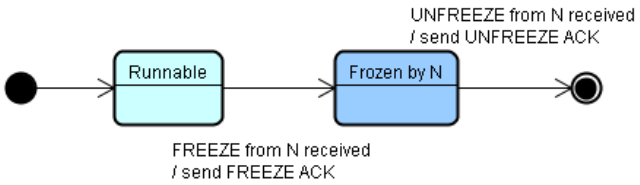

Рис. 3. Диаграмма состояний замораживаемого агента.

Если агент не начал миграцию, то получив сообщение FREEZE от N, он обязан стать замороженным N и отправить подтверждение FREEZE ACK. После получения сообщения UNFREEZE от N агент перестаёт быть замороженным N и отправляет подтверждение UNFREEZE ACK.

Рассмотрим сообщения, используемые в процессе миграции. Во всех таких сообщениях (FREEZE, UNFREEZE, FREEZE ACK, UNFREEZE ACK) содержится идентификатор агента-отправителя S и агента-получателя R. В дальнейшем опустим описание этих элементов.

Сообщения FREEZE и FREEZE ACK – это пары <S, R>. FREEZE показывает намерение S мигрировать, а FREEZE ACK является уведомлением для R, о том, что S получил соответствующий FREEZE.

Сообщение UNFREEZE является кортежём <S, R, P>, где P – новое положение мигрировавшего агента. При получении UNFREEZE положение S в таблице NPT изменяется на P. Сообщение UNFREEZE ACK (подтверждение о получении UNFREEZE) – это кортеж <S, R, DM>, где DM – сообщения от S к R, откладываемые в то время, пока S был заморожен R.

2.3.5 Выполнение требований

Покажем, каким образом данный протокол удовлетворяет поставленным требованиям.

Асинхронность протокола обеспечивается использованием асинхронной отправки сообщений более низкоуровневыми средствами.

Протокол использует понятие соседства, обеспечивающее масштабируемость протокола для регулярных структур данных и вычислений, часто встречающихся в задачах численного моделирования. Для задач численного моделирования характерна локальность использования данных: для обсчёта некоторой области моделируемого пространства, как

правило, требуются данные только из соседних областей. При решении таких задач каждый агент отвечает за обсчёт определённой области пространства и обменивается сообщениями только с агентами, отвечающими за соседние области. В этом случае количество соседей агента не зависит от размеров задачи, поэтому операции протокола остаются неизменными при росте размеров задачи.

Проанализируем ресурсы, потребляемые протоколом, и покажем его эффективность – то, что доля потребляемых протоколом ресурсов мала по сравнению с ресурсами, потребляемыми решаемой задачей.

Рассмотрим характерную программу в RFES4. Программа исполняется на кластере, содержащем около одной тысячи вычислительных узлов. На каждом вычислительном узле находится по несколько агентов, которые за счёт внутренних данных суммарно занимают практически всю пользовательскую память узла.

Зафиксируем порядки ресурсов, потребляемые программой, численно. Пусть в кластере 1000 вычислительных узлов, на каждом из них по 10 агентов. Суммарное количество агентов m=10000. Размер пользовательской памяти в вычислительном узле равен 1 гигабайту, поэтому размер внутренних данных агента – 100 мегабайт. Размер сообщения – 5 мегабайт.

На каждом вычислительном узле хранится таблица PT положений агентов, в худшем случае содержащая по записи с положением на каждого агента. Идентификатор агента имеет размер 8 байт, положение – 4 байта. Порядок размера таблицы PT – 12×m байт=120 килобайт. Доля размера таблицы PT от памяти, потребляемой задачи равна

$$\frac{120\ \text{килобайт}}{1\ \text{гигабайт}} \sim 10^{-4}.$$

К каждому сообщению между агентами прикрепляются идентификаторы отправителя и получателя, то есть 16 байт. Во время миграции агента A, помимо данных, содержащихся в самом агенте, отправляется NPT – часть PT, содержащая соседей A. В худшем случае, все остальные агенты являются его соседями и размер NPT равен 12×(m-1). Сообщения FREEZE и UNFREEZE имеют размер 20 байт, FREEZE ACK – 16 байт, сообщения UNFREEZE ACK, помимо отложенных сообщений, также 16 байт. Доля дополнительной информации в сообщении с мигрирующим агентом равна

$$\frac{120\ \text{килобайт}}{5\ \text{мегабайт}} \sim 3 \times 10^{-2}.$$

Доля заголовка сетевого сообщения от размера тела характерного сообщения составляет

$$\frac{20\ \text{байт}}{5\ \text{мегабайт}} \sim 4 \times 10^{-6}.$$

Сложность алгоритмов протокола имеет порядок поиска значения в хэш-таблице, так как остальные операции имеют фиксированное время исполнения. Доля вычислительных ресурсов, потребляемых протоколом, мала по сравнению с характерной пользовательской программой и количественно показана в главе 3.4 результатами тестирования конкретной реализации протокола.

Таким образом, доля ресурсов, потребляемых протоколом, по сравнению с ресурсами, затрачиваемыми во время исполнения больших численных моделей, чрезвычайно мала.

2.4 Корректность протокола

Докажем корректность протокола, представленного диаграммой состояний на рисунках 2 и 3. Он должен обеспечивать миграцию агентов и доставку сообщений между ними в условиях надёжности узлов и сети.

Всё доказательство корректности разбивается на доказательство более простых утверждений. Для того, чтобы доказать корректность миграции, достаточно доказать, что после миграции агент перейдёт в основное состояние, переместившись на целевой вычислительный узел или оставшись на том узле, с которого начинал миграцию. Для того, чтобы доказать корректность доставки сообщений, необходимо доказать, что отправленные сообщения будут гарантированно доставлены агенту-получателю.

При доказательстве корректности порядок доставки сетевых сообщений является существенным фактором.

Если сетевые сообщения доставляются в произвольном порядке, возможна следующая ситуация. Агент A отправил сообщение M соседу N. В то время, пока соответствующее M сетевое сообщение M’ доставлялось, агент N был выбран к миграции, провёл всю процедуру миграции и переместился на другой вычислительный узел. В тот момент, когда сообщение M’ было доставлено на целевой узел, агента-получателя N на нём уже не было, и сообщение M оказалось утерянным.

Зафиксируем порядок обработки сетевых сообщений в протоколе. Каждое сообщение M типа FREEZE, UNFREEZE, FREEZE ACK или UNFREEZE ACK, то есть сообщение, используемое при заморозке, отправленное с вычислительного узла N1 на узел N2, обрабатывается после того, как все сетевые сообщения, отправленные с N1 на N2 хронологически раньше M, уже обработаны. В частности, сообщения, используемые при заморозке, отправленные с N1 на N2, обрабатываются узлом-получателем в порядке отправки. Назовём сообщения, обладающие таким свойством fence-сообщениями.

В вышеуказанном примере сообщение FREEZE ACK, отправленное агентом A соседу N, необходимо для начала миграции N. Подтверждение FREEZE ACK не будет

обработано до обработки всех предшествующих сообщений, включая M'. Поэтому сообщение M' не будет утеряно.

2.4.1 Корректность миграции

Чтобы доказать корректность миграции, достаточно доказать, что любой агент, начавший миграцию, перейдёт в конечное состояние диаграммы состояний на рисунке 2.

Агент не перейдёт в конечное состояние во время миграции, только если не дождётся события, требуемого для некоторого перехода. Рассмотрим все состояния, в которых для перехода в следующее состояние агент ожидает некоторого события, и покажем, что такое событие реализуется.

Первым таким состоянием является ожидание подтверждений FREEZE ACK от соседей после отправки им сообщений FREEZE. Если не произошло коллизий, то есть каждое сообщение FREEZE было доставлено тогда, когда его агент-получатель не начинал миграцию, то каждый сосед отправит подтверждение FREEZE ACK и мигрирующий агент перейдёт в следующее состояние. В случае коллизии сообщений FREEZE агент либо проигрывает и переходит в следующее состояние, либо выигрывает и начинает ожидать сообщения UNFREEZE от проигравшего агента. Проигравший агент обязан разморозить всех соседей, поэтому мигрирующий агент снова перейдёт в состояние ожидание сообщений FREEZE ACK. Проигравший агент замораживается выигравшим агентом и поэтому не может начинать миграцию, в частности вызывать новую коллизию. Так как количество соседей агента конечно, агент гарантированно перейдёт в следующее состояние.

После завершения миграции агент, отправив сообщения UNFREEZE своим соседям, ожидает подтверждений UNFREEZE ACK от них. Замороженный агент обязан ответить на сообщение UNFREEZE сообщением UNFREEZE ACK, поэтому миграция агента гарантированно заканчивается переходом в основное состояние.

2.4.2 Корректность доставки сообщений

Если агент A замораживает соседа NA, то все сообщения, отправленные от NA к A перед отправкой сообщения FREEZE ACK, будут обработаны A до миграции по свойству fence-сообщений.

Благодаря тому, что агент не может начать новую миграцию, не разморозив своих соседей, то есть не получив от них сообщений UNFREEZE ACK, отложенные сообщения, передаваемые в UNFREEZE ACK, гарантировано будут доставлены.

Покажем актуальность таблиц PT. Будем считать, что на момент запуска задачи таблицы PT содержали актуальные значения. Рассмотрим случаи, когда необходимо обновить значение NP на вычислительном узле N – после миграции агента A, сосед которого NA находится на N и после миграции на узел N некоторого агента. После миграции агента A на узел N таблицу PN необходимо дополнить позициями соседей A.

В первом случае положение мигрировавшего агента обновляется, когда любой из его соседей, находящийся на рассматриваемом вычислительном узле, получает от него сообщение UNFREEZE. Положение P, находящееся в UNFREEZE актуально, так как агент не может мигрировать, не разморозив соседей, а эта процедура ещё не закончена.

Во втором случае положения соседей мигрирующего агента обновляются при получении сетевого сообщения с этим агентом. Соседи мигрирующего агента обязаны быть в замороженном состоянии во время его миграции, поэтому их положения актуальны.

Все сообщения, отправленные агенту до его миграции, во время его миграции и после его миграции будут доставлены, поэтому корректность доставки сообщений доказана.

3 Реализация

Протокол представляет собой двухуровневый модуль, реализованный на языке C++. Первый уровень отвечает за хранение реестра агентов и их состояний. Он предоставляет интерфейс для доступа к реестру агентов, их миграции, а так же для отправки сообщений агентам и вычислительным узлам. Второй уровень используется первым для отправки и получения сетевых сообщений с использованием библиотеки MPI. Он хранит таблицу PT и занимается переупорядочиванием fence-сообщений.

3.1 Fence-сообщения

Отправка сетевых сообщений в протоколе осуществляется посредством MPI, который не гарантирует порядок доставки асинхронных сообщений [16]. Поэтому при реализации протокола потребовалось обеспечивать поддержку fence-сообщений самостоятельно.

3.2 Структура

Протокол представлен четырьмя основными классами: SimpleExecutiveSystem, MailingSystem, MPISender, MPIReceiver. Первым уровнем протокола является SimpleExecutiveSystem, вторым – MailingSystem, MPISender и MPIReceiver. На каждом вычислительном узле запущено по экземпляру каждого из этих четырёх классов, вместе они организуют почтовую систему.

SimpleExecutiveSystem содержит в себе реестр информации об агентах и предоставляет пользовательский интерфейс для инициации миграции, отправки сообщений агентам и обработки полученных сообщений. Информация об агенте FragmentInfo состоит из описания агента FragmentDescription, состояния подтверждений от соседей во время миграции MovingFragmentInfo, состояние самого агента (основное, начал миграцию, заканчивает миграцию), списка отложенных исходящих сообщений, а так же принятых, но ещё не обработанных сообщений.

FragmentDescription содержит идентификатор агента, идентификатор его кода и список соседей. На каждом вычислительном узле существует фабрика, позволяющая по идентификатору кода определить функцию агента, благодаря чему агент при миграции восстанавливается по FragmentDescription.

Все полученные сетевые сообщения обрабатываются SimpleExecutiveSystem. В зависимости от содержания сообщения могу модифицироваться состояния некоторых агентов, изменяться таблица PT и посылаться сетевые сообщения на другие вычислительные узлы, где они будут обработаны аналогичными экземплярами SimpleExecutiveSystem.

MPISender и MPIReceiver являются интерфейсом для посылки и получения сетевых сообщений. Таблица положений PT содержится в объекте MPISender и используется при каждой отправке соответственно сообщения агенту.

MailingSystem – это промежуточное звено между пользовательским и сетевым уровнями, реализующее поддержку fence-сообщений.

Когда через интерфейс SimpleExecutiveSystem отправляется сообщение некоторому агенту, оно передаётся MailingSystem. MailingSystem ведёт нумерацию отправленных сетевых сообщений для каждого вычислительного узла. К посылаемому сообщению дописывается его порядковый номер, тип (обычное сообщение или fence-сообщение) и сформированное сообщение передаётся объекту MPISender для непосредственной отправки.

При получении сетевого сообщения MPIReceiver передаёт его MailingSystem. MailingSystem определяет тип и номер этого сообщения. Если полученное сообщение является обычным сообщением, то оно передаётся SimpleExecutiveSystem на обработку. Если полученное сообщение является fence-сообщением, то оно задерживается до тех пор, пока все сетевые сообщения с меньшими номерами, отправленные с того же узла, не будут переданы на обработку SimpleExecutiveSystem.

Данная реализация многопоточна (thread-safe) и в полной мере реализует протокол, но накладывает некоторые ограничения на программное обеспечение вычислительного узла.

- Для исполнения необходима библиотека MPI.
- От реализации MPI требуется обеспечение уровня потоковой безопасности MPI_THREAD_MULTIPLE.
- Фрагментированные программы, использующие этот протокол, не могут пользоваться MPI напрямую и должны осуществлять обмен сообщениями с помощью протокола.

3.3 Внедрение

В RFES4 протокол обмена сообщениями и миграции агентов является точкой расширения (extension point). Любой протокол, реализующий определённый интерфейс, может быть использован для этих целей. С помощью протокола RFES4 организует запуск агентов для обработки полученных сообщений, передавая его им в качестве средства для коммуникации с другими агентами. Помимо этого, RFES4 выполняет более высокоуровневые операции над агентами, в частности балансировку загрузки

вычислительных узлов [17]. В ходе балансировки некоторые агенты с наиболее загруженных вычислительных узлов мигрируют на менее загруженные.

Класс SimpleExecutiveSystem является расширением, удовлетворяющим предъявленному интерфейсу. Реализованный протокол был внедрён в RFES4, его работоспособность была проверена на тестовых задачах.

3.4 Тестирование

Для тестирования было произведено профилирование протокольных процедур на двух задачах. В первой задаче, реализующей вычисление чисел Фибоначчи, агенты обмениваются большим количеством легковесных сообщений. Во второй задаче, производящей систолическое умножение матриц, обмен сообщениями происходит гораздо реже, размер каждого сообщения при этом больше, а вычисления более трудоёмки.

Характерным параметром является отношение времени исполнения протокольных процедур ко времени исполнения всей программы. Назовём это отношение R.

Программа вычисления чисел Фибоначчи не завершается, поэтому замеренные результаты R имеют приближённый характер.

Ниже, в таблицах 1 и 2, приведены результаты тестирования на четырёхпроцессорной системе Intel® Itanium®2 «smp4x64» (см. Приложение А).

Таблица 1. Отношение времени исполнения протокольных процедур к времени исполнения программы, вычисляющей числа Фибоначчи.

Номер запуска	1	2	3	4	5
R	2.6×10^{-4}	2.4×10^{-4}	2.2×10^{-4}	2.8×10^{-5}	2.7×10^{-5}

Таблица 2. Отношение времени исполнения протокольных процедур к времени исполнения программы, выполняющей систолическое умножение матриц.

Номер запуска	1	2	3	4	5
R	6.3×10^{-5}	3×10^{-5}	2.1×10^{-5}	2.4×10^{-5}	4×10^{-5}

Благодаря тому, что значение R в первой задаче имеет приближённый характер, замеренные значения имеют меньшие отклонения, чем во второй задаче.

То, что значение R во второй задаче на порядок меньше значения R в первой задаче, объясняется тем, что вторая задача более ресурсоёмкая и накладные расходы на процедуры протокола сглаживаются.

Тестирование показало, что доля ресурсов, использованных для обеспечения работы протокола незначительна.

4 Заключение

В работе выполнен обзор средств, обеспечивающих межкомпонентную коммуникацию в условиях миграции этих компонентов между вычислительными узлами, и рассмотрена реализация в них протокола доставки таких сообщений. Реализован протокол доставки сообщений между агентами и обеспечения миграции агентов в исполнительной системе фрагментированного программирования RFES4, и доказана его корректность. Проведено тестирование этого протокола.

На защиту выносятся следующие пункты:

- Разработан протокол обмена сообщениями между агентами и их миграции, удовлетворяющий предъявленным требованиям.
- Доказана корректность разработанного протокола.
- Протокол реализован и внедрён в исполнительную систему RFES4.

Реализованный протокол является законченным модулем, готовым к использованию. Однако, это лишь составная часть более общего проекта по созданию и развитию системы исполнения фрагментированных программ. Работу планируется продолжить в магистратуре по следующим направлениям:

- Реализация реальных задач численного моделирования в системе фрагментированного программирования.
- Разработка и оптимизация системных алгоритмов обмена сообщениями и исполнения фрагментированных программ, реализующих сверхбольшие численные модели.

Литература

1. Malyskin V.E. Nechaev S.P. Tschukin G.A. Kalgin, K.V., Runtime system for parallel execution of fragmented subroutines. Proceedings of the 9th International conference on Parallel Computing Technologies (PaCT-2007), LNCS 4671 (2007), 544-552.
2. Malyshkin V.E. Kraeva, M.A., Assembly technology for parallel realization of numerical models on mimd-multicomputers, the Int. Journal on Future Generation Computer Systems, Elsevier Science, NH 17 (2001), no. 6, 755-765.
3. V.Malyshkin, Assembling of parallel programs for large scale numerical modeling, The Handbook of Research on Scalable Computing Technologies, IGI Global, Kuan-Ching Li and others (Editors) (2010), 295-311.
4. Вальковский, В.А., Малышкин В.Э. Синтез параллельных программ и систем на вычислительных моделях. – Новосибирск: Наука. Сибирское отделение, 1988. – 128 с.
5. Dan, P., Dongsheng, W., Youhui, Z., and Meiming, S. 1999. Quasi-asynchronous migration: a novel migration protocol for PVM tasks. SIGOPS Oper. Syst. Rev. 33, 2 (Apr. 1999), 5-14.
6. BLCR, https://ftg.lbl.gov/CheckpointRestart/CheckpointRestart.shtml
7. Walker, B., Popek, G., English, R., Kline, C., and Thiel, G. 1983. The LOCUS distributed operating system. In Proceedings of the Ninth ACM Symposium on Operating Systems Principles (Bretton Woods, New Hampshire, United States, October 10 - 13, 1983). SOSP '83. ACM, New York, NY, 49-70.
8. MOSIX, http://www.mosix.org
9. Sprite, http://www.eecs.berkeley.edu/Research/Projects/CS/sprite/sprite.html
10. Rashid, R. F. and Robertson, G. G. 1981. Accent: A communication oriented network operating system kernel. In Proceedings of the Eighth ACM Symposium on Operating Systems Principles (Pacific Grove, California, United States, December 14 - 16, 1981). SOSP '81. ACM, New York, NY, 64-75.
11. Cheriton, D. 1988. The V distributed system. Commun. ACM 31, 3 (Mar. 1988), 314-333.
12. Powell, M. L. and Miller, B. P. 1983. Process migration in DEMOS/MP. SIGOPS Oper. Syst. Rev. 17, 5 (Dec. 1983), 110-119.
13. Wong, D., Paciorek, N., Walsh, T., DiCelie, J., Young, M., and Peet, B. 1997. Concordia: An Infrastructure for Collaborating Mobile Agents. In Proceedings of the First international Workshop on Mobile Agents (April 07 - 08, 1997). K. Rothermel and R. Popescu-Zeletin, Eds. Lecture Notes In Computer Science, vol. 1219. Springer-Verlag, London, 86-97.

14. Lawlor, O. S. and Kalé, L. V. 2001. Supporting dynamic parallel object arrays. In Proceedings of the 2001 Joint ACM-ISCOPE Conference on Java Grande (Palo Alto, California, United States). JGI '01. ACM, New York, NY, 21-28.

15. Малышкин, В.Э Параллельное программирование мультикомпьютеров. – Изд-во НГТУ, Новосибирск, 2006. – 296 с.

16. MPI-2.2 Standard, http://www.mpi-forum.org/docs/mpi-2.2/mpi22-report.pdf

17. Grankin V.K., Malyshkin V.E., Perepelkin V.A., Talnikov A.V. Automatic Provision of Dynamic Load Balancing in Fragmented Programs. 12th International Conference on Humans and computers (HC-2009), Aizu, Japan, 2009. 130 – 134.

Приложение А. Описание используемого аппаратного обеспечения

Название	Характеристика
smp4x64	Вычислительный узел HP Integrity rx4640, 4 процессора Itanium2, 1.5 GHz, Кэш-память: L1 16 KB, L2 256 KB, L3 4 MB. Оперативная память: 64 GB.