МИНОБРНАУКИ РОССИИ

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

Факультет информационных технологий

Кафедра параллельных вычислений

Направление подготовки: 230100 ИНФОРМАТИКА И ВЫЧИСЛИТЕЛЬНАЯ ТЕХНИКА

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ БАКАЛАВРСКАЯ РАБОТА

РАЗРАБОТКА И РЕАЛИЗАЦИЯ АЛГОРИТМОВ RUN-TIME ОПТИМИЗАЦИИ ИСПОЛНЕНИЯ ФРАГМЕНТИРОВАННЫХ ПРОГРАММ

ДУПЛИЩЕВ Илья Константинович

«К защите допущена»

Заведующий кафедрой,

д.т.н., проф.

Малышкин В. Э./.....

«.....» ..... 20..... г.

Научный руководитель

м.н.с., ИВМиМГ СО РАН,

Перепёлкин В. А./.....

«.....» ..... 20..... г.

Дата защиты: « 20 » июня 2014 г.

Автор: Дуплищев И. К./.....

Новосибирск, 2014 г.

МИНОБРНАУКИ РОССИИ

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

Факультет информационных технологий
Кафедра параллельных вычислений
Направление подготовки: 230100 ИНФОРМАТИКА И ВЫЧИСЛИТЕЛЬНАЯ ТЕХНИКА

УТВЕРЖДАЮ
Зав. кафедрой Малышкин В. Э.
.....
«.....» ..... 20.....г.

ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ БАКАЛАВРСКУЮ РАБОТУ

Студенту: Дуплищеву Илье Константиновичу
Тема: Разработка и реализация алгоритмов run-time оптимизации исполнения фрагментированных программ
Цель работы: Разработать алгоритм динамической балансировки нагрузки на вычислительные узлы мультикомпьютера для системы фрагментированного программирования
Структурные части работы: Первая часть — обзор существующих решений поставленной проблемы. Вторая часть — разработка и реализация имитационной модели исполнения фрагментированной программы на мультикомпьютере для тестирования реализаций динамическго балансировщика. Третья часть — разработка и реализация алгоритмов динамической балансировки нагрузки на вычислительные узлы мультикомпьютера для системы фрагментированного программирования. Четвёртая часть — тестирование разработанных алгоритмов на реализованной имитационной модели. Получение и анализ результатов. Последняя часть — аппробация полученных результатов на международной научной студенческой конференции и оформление выпускной квалификационной работы

Научный руководитель
м.н.с., ИВМиМГ СО РАН,
Перепёлкин В. А. /.....
«.....» ..... 20..... г.

Задание принял к исполнению
Дуплищев И. К. /.....
«.....» ..... 20..... г.

Содержание

ВВЕДЕНИЕ **5**

Параллельное программирование 5

Фрагментированное программирование 6

Численные алгоритмы 7

Описание проблемы и ее актуальность 7

Предлагаемый подход и его новизна 9

Обзор и анализ родственных работ 11

Mizan: Система динамической балансировки загрузки для крупномасштабных графовых вычислений 11

Статическая балансировка нагрузки для распределенной системы, основанная на кооперативных играх 12

Отображение параллельных программ на многоядерные компьютеры рекуррентными нейронными сетями 12

Диффузионная балансировка 13

Постановка задачи 13

1 Предлагаемое решение **15**

1.1 Имитационная модель 15

1.1.1 Драйвер управления 16

1.1.2 Узел 17

1.1.3 Драйвер связи 19

1.1.4 Генератор задач 19

1.1.5 Профилировщик 20

1.1.6 Балансировщик 20

1.2 Алгоритмы балансировки 21

1.2.1 Адаптивный вариант диффузионной динамической балансировки на основе многослойного персептрона . . . 22

1.2.2 Алгоритм динамической балансировки на основе прогнозирования ESOINN . . . 24

1.2.3 Алгоритм динамической балансировки на основе прогнозирования SOM . . . 26

1.2.4 Алгоритм динамической балансировки на основе прогнозирования многослойным персептроном . . . 27

1.2.5 Гибридный вариант динамической балансировки на основе прогнозирования и адаптивного алгоритма диффузионной балансировки с нейронной сетью . . . 27

1.3 Соответствие представленных алгоритмов к требованиям динамического балансировщика . . . 29

2 Описание реализации и тестирование **30**

2.1 Имитационная модель . . . 30

2.1.1 ESOINN . . . 34

2.2 Тестирование . . . 34

2.2.1 Конфигурация 1 . . . 35

2.2.2 Конфигурация 2 . . . 38

2.3 Анализ полученных результатов . . . 41

Заключение **43**

Список литературы **44**

ВВЕДЕНИЕ

В настоящее время компьютеры широко используются для решения различных задач науки и техники, например, для моделирования различных физических, химических, социальных и других процессов. На сегодняшний день компьютеры позволяют решать задачи со сложностью порядка 2^{64} ($\sim 10^{19}$) вычислительных операций и требующие хранение терабайтов информации, и то при условии использования параллельных вычислений. Но и этого не всегда хватает - на большие вычислительные задачи существующих мощностей недостаточно. При этом результат для конкретной задачи должен быть получен за приемлемое время. К таким задачам, например, относится решение вычислительных задач больших размерностей, моделирование в различных областях: физика (метод частиц в ячейках, моделирование столкновения галактик), биология (моделирование белков и их взаимодействие), химия (моделирование реакций) и т.д. На текущий момент использование одного, пусть даже самого мощного, последовательного компьютера для решения таких задач невозможно.

Параллельное программирование

Один из способов ускорить вычисления - использовать сразу несколько компьютеров, работающих совместно. Такую систему называют распределенным вычислителем, и подход с его использованием носит название параллельных вычислений. Несмотря на кажущуюся простоту решения, такой подход является весьма нетривиальной задачей по проектированию вычислительной техники и разработке алгоритмов.

Для распределённого вычислителя, где два или более компьютера подключены через коммуникационную сеть, выгодно делиться ресурсами для выполнения различных задач. Кроме совместно используемых данных и устройств ввода/вывода, вычислительные узлы могут увеличить производительность системы, совместно используя вычислительные мощности. Для эффективного использования параллельных вычислений на распределённом вычислителе в ряде случаев приходится прибегать к балансировке нагрузки на вычислительные узлы. Балансировка нагрузки на вычислительные узлы является механизмом, который позволяет задачам перемещаться внутри компьютерной сети, что приводит к более быстрому их выполнению и лучшей утилизации сети. Балансировка является одной из самых важных частей параллельной программы, существенно увеличивающая производительность системы и минимизируя простои. [1-17]

Для упрощения реализаций части алгоритмов, использующих параллельное вычисление, была разработана технология фрагментированного программирования, позволяющая скрыть часть проблематики системного параллельного программирования и, в частности, балансировки нагрузки.

Фрагментированное программирование

Технология фрагментированного программирования - это технология программирования, призванная облегчить процесс создания параллельных программ. Эта технология разрабатывается в ИВМиМГ СО РАН. Технология предназначается для реализации на суперкомпьютерах больших задач численного моделирования. Идея технологии заключается в том, что параллельная программа представляется в фрагментированном виде, а именно: переменные и операции прикладного алгоритма агрегируются в фрагменты вычислений и фрагменты данных соответственно, и допустимый порядок исполнения операций определяется информационными зависимостями между фрагментами вычислений. Такая фрагментированная структура программы сохраняется во время её исполнения. Это позволяет автоматически обеспечивать такие динамические свойства программы как динамическая балансировка нагрузки, обеспечение коммуникаций на фоне вычислений, настройка на аппаратные ресурсы. Важной особенностью является то, что детали реализации на конкретном вычислителе отделены от описания прикладного алгоритма (эти части можно назвать соответственно прикладной и системной). Это позволяет, в частности, решать задачу распределения ресурсов не затрагивая непосредственно сам алгоритм [20].

В общем случае для фрагментированной программы существует множество допустимых способов ее исполнения и распределения по ресурсам, но не все варианты будут работать эффективно с точки зрения времени исполнения. Многие варианты окажутся неприемлемыми, и это связано со сложностью отображения программы на ресурсы суперкомпьютера. Фактически выбор эффективного способа исполнения сводится к классической задаче отображения графа программы на граф вычислителя, которая в общем случае является NP-полной.

В связи с этим в фрагментированной программе также может содержаться дополнительная информация, описывающая её особенности. Эта информация нужна для оптимизации исполнения программы и может включать в себя:

- Сложность фрагментов вычислений
- Объем необходимой памяти
- Тип реализуемого алгоритма

Исполнение фрагментированной программы представляет собой запуск её фрагментов в допустимом порядке исполнения.

В частности, особенности технологии фрагментированного программирования (ТФП) позволяют упростить автоматизацию динамической балансировки нагрузки. Это связано с разбиением программы на фрагменты с заданными зависимостями, что позволяет балансировщику заниматься отображением их на вычислительные ресурсы, не вдаваясь в детали алгоритма. Кроме этого, способ исполнения не изменяет работу фрагментированной программы. Это все позволяет в ТФП автоматизировать процесс балансировки нагрузки. Кроме этого, среди численных моделей, для реализации которых нужна ТФП, существенную часть составляют задачи, требующие эффективной работы динамической балансировки.

В настоящей работе будет решаться задача в рамках проекта ТФП.

Численные алгоритмы

Технология фрагментированного программирования предназначена для работы не со всеми прикладными алгоритмами, а только с некоторым классом численных алгоритмов. Это связано с их систематичностью, что позволяет в существенно автоматизировать программирование этих алгоритмов.

В общем случае задача автоматизации динамической балансировки неразрешима, но при использовании систематичности части численных алгоритмов это становится возможным. В данной работе используется это свойство.

Описание проблемы и ее актуальность

Выравнивание нагрузки - процесс перераспределения рабочей нагрузки среди вычислительных узлов распределенной системы для большей утилизации ресурсов и уменьшения времени выполнения задач. Рабочая нагрузка для вычислительных узлов распределённой системы представляет собой набор подзадач, который может исполняться на нескольких вычислителях. Для ТФП этими подзадачами являются фрагменты вычислений. От качества распределения рабочей нагрузки будет зависеть то, насколько эффективно задействованы ресурсы распределённого вычислителя.

Следует различать два вида балансировок по их работе:

- *статическая балансировка* - вид балансировки, в котором отображение программы на вычислительные узлы распределённого вычислителя происходит до этапа запуска программы или как отдельная ступень её исполнения. Это решение не изменяется с ходом работы программы, т.е. если дисбаланс возник, он не устраняется, и заданное изначально распределение на вычислительные узлы сохраняется.

Принятие решения статической балансировки может учитывать все вычислительные узлы, которых может быть достаточно большое количество (порядка 10^6) и рассматривать в целом целевую программу. В общем случае эта задача сводится к вложению графа параллельной программы в граф вычислительной системы и является NP-полной. Ввиду сложности задачи её нецелесообразно решать во время исполнения параллельной программы.

– *динамическая балансировка* в отличие от статической выполняется непосредственно в процессе работы программы. Для большого распределённого вычислителя неприменимо использование централизованных алгоритмов. Это связано с проблемой передачи большого числа информации от узла к узлу, характеризующееся временными задержками и их масштабируемости в зависимости от расстояния передачи. Поэтому алгоритм динамической балансировки для большого распределённого вычислителя обязан работать локально и децентрализованно, с некоторым небольшим количеством узлов для эффективной работы. В противном случае временные задержки могут стать неприемлемыми, а результат работы балансировщика - неэффективным. С уменьшением количества обрабатываемых узлов также уменьшается сложность балансировки, что способствует ускорению работы алгоритма.

Алгоритм динамической балансировки нагрузки для эффективной работы может учитывать следующие характеристики:

– Оценка загрузки
– Сравнение загруженности узлов
– Индексы производительности
– Стабильность системы
– Количество информации, которой будут обмениваться узлы
– Оценки требования к ресурсам задачи
– Выбор задач для передачи
– Выбор узлов, на которые нужно осуществлять передачу
– И прочее.

Сложно недооценить важность балансировки нагрузки для распределённого вычислителя. Конечно, есть возможность не создавать её как компонент системы (в нашем случае это система фрагментированного программирования) и предоставить возможность программисту самому позаботиться об распределении ресурсов. Такой вариант, безусловно, будет работать лучше при условии квалифицированности программиста, чем любой алгоритм сам будет это делать. Самым большим недостатком такого подхода является большой интеллектуальный труд, возложенный на плечи человека, реализующего конкретный алгоритм, которому нет дела до реализации балансировки. Его основной целью является получение результата вычислений. Таким образом, создание отдельной компоненты системы для балансировки программ облегчит реализацию алгоритмов (в нашем случае это вычислительные алгоритмы).

Стоит отметить, несмотря на возможные положительные качества динамической балансировки нагрузки, она, как правило, очень сложна в реализации, что затрудняет на практике её использование, особенно в суперкомпьютерах и параллельных программах соответствующих размеров.

Данная работа посвящена исследованию алгоритмов динамической балансировки нагрузки на вычислительные узлы мультикомпьютера в системе фрагментированного программирования.

Предлагаемый подход и его новизна

Для работы алгоритма динамической балансировки можно использовать статистику работы программы. Это позволяет гибко подбирать критерий возникновения дисбаланса для отдельной программы или целого класса алгоритмов, возможно, теряя точность. Также, используя статистику, становится возможным находить закономерности в работе вычислительной программы. Их можно использовать для прогнозирования динамики модели вычислений непосредственно во время работы, корректируя нагрузку.

Статистика работы программы представляет собой набор данных об исполнении программы. Для использовании этой информации нужно научиться правильно её анализировать. В качестве анализа статистики работы программы предлагается использовать алгоритмы классификации данных и разбор временных последовательностей данных об исполнении программы для поиска закономерностей.

Для задач классификаций, прогнозирования и аппроксимаций часто используют нейронные сети. Основная идея заключается в использовании нейронной сети для поиска скрытых закономерностей в работе вычислительной программы, их выявление и использование. Анализ запуска работы программы нейронной сетью позволит скоррек-

тировать критерий возникновения дисбаланса (способность сети к аппроксимации) и получить инструмент прогнозирования нагрузки.

Алгоритм динамической балансировки нагрузки должен учитывать множество факторов, часть из них:

- Текущее состояние загрузки
- Критерий дисбаланса
- Топологию сети вычислителя

Для увеличения эффективности алгоритма к списку можно добавить следующее:

- Анализ запуска работы программы
- Использование дополнительной информации, которую может предоставить система фрагментированного программирования
- Информация о классе реализованного в программе алгоритма

Новизна работы заключается в использовании нейронных сетей для *динамической* балансировки нагрузки на вычислительные узлы суперкомпьютера для системы фрагментированного программирования. Важными свойствами алгоритма динамической балансировки являются:

- Масштабируемость
- Локальность и учёт соседства
- Децентрализованность
- Учёт динамики модели

Обзор и анализ родственных работ

В обзоре рассмотрены статьи, рассматривающие проблемы балансировки для распределённых вычислителей, использование нейронной сети для балансировки.

Требования, поставленные для алгоритма балансировки:

- Возможность динамической работы балансировщика для системы фрагментированного программирования
- Масштабируемость для использования на суперкомпьютерах, которая включает в себя:

  1. Децентрализованность - алгоритм не имеет единого центра для своего функционирования
  2. Локальность и учёт соседства - коммуникации с соседними узлами

Следует вспомнить, что под динамической работой подразумевается использование небольших ресурсов распределённого вычислителя по сравнению с вычислительными требованиями исполняющейся программы.

Mizan: Система динамической балансировки загрузки для крупномасштабных графовых вычислений

Mizan является модификацией существующей системы Pregel, которая, в свою очередь, была разработана для масштабируемых графовых вычислений. Одним из улучшений новой системы является использование более эффективного алгоритма динамической балансировки нагрузки. Свойствами новой балансировки являются: поддержка различных схем разбиений графа, поддержка мелкозернистой балансировки, слежение за характеристиками всех вершин во время работы и использование распределённых хэш-таблиц для эффективности. Используя замеры, сделанные через определённые шаги, алгоритм конструирует план миграции, который минимизирует затраты на перенос вершин между узлами. Все компоненты Mizan поддерживают распределенное исполнение. [22]

Системы графовых вычислений, основанные на Pregel, используют модель вычислений, называемой Bulk Synchronous Parallel (BSP). Для системы фрагментированного программирования этот подход не подходит и, следовательно, алгоритм, так как он использует особенности BSP.

Статическая балансировка нагрузки для распределенной системы, основанная на кооперативных играх

В статье представлено решение для проблемы статической балансировки нагрузки на распределённый вычислитель с использованием кооперативных игр между вычислительными узлами. Алгоритм основывается на решении Нэша задачи о сделках. [19]

На вход алгоритма подаются параметры скорости обработки узлов и общая скорость прибытия задач на узлы. На выходе получаем распределение нагрузки на узлы.

Для данной работы алгоритм не подходит по причинам централизованности и своей изначальной статической работы. Оценка работы алгоритма в случае использования в динамичном режиме и подачи частичной информации о состоянии вычислителя для децентрализованного исполнения неизвестна.

Отображение параллельных программ на многоядерные компьютеры рекуррентными нейронными сетями

В статье рассмотрена задача отображения графа параллельной программы с взвешенными вершинами (процессами) и рёбрами (межпроцессорными обменами) на произвольный взвешенный граф распределённой вычислительной системы. Предложен алгоритм решения этой задачи, основанный на использовании сетей Хопфилда. Алгоритм протестирован на частном случае рассмотренной задачи — отображении ряда графов параллельных программ (однородное множество независимых заданий — пустой граф, линейка, кольцо, решётка, неоднородная решётка) на многоядерный компьютер. Эксперименты показали, что для графов параллельных программ с несколькими десятками вершин предложенный алгоритм позволяет получить хорошо сбалансированные субоптимальные отображения: величина относительного дисбаланса нагрузки на ядро не превышает 5%; подграф графа программы, отображённый на отдельное ядро, имеет близкое к минимуму количество внешних рёбер, соответствующих межъядерным обменам. [23]

Предложенный алгоритм хорошо себя показал в работе статической балансировки. Адаптировать его работу для системы фрагментированного программирования не является проблемой, но для использования в динамической балансировке с поставленными требованиями он не подходит, так как множество фрагментов во фрагментированной программе является динамическим и невозможно изначально его полностью

определить. Из-за этого фактора решение, предлагаемое сетью, в различные моменты исполнения программы может значительно изменяться, тем самым вызывая большое количество передач между вычислительными узлами. Кроме этого следует отметить централизованность алгоритма, что не также подходит по требованиям к динамическому балансировщику.

Диффузионная балансировка

Диффузионная балансировка является известным локальным алгоритмом балансировки. Основным параметром, использующимся в алгоритме, является порог дисбаланса. Этот параметр выставляется либо аналитиком, либо специальным алгоритмом. Суть диффузионной балансировки сводится к локальному выявлению вычислительных узлов, чья нагрузка превосходит заданный порог дисбаланса. Дальнейшие действия предполагают смещение нагрузки на более простаивающие узлы.

Алгоритм применим для использования в динамической балансировке нагрузки на вычислительные узлы распределённого вычислителя для системы фрагментированного программирования. Недостатки:

- Отсутствие адаптации к задаче
- Необходимость ручного определения порога дисбаланса или составления специального алгоритма по его вычислению.

Диффузионная балансировка используется как эталонный алгоритм для сравнения реализаций балансировщиков так как на текущий момент реализован в системе фрагментированного программирования.

Постановка задачи

Требуется разработать алгоритм динамической балансировки на вычислительные узлы распределённого вычислителя для системы фрагментированного программирования с необходимыми свойствами:

- Децентрализованность
- Масштабируемость

– Локальность коммуникаций

– Адаптация к задаче

Алгоритм должен состоять из двух составляющих: компоненту, использующую одну или несколько нейронных сетей, и компоненту, обрабатывающую результат работы одной или нескольких нейронных сетей и сформировывающую итоговый результат работы балансировщика, понятный системе. Алгоритм динамической балансировки должен уметь адаптироваться к задаче, выявлять дисбаланс и устранять его. Кроме этого, его результат должен быть лучше, чем результат работы диффузионной балансировки.

1 Предлагаемое решение

Для тестирования различных алгоритмов балансировки была разработана и реализована имитационная модель исполняющейся фрагментированной программы. Необходимость в имитационной модели обуславливается сложностью интеграции множества реализаций балансировщика в реальную систему и запуск на распределённом вычислителе. Адекватность реализованной имитационной модели состоит в теоретической схожести работы с системой фрагментированного программирования и подтверждается практическим сравнительным тестированием с реальной программой.

1.1 Имитационная модель

Имитационная модель создаёт множество моделируемых вычислительных узлов, настраивает коммуникации между ними, генерирует множество вычислительных задач и собирает статистику по работе программы. Моделирование исполнения фрагментированной программы происходит последовательно на одном вычислителе.

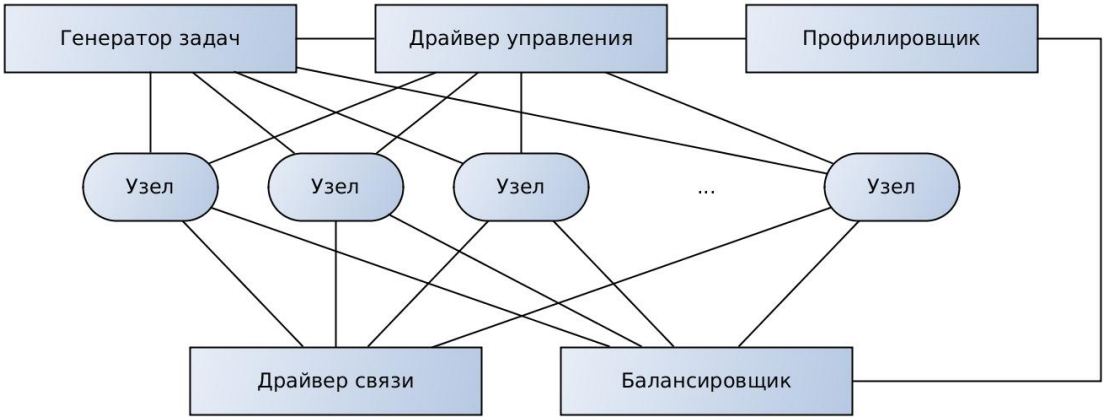

Рис. 1: Схема имитационной модели

На рисунке 2.1 показаны следующие элементы:

- *Драйвер управления* - программа, отвечающая за инициализацию и запуск узлов.
- *Узел* - модуль, выполняющий роль рабочего узла в системе фрагментированного программирования.
- *Драйвер связи* - модуль, отвечающий за коммуникации между узлами. Также обеспечивает необходимую топологию моделируемой системы.
- *Генератор задач* - модуль, отвечающий за генерацию вычислительных задач для узлов.

– *Профилировщик* - компонент системы, отвечающий за сбор данных работы узлов (создания профиля работы) и анализ профиля для балансировщика.

– *Балансировщик* - компонент системы, встраиваемый в узлы и занимающийся балансировкой нагрузки.

Далее будет более подробно описаны составляющие системы моделирования.

1.1.1 Драйвер управления

Драйвер управления изначально является ядром системы моделирования. В нём сосредоточены такие элементы как конфигурация исполнения системы, инициализация ее элементов и функция сериализации для балансировщиков. Этот компонент содержит шкалу модельного времени и проецирует исполнение фрагментированной программы в мультикомпьютере на неё.
Конфигурация системы включает в себя следующие параметры:

– необходимость записи и/или чтения файла профиля

– выбор балансировщика

– объем исполнительных задач, которые может взять на себя узел

– порог диффузионной балансировки

– задержка пополнения исполнительных задач для узла

– коэффициент задержки передачи данных через узел

– модельное время работы различных балансировщиков

– количество задач, которое будет передаваться балансировщиком между узлами

– модельное время задержки получения информации о соседях для узла

– параметры генерируемых задач

В штатном режиме драйвер управления постоянно опрашивает узлы и получает информацию о ближайшем временном событии. После полного обхода он выбирает ближайшее и передаёт управление соответствующему узлу. Временные задержки при его работе не учитываются системой.

Обработка профиля требуется далеко не при каждом запуске как модели, так и реальной системы из-за затрат по времени и ресурсам. В общем, эта процедура необходима для корректировки работы балансировщика с использованием нейронных сетей, т.к. профиль даёт необходимую выборку данных, тем самым позволяя алгоритму *адаптироваться* под конкретную задачу.
Анализ полученных данных играет важную роль для обучения сети, этой частью занимается профилировщик.
Часть параметров системы моделирования необходима для физического соответствия с реальным мультикомпьютером. При их неправильном выставлении можно получить результат, не соответствующий с действительностью. Практически любой алгоритм балансировки может работать эффективно, что неприемлемо. Поэтому подбор параметров происходил в сравнении с работающим мультикомпьютером.

1.1.2 Узел

Узел - часть программы, отвечающая за моделирование узла мультикомпьютера, включающего процессор, сетевой интерфейс и свою память. Узел является самостоятельной и обособленной частью системы, алгоритмы, работающие внутри неё, обязаны быть децентрализованными и работать локально.

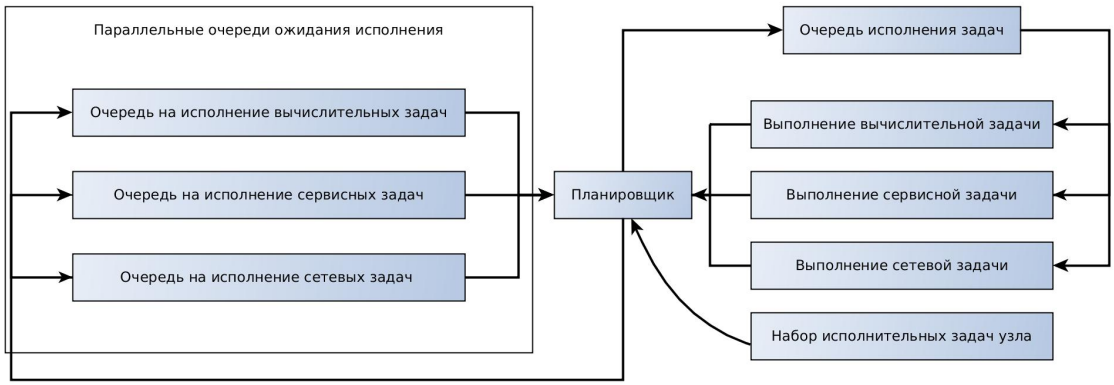

Рис. 2: Обобщённое внутреннее представление узла

На данном уровне детализации можно обособить следующие компоненты:

- *Параллельные очереди ожидания исполнения*
- *Планировщик*
- *Очередь исполнения задач*
- *Выполнение различных типов задач*

– *Набор исполнительных задач узла*

Рассмотрим более подробно каждую часть.

Параллельные очереди ожидания исполнения

Каждое ядро работает со следующими типами задач:

– *Вычислительная задача* - аналогия реальных вычислений. Более подробно описана в пункте «Генератор задач».

– *Сервисная задача* - выполнение операций, необходимых для корректного исполнения в узле. Включает в себя следующие действия:

  1. Пополнение набора исполнительных задач узла
  2. Запуск локальной копии балансировщика
  3. Обновление информации загрузки соседних узлов (необходимая информация для балансировщика)

– *Сетевая задача* - отправка и приём вычислительных задач

Все описанные очереди работают независимо друг от друга, но выполнение внутри них осуществляется последовательно. Хотя это возможно регулировать.

Каждая очередь содержит в себе временные отметки начала исполнения содержащихся задач. Если текущее модельное время совпадает с началом одной из них, то эта задача извлекается и передаётся в очередь исполняющихся задач.

Планировщик

При передаче управления узлу начинает выполнятся *планировщик.* В его обязанности входит поиск и обработку задачи, чьё время исполнения соответствует текущему модельному времени, генерация сервисных задач, добавление в очередь исполнения вычислительных задач.

Очередь исполнения задач

Очередь исполнения задач содержит в себе задачи всех типов, время начала исполнения которых уже прошло, и они ожидают своего времени завершения. Данный компонент эмулирует выполнение задачи соответствующего типа узлом распределённого вычислителя.

Выполнение различных типов задач

После того, как текущий момент модельного времени равен времени завершения одной из задач, она извлекается из очереди исполнения и передаётся в специальный модуль программы, отвечающий за результат выполнения задачи. Например, если пришло время исполнения задачи передачи данных, выполняется непосредственно сама передача.

Набор исполнительных задач узла

Набор исполнительных задач узла содержит в себе очередь из исполнительных задач, которые содержит в себе вычислительный узел. Эти задачи ожидают момента, когда планировщик поставит их на выполнение, или балансировщик примет решение о передачи на соседние вычислительные узлы. Данный компонент программы соответствует множеству фрагментов вычислений для системы фрагментированного программирования, находящихся на одном узле распределённого вычислителя.

1.1.3 Драйвер связи

Драйвер связи является частью системы моделирования, отвечающий за коммуникацию внутри узлов. В зависимости от конфигурации этой части программы будет реализована различная топология связи для смоделированного мультикомпьютера.

Для достижения соответствия работы с реальными системами используются заданные заранее временные задержки на передачу сообщения.

1.1.4 Генератор задач

Генератор задач - часть системы, отвечающий за генерацию вычислительных задач для узлов. Вычислительная задача является объектом с параметром времени исполнения и возможностью задания информационных зависимостей. Для узла задача является атомарным объектом, она не может быть разделена (распараллелена) и представляет собой последовательную программу из реального мира.

Исполнительная задача и необходимые для нее данные считаются одним объектом.

1.1.5 Профилировщик

Профилировщик является частью системы моделирования, отвечающий за *сбор*, *хранение* и *анализ* статистики работы программы.

Сбор данных производится после каждого изменения состояния нагруженности узлов. Собранные данные записываются в специальный файл после завершения моделирования. При следующем запуске системы, если был задан параметр использования профиля, выполняется его чтение и анализ. Результатом является необходимый набор обучающих выборок для разных нейронных сетей, используемых в балансировщиках.

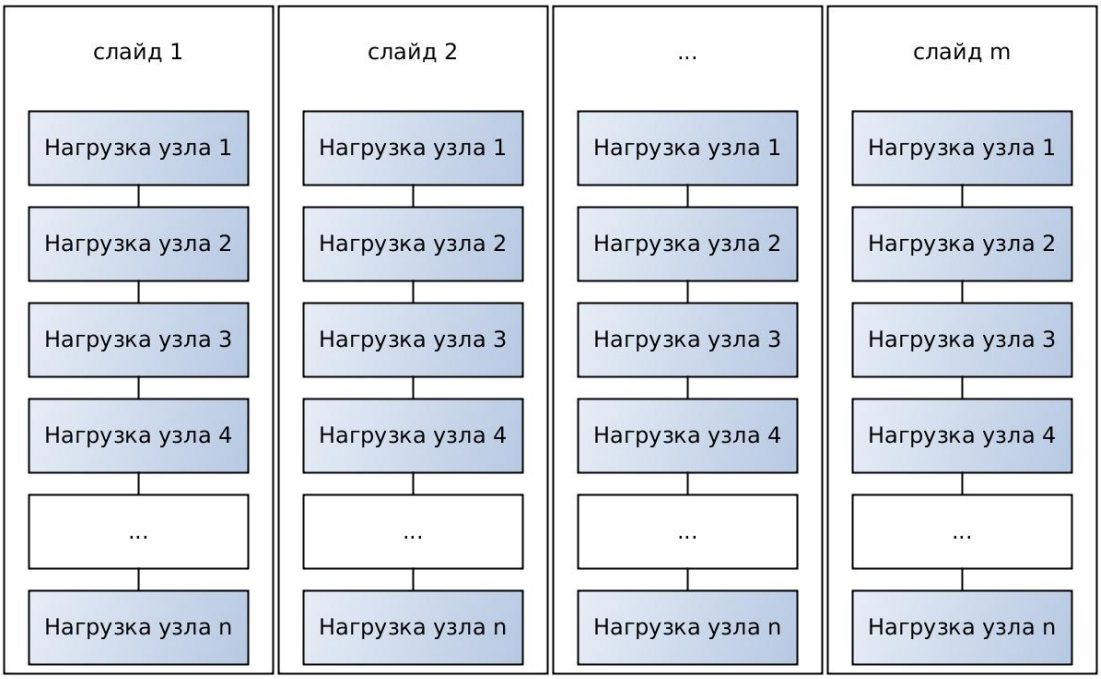

Рис. 3: Структура файла профиля

На рисунке 2.3 представлена структура файла профиля, который записывается при завершении работы программы. Слайдом является *атомарное* состояние нагруженности системы, т.е. изображена полная картина нагруженности мультикомпьютера.

1.1.6 Балансировщик

Балансировщик - компонент имитационной модели, содержащий в себе интерфейс для работы с внешне подключаемым балансировщиком. Балансировщик содержится в каждом узле и его исполнение входит в класс сервисных задач. Данные для работы передаются в формате нагруженности пяти узлов: центральный (текущий), два соседа слева и справа. Полученная информация содержит в себе направление для передачи задач, если он решил, что необходимо разгрузить данный узел, или отсутствие направления, в случае если дисбаланса нет, или он не прогнозируется, либо соседние узлы не могут принять вычислительные задачи в силу своей нагруженности.

1.2 Алгоритмы балансировки

Для балансировки применялись различные алгоритмы с целью проверки их работы для данной задачи. Сравнение работы производилось с эталонным вариантом диффузионной балансировки, используемой на текущий момент системой фрагментированного программирования. Основой представленных алгоритмов являются сети:

- *Многослойный персептрон* - относительно старая нейронная сеть, которая используется для решения задач прогнозирования и аппроксимации. Выбрана так как является хорошо изученной
- *Модифицированный алгоритм самоорганизующейся инкрементальной нейронной сети* - относительно новый алгоритм кластеризующей нейронной сети для обучения без учителя и конечной цели обучения. Не требует априорных знаний о количестве или свойствах искомых классов [24]
- *Самоорганизующиеся карты Кохонена* - нейронная сеть с обучением без учителя и выполняющая задачу кластеризации. Выбрана для сравнения с модифицированной самоорганизующейся инкрементальной нейронной сетью.

Сами алгоритмы работы разделены по принципу работы:

- Самокорректирующий вариант работы по аналогии с диффузионной балансировкой
- Прогнозирование нагрузки на узлы по результатам ранее запущенной программы
- Гибридный вариант

Следует отметить, что для работы сетей использовалась информация о работе программы в предыдущие запуски, что делает алгоритм балансировки зависимым от конкретной программы в общем случае и от класса алгоритма, реализуемым во фрагментируемой программе в частном случае. Объяснение описано в подсекции «Введение. Численные алгоритмы» . Информацию о запуске программы будем называть профилем программы, он содержит в себе данные по нагруженности всех узлов по времени исполнения программы. Имитационная модель позволяет с высокой точностью собирать профиль программы. Реальная система фрагментированного программирования также способна на такое.

1.2.1 Адаптивный вариант диффузионной динамической балансировки на основе многослойного персептрона

Данный алгоритм динамической балансировки содержит в себе алгоритм диффузионной балансировки, выполняющийся нейронной сетью класса «многослойный персептрон». Содержание диффузионной балансировки обуславливается начальной обучающей выборкой, которая сгенерирована с использованием диффузионного алгоритма, но отличается от него тем, что при последующих запусках нейронная сеть дообучается на результатах работы программы, адаптируясь к её специфике.

Важным свойством данной нейронной сети для предложенного алгоритма является обучение с учителем, т.е. сеть будет пытаться найти скрытые зависимости, необходимые балансировщику.

Описание сети

Многослойный персептрон - нейронная сеть, состоящая из нейронов, расположенных на разных уровнях. Помимо входного и выходного слоёв, имеется ещё, как минимум, один внутренний, т.е. скрытый слой.

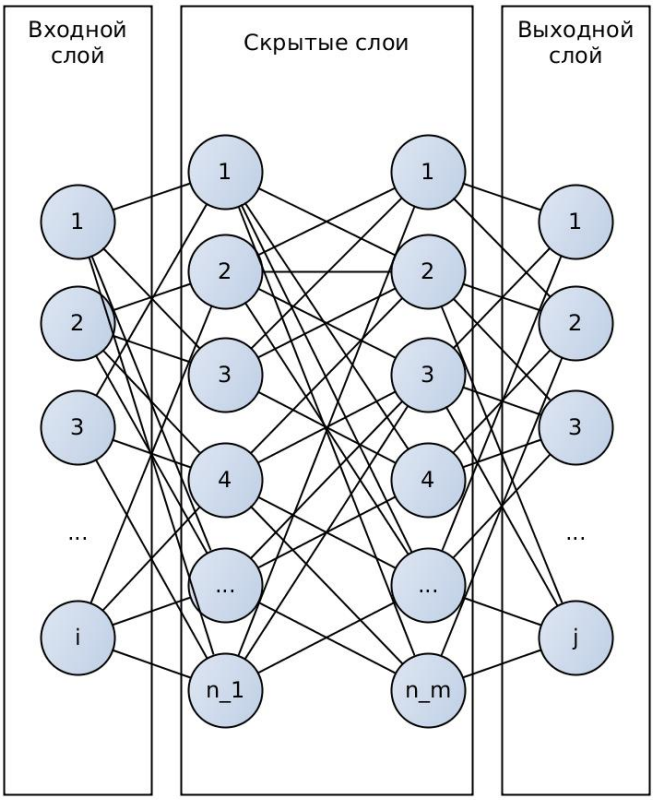

Рис. 4: Схема многослойного персептрона

Работа в балансировщике

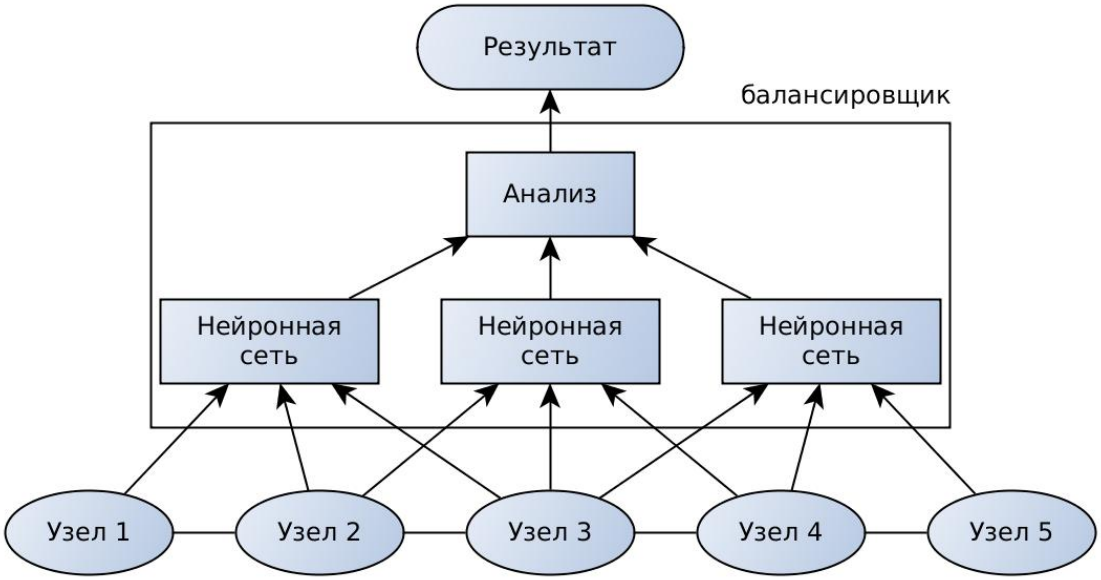

Рис. 5: Общая схема работы балансировщика

Нейронная сеть обучается работать с тремя узлами, определяя возможную ситуацию с возникновением дисбаланса. На вход подаётся состояние нагрузки, на выход - ожидание дисбаланса, одним из следующих значений:

1. Недогруженность узла
2. Промежуточное состояние
3. Перегруженность узла

Результирующее значение является относительным от заданного порога дисбаланса при первом использовании, последующие запуски корректируются по профилю программы.

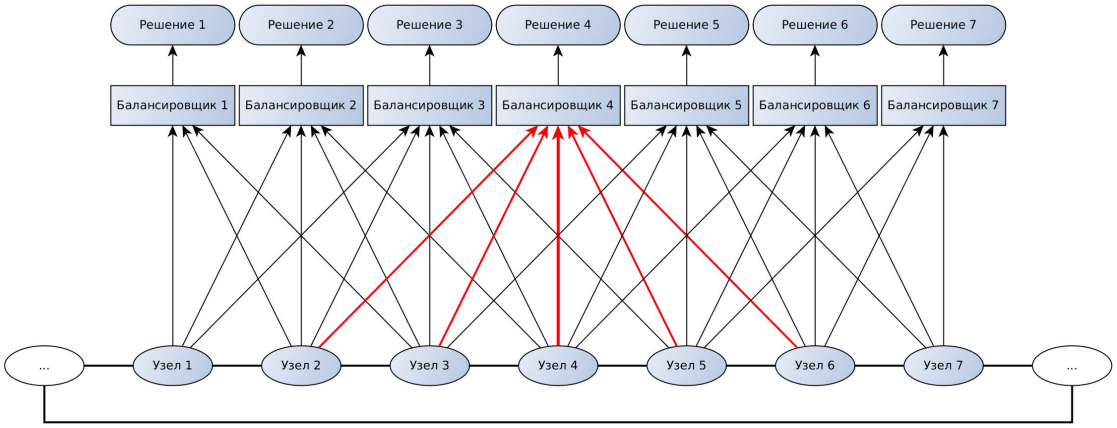

Рис. 6: Представление работы балансировщика для всей системы

1.2.2 Алгоритм динамической балансировки на основе прогнозирования ESOINN

Описание ESOINN

Enhanced Self-Organizing Incremental Neural Network - улучшенный алгоритм самоорганизующейся инкрементальной нейронной сети был создан для обучения без учителя и без конечной цели обучения в режиме эксплуатации. Как видно из названия, сам алгоритм является улучшенной версией своего предшественника и имеет общую с ним концепцию, но лишён ряда недостатков. Оба алгоритма являются кластеризаторами.

Основная идея алгоритма заключается в построении вероятностной модели данных на основе предоставляемых системе образов. В процессе обучения алгоритм строит граф, каждая вершина которого лежит в области локального максимума плотности вероятности, ребра соединяют вершины, относящиеся к одним и тем же классам. В итоге классы образуют области высокой плотности вероятности в пространстве и сеть пытается построить граф, являющимся приближением областей и их взаимного расположения.

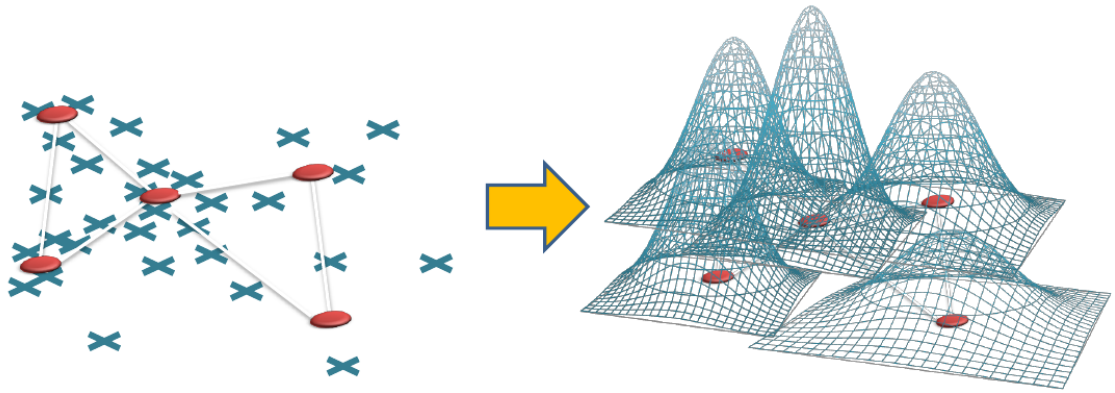

Рис. 7: Обработка данных сетью ESOINN. Образование классов с высокой плотностью

Граф в целом представляет собой смесь распределений, анализируя которую можно определить число классов в исходных данных, их пространственное распределение и прочие характеристики.

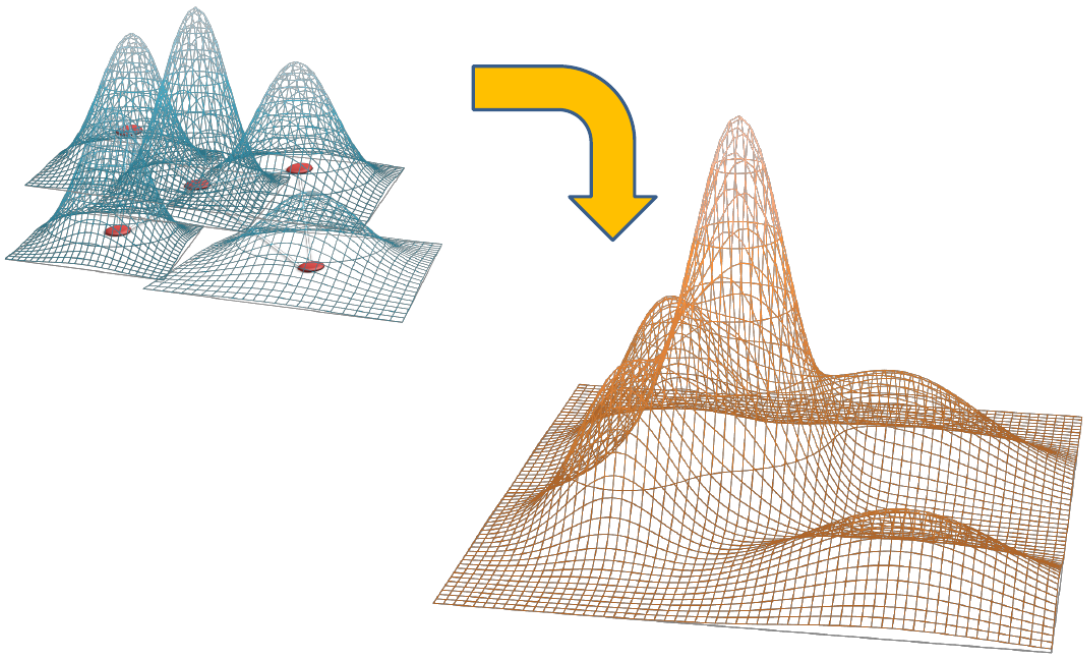

Рис. 8: Обработка данных сетью ESOINN. Выделение классов

Обучение и работа

Данные для обучения берутся из ранее созданного профиля программы. Происходит разделение файла как показано на рисунке 2.9.

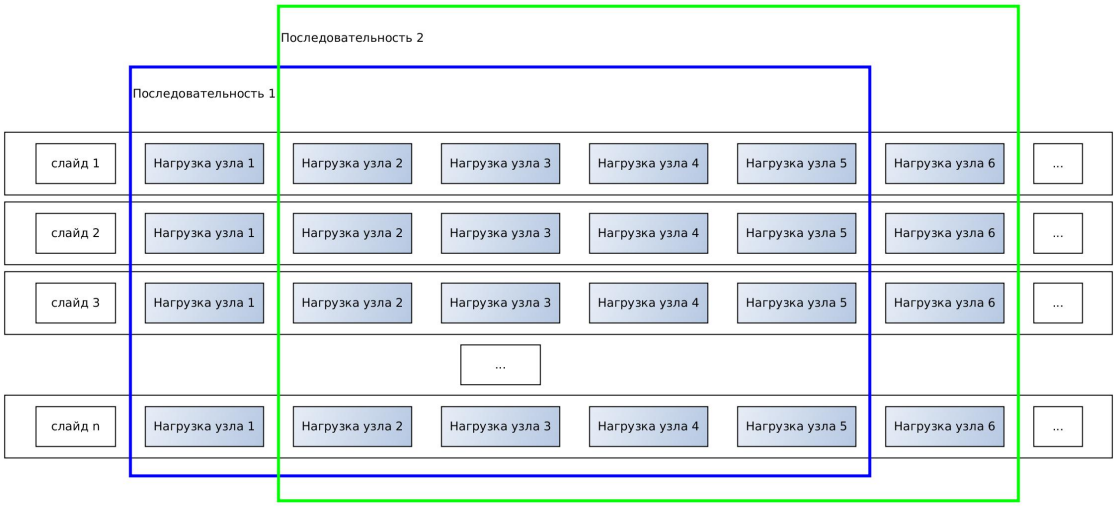

Рис. 9: Разделение профиля программы

Набор нагрузок пяти узлов в одном слайде назовём для удобства *load_vector*. Количество связуемых нагрузок обусловлено необходимостью в обеспечении локальности балансировщика и его скоростью работы. Использование пяти узлов является минимальным количеством узлов, которое позволяет учитывать возможность возникновения дисбаланса для соседних узлов с центральным. Последовательность таких векторов образует историю нагруженности для пяти узлов соответственно. На следующем

этапе формируется набор векторов, состоящих из последовательности load_vector, её назовём *vector5.*

Наборы vector5 уже содержат в себе необходимую информацию о поведении во времени части системы, и они не имеют зависимостей между собой. Используя эту информацию, проводится перемешивание внутри набора для улучшения качества работы нейронной сети.

После подготовки обучающей выборки происходит инициализация нейронной сети и непосредственно её обучение по сформированным данным. По окончании обучения сеть предоставит информацию о количестве обнаруженных классов и вектор-прототип каждого из них.

Каждый обнаруженный класс содержит в себе обобщение свойств входящих в него векторов, а так как обучающая выборка состоит из последовательности работы фрагментированной программы, запущенной ранее, то обнаруженные классы содержат в себе некоторое обобщённое поведение программы. Процесс работы балансировщика, основанного на данной схеме, сводится к формированию четырёх load_vector для центрального узла (состояние загрузки двух соседей слева, справа и состояние центрального узла), после этого формируется vector5, который состоит из полученных четырёх load_vector с добавлением нулевого остатка (load_vector, состоящий из пяти нулей. Использование нулей обусловлено тем, что сеть способна работать с шумом, а этот шум и является нулевым остатком). Полученный вектор сравнивается со всеми векторами-прототипами, предоставленными сетью, и выбирается ближайший к нему. Для сравнения векторов используется евклидова метрика.

После нахождения ближайшего класса, берётся его вектор-прототип. Он так же является vector5 по своей структуре, и его последняя компонента load_vector является прогнозом нагруженности центрального узла и его соседей. Оставшаяся часть балансировщика анализирует прогноз нагруженности и принимает решение о необходимости передачи текущих задач на соседние узлы.

1.2.3 Алгоритм динамической балансировки на основе прогнозирования SOM

Описание SOM

SOM (Self-organized map) - Самоорганизующаяся карта Кохонена, нейронная сеть с обучением без учителя и выполняющая задачу визуализации и кластеризации. Является методом проецирования многомерного пространства в пространство с меньшей размерностью, применяется также для решения задач моделирования, прогнозирования и др. Является одной из версий нейронных сетей Кохонена.

Самоорганизующаяся карта состоит из компонентов, называемых узлами или нейронами. Их количество задаётся аналитиком. Каждый из узлов описывается двумя векторами. Первый — т. н. вектор веса $\mathbf{m}$, имеющий такую же размерность, что и входные

данные. Второй — вектор **r**, представляющий собой координаты узла на карте. Обычно узлы располагают в вершинах регулярной решётки с квадратными или шестиугольными ячейками.

Изначально известна размерность входных данных, по ней некоторым образом строится первоначальный вариант карты. В процессе обучения векторы веса узлов приближаются к входным данным. Для каждого вектора входных данных выбирается наиболее похожий по вектору веса узел, и значение его вектора веса приближается к вектору данных. Также к входящему вектору приближаются векторы веса нескольких узлов, расположенных рядом, таким образом, если в множестве входных данных два наблюдения были схожи, на карте им будут соответствовать близкие узлы. Циклический процесс обучения, перебирающий входные данные, заканчивается по достижении картой допустимой (заранее заданной аналитиком) погрешности, или по совершении заданного количества итераций.

Обучение и работа

Процесс обучения и работы полностью идентичен балансировщику на основе ESOINN. Разница в работе заключается в «статичности» сети, она не разрастается в процессе работы, её размер зависит от выбора аналитика.

1.2.4 Алгоритм динамической балансировки на основе прогнозирования многослойным персептроном

Алгоритм динамической балансировки на основе прогнозирования многослойным персептроном является интеграцией варианта с адаптивной диффузионной балансировкой и использования профиля программы для прогнозирования. Обучение персептронной сети схоже с обучением классификаторов, описанных выше, с разницей в присутствии «учителя» и отличием в выдаваемом результате.

На обучение опять же подаются полученные из профиля vector5, в качестве ожидаемого ответа используется состояние дисбаланса последней компоненты, вектора нагрузки load_vector.

Процесс работы сети также схож с процессом работы классификаторов с разницей в полученном результате. На выходе получаем одно из состояний оценки дисбаланса в прогнозируемый следующий момент времени. Оценка дисбаланса обрабатывается другой компонентой балансировщика и, учитывая решение нейронной сети, выдаётся результат работы балансировщика - информация о необходимости разгрузки или её отсутствие.

1.2.5 Гибридный вариант динамической балансировки на основе прогнозирования и адаптивного алгоритма диффузионной балансировки с нейронной сетью

Последний предложенный алгоритм является гибридным вариантом динамической ба-

лансировки на вычислительные узлы распределённого вычислителя для системы фрагментированного программирования и включает в себя две нейронные сети:

1. Многослойный персептрон
2. Один из вариантов сети с прогнозированием

Следует отметить во избежание недопонимания, данный алгоритм не использует гибридную нейронную сеть, а содержит два типа сетей, работающих раздельно. Лишь анализ полученных результатов использует сразу все данные работы сетей.

Многослойный персептрон используется по аналогии с адаптивным диффузионным алгоритмом динамической балансировки. Второй тип сети был использован из алгоритмов динамической балансировки с прогнозированием.

Необходимость данного алгоритма была вызвана анализом всех вариантах динамической балансировки с прогнозированием, представленных выше. Если внимательно рассмотреть данные по полученному профилю работы программы, можно заметить, что он в себе содержит работу балансировщика, исполняющегося в тот запуск. Это означает наличие корреляции выборки не только с загруженностью, но и с внешним воздействием. На практике это означает, что в случае выполнения программы, когда используемый балансировщик выявил и устранил дисбаланс, данные профиля не будут содержать эту информацию и по факту составленный прогноз из этих данных будет верен только в совокупности с используемым балансировщиком. При использовании лишь только прогноза нагруженности, балансировщик в части случаев будет неверно определять состояние дисбаланса.

Схема работы гибридного варианта балансировщика:

- Обучение
  - Обучение персептронной сети согласно адаптивному диффузионному алгоритму динамической балансировки
  - Обучение прогнозирующей сети согласно соответствующему алгоритму динамической балансировки
- Работа

- Вычисление оценки дисбаланса персептронной сетью
- Получение прогноза нагрузки в следующий шаг исполнения фрагментированной программы
- Корректировка результата с оценкой дисбаланса полученным прогнозом

1.3 Соответствие представленных алгоритмов к требованиям динамического балансировщика

Все представленные алгоритмы динамической балансировки нагрузки на узлы распределённого вычислителя соответствуют поставленным к ним требованиям:

- Масштабируемость - все представленные алгоритмы могут быть адаптированы к различному числу узлов, с которыми они будут взаимодействовать так как выполняется свойства локальности и децентрализации
- Локальность коммуникаций - все представленные алгоритмы обладают свойством локальности коммуникаций, так как коммуникации происходят только с соседними узлами с окрестностью 2
- Децентрализованность - в алгоритме нет точек централизации, все представленные алгоритмы работают независимо друг от друга над определённой, сравнительно небольшой частью распределённого вычислителя
- Адаптация к задаче - все представленные алгоритмы в силу работы с нейронной сетью и обучению на профиле программы, способны адаптироваться к задачам

2 Описание реализации и тестирование

Реализация работы была сделана с использованием языка программирования Ruby, что обусловлено относительно низкими временными затратами на разработку прототипов. Для использования в реальной системе фрагментированного программирования, не в имитационной модели, следует реализовать разработанные алгоритмы на языке C/C++, но это это не входит в поставленную задачу. Общий объем кода составляет 2934 строки. Для запуска имитационной модели необходимо наличие пакета «Ai4r», где хранится реализация многослойного персептрона и SOM. Сеть ESOINN была реализована отдельно из-за отсутствия готового варианта.

2.1 Имитационная модель

Напомним, что тестирование предложенных алгоритмов производилось в разработанной имитационной модели. Для тестирования её адекватности использовались специальные запуски в сравнении с программой, написанной с MPI. Исполняемая задача - централизованная балансировка. Результат представлен на рисунке 2.10. На оси **x** указан номер вычислительного узла, по оси **y** - количество исполненных задач.

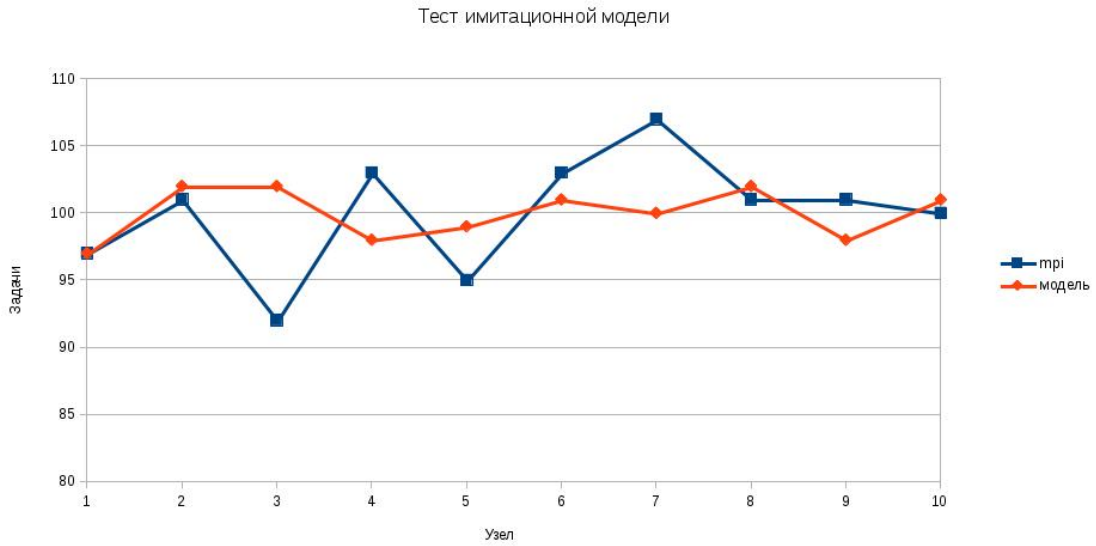

Рис. 10: Тест имитационной модели

Из графика можно сделать вывод об адекватности имитационной модели в сравнении с реальной программой. Небольшие различия в графиках фактором случайности при генерации задач и не являются существенными для анализа свойств алгоритмов балансировки.

Составляющие имитационной модели:

- *main.rb* - файл, содержащий основные конфигурации имитационной модели, включающих в себя выбор балансировщика и опции для нейронных сетей. Также отвечает за инициализацию и сериализацию балансировщиков, вывод результатов
- *executor.rb* - исполнительная часть программы. Инициализирует вычислительные узлы, генерацию вычислительных задач. Является «управляющим драйвером» из описания имитационной модели
- *core.rb* - абстракция вычислительного узла
- *timeline.rb* - компонент вычислительного узла. Регулирует очереди ожидания на исполнения задач. В себе хранит временные метки начала задач
- *feed.rb* - генератор вычислительных задач
- *comm.rb* - компонент, отвечающий за коммуникацию между узлами
- *balancer.rb* - реализации балансировщиков
- *profile.rb* - компонент, отвечающий за составление и анализ профиля программы
- *ai.rb* - классы нейронных сетей и операции для них
- *task.rb* - классы задач для вычислительного узла
- *helper.rb* - набор вспомогательных методов

Запуск имитационной модели:

ruby ./main.rb [число задач на исполнение] [число вычислительных узлов]

Для корректного использования имитационной модели необходимо правильно задать параметры внутри файла main.rb. Их описание (на примере значений по умолчанию):

- #PROFILE CONFIG - опции для регулирования работы с профилем программы
    - $WRITE_PROFILE = false - опция записи профиля
    - $READ_PROFILE = false - опция чтения и анализа профиля для генерации обучающей выборки
- #BALANCE CONFIG - опция выбора балансировщика

- $DIFFUSION_BALANCE = true - опция использования диффузионного динамического балансировщика
- $SIMPLE_NEURON_BALANCE = false - опция использования адаптивного диффузионного динамического балансировщика на основе персептронной сети
- $NEURON5_BALANCE = false - тестовая опция динамического балансировщика на основе персептронной сети. В результатах не участвует. Требует дополнительного анализа и тестирования
- $HYBRID_NEURON_BALANCE = false - тестовая опция динамического балансировщика на основе гибридной нейронной сети. В результатах не участвует. Требует дополнительного анализа и тестирования
- $ESOINN_PREDICTION_BALANCE = false - опция использования динамического балансировщика на основе прогнозирования ESOINN
- $SOM_PREDICTION_BALANCE = false - опция использования динамического балансировщика на основе прогнозирования SOM
- $PERC_PREDICTION_BALANCE = false - опция использования динамического балансировщика на основе прогнозирования многослойного персептрона
- $HYBRID_PREDICTION_BALANCE = false - опция использования гибридного динамического балансировщика

– #HYBRID_PREDICTION CONFIG - выбор используемой нейронной сети для гибридного динамического балансировщика. В случае отсутствия использования прогнозирующих сетей исполняется равносильно «$SIMPLE_NEURON_BALANCE = true»

- $HYBRID_PREDICTION_PERC = false - использовать прогнозирование многослойного персептрона
- $HYBRID_PREDICTION_ESOINN = false - использовать прогнозирование ESOINN
- $HYBRID_PREDICTION_SOM = false - использовать прогнозирование SOM

– #CORE CONFIG - конфигурация работы вычислительных узлов

- $DISTANCE_KOEF_PER_CORE = 10 - задержка для пересылки задач в расчёте на один узел

- $DIFFUSION_THRESHOLD = 15 - порог диффузионной балансировки
- $TASK_CAPACITY_PER_CORE = 30 - вместимость вычислительных задач на один вычислительный узел. Большее количество запрашиваться не будет
- $FEED_REQEST_TIME = 100 - модельное время обновления запроса на получение вычислительных задач активным вычислительным узлом
- $DIFFUSION_BALANCE_TIME = 100 - модельное время работы диффузионной балансировки
- $NEURON_PERC_BALANCE_TIME = 100 - модельное время работы нейронной сети (у всех реализаций используется одно значение)
- $TRANSFER_PACKAGE_CAPACITY = 1 - количество вычислительных задач, которое будет передано балансировщиком в случае обнаружения дисбаланса
- $LCR_STATUS_REQUEST_TIME = 100 - модельное время обновления информации о соседях слева и справа
- $LLCRR_STATUS_REQUEST_TIME = 100 - модельное время обновления информации о двух соседях слева и справа
- $MIN_TASK_DIFF = 1000 - минимальное время исполнения вычислительной задачи
- $MAX_TASK_DIFF = 10000 - максимальное время исполнения вычислительной задачи

– #ESOINN CONFIG - конфигурация сети ESOINN

- $MAX_INT = 2**64
- $MAX_AGE = 100 - максимально допустимый возраст рёбер
- $lambda = 50 - период самообработки сети. Измеряется в количестве поданных векторов данных
- $c1 = 0.0001 - коэффициент для удаления вершины
- $c2 = 10.0 - коэффициент для удаления вершины

– #SOM CONFIG - конфигурация сети SOM

- $NUM_OF_NODES = 5

    - $LAYER_NUM_OF_NODES = 25

- #MISC CONFIG - имена файлов для сериализации балансировщиков

    - $SIMPLE_PERC_SER_NAME = "net_ser"
    - $TEST_PERC_SER_NAME = "net5_ser"
    - $HYBRID_ESOINN_NET_SER_NAME = "hybrid_esoinn_ser"
    - $HYBRID_SOM_NET_SER_NAME = "hybrid_som_ser"
    - $ESOINN_PREDICTION_NET_SER_NAME = "esoinn_prediction_ser"
    - $SOM_PREDICTION_NET_SER_NAME = "som_prediction_ser"
    - $PERC_PREDICTION_NET_SER_NAME = "perc_prediction_ser"

2.1.1 ESOINN

Реализация этой нейронной сети состоит из 3-х файлов для самостоятельной работы или тестирования:

- main.rb
- esoinn.rb
- selftest.rb

Для использования сети как отдельной реализации нужен только файл esoinn.rb и конфигурации из main.rb. Файл selftest.rb предназначен для тестирования работы.

2.2 Тестирование

Запуск различных алгоритмов динамической балансировки производился на параметрах: 1000 вычислительных задач на 50 вычислительных узлов.

Тестирование алгоритмов динамической балансировки только на основе прогнозирования не было проведено из-за обнаруженной корреляции между обучающими данными и работы внешнего балансировщика, который не участвует в работе этих алгоритмов.

Данная проблема описана в подсекции «Гибридный вариант динамической балансировки на основе прогнозирования и адаптивного алгоритма диффузионной балансировки с нейронной сетью»

В графиках адаптации для различных алгоритмов указаны два запуска: до обучения на своих же результатах и после.

Условные обозначения:

- *Эталон* - динамическая диффузионная балансировка
- *Алгоритм1* - адаптивный вариант диффузионной динамической балансировки на основе многослойного персептрона
- *Алгоритм2* - гибридный вариант динамического балансировщика на основе прогнозирования многослойным персептроном
- *Алгоритм3* - гибридный вариант динамического балансировщика на основе прогнозирования ESOINN
- *Алгоритм4* - гибридный вариант динамического балансировщика на основе прогнозирования SOM

2.2.1 Конфигурация 1

Генерация задач со сложностью 1000 - 10000 мс модельного времени

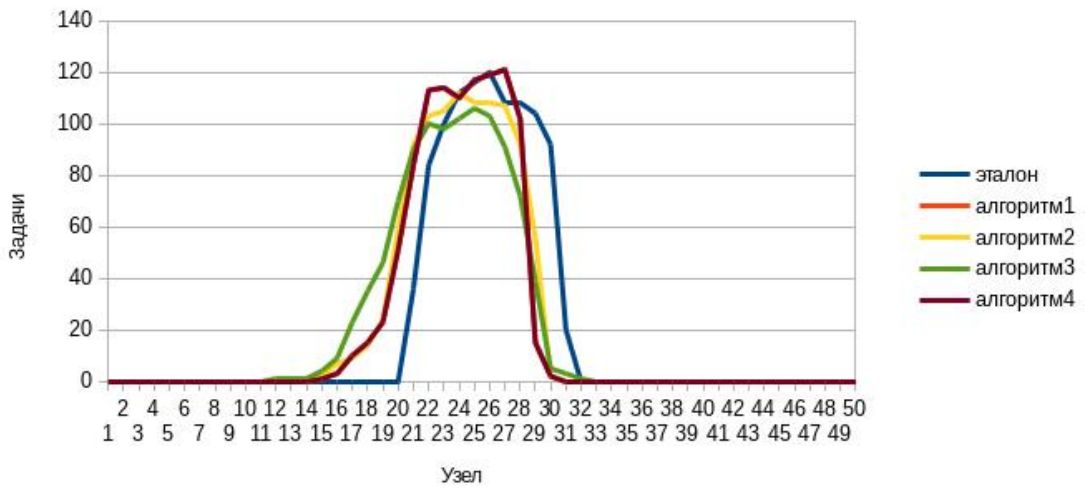

Рис. 11: Конфигурация 1. Первичный запуск

Обучение нейронных сетей, составляющих прогноз производилось по профилю работы диффузионного балансировщика. Алгоритм1 и алгоритм4 имеют идентичный результат запуска.

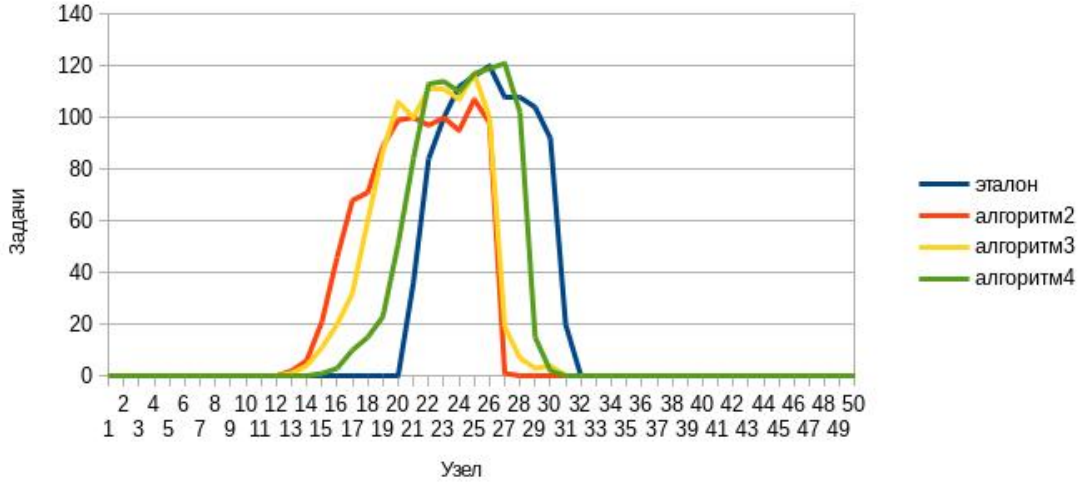

Рис. 12: Конфигурация 1. Вторичный запуск

Использовался профиль предыдущего запуска. Алгоритм 1 не участвовал в повторном тестировании, на текущий момент он не использует профиль для обучения.

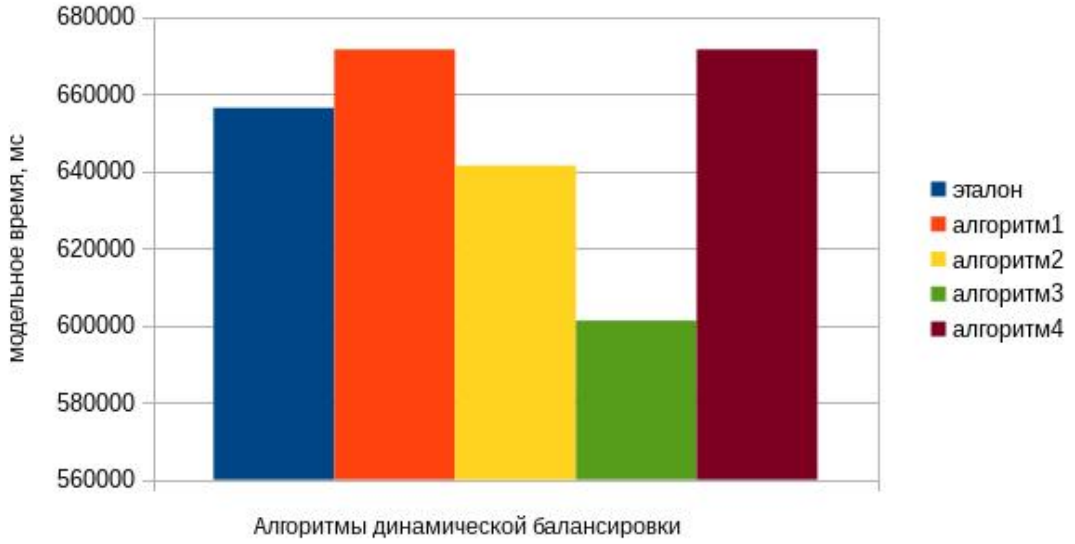

Рис. 13: Конфигурация 1. Сравнение времени работы имитационной модели для первого запуска

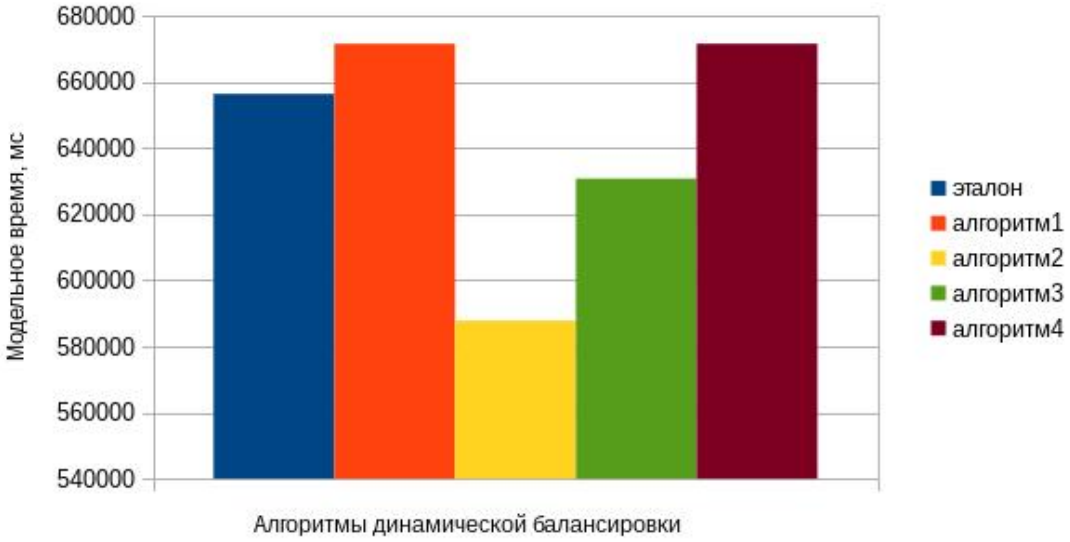

Рис. 14: Конфигурация 1. Сравнение времени работы имитационной модели для второго запуска

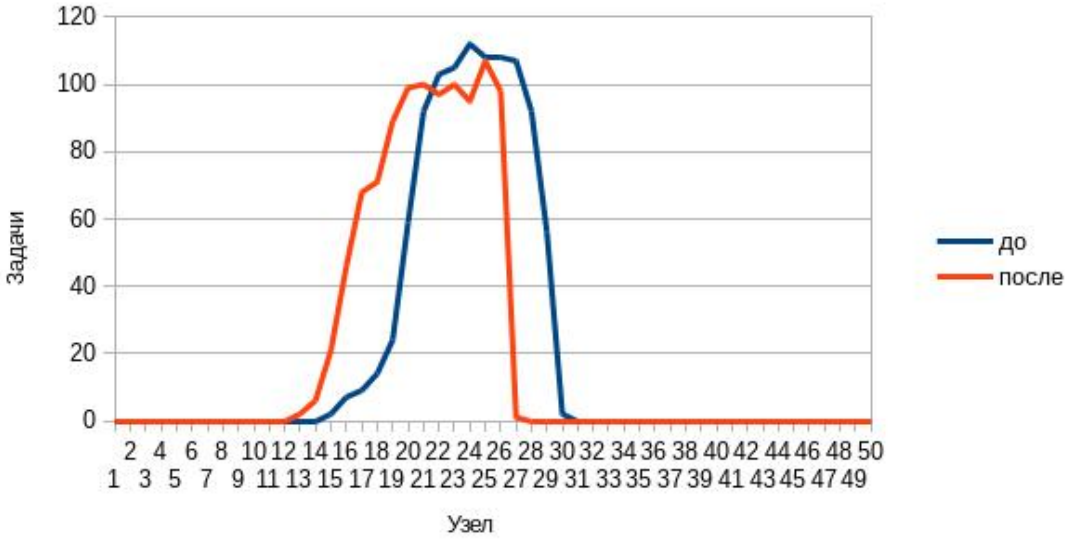

Рис. 15: Конфигурация 1. Адаптация для гибридного варианта динамического балансировщика на основе прогнозирования многослойным персептроном

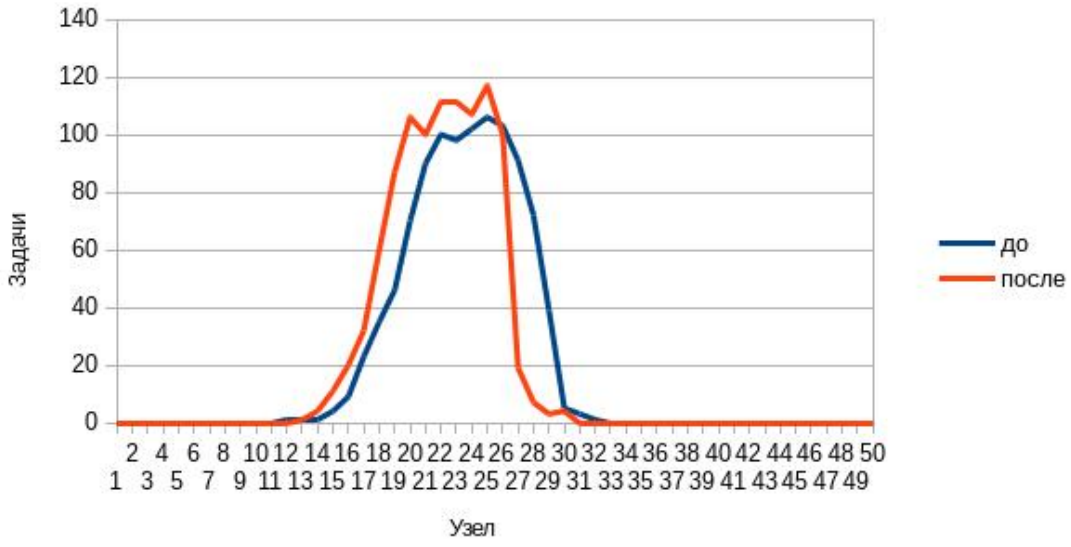

Рис. 16: Конфигурация 1. Адаптация для гибридного варианта динамического балансировщика на основе прогнозирования ESOINN

Изменение работы гибридного варианта динамического балансировщика на основе прогнозирования SOM при использования профиля запуска не произошло на текущей конфигурации.

2.2.2 Конфигурация 2

Генерация задач со сложностью 10000 - 50000 мс модельного времени

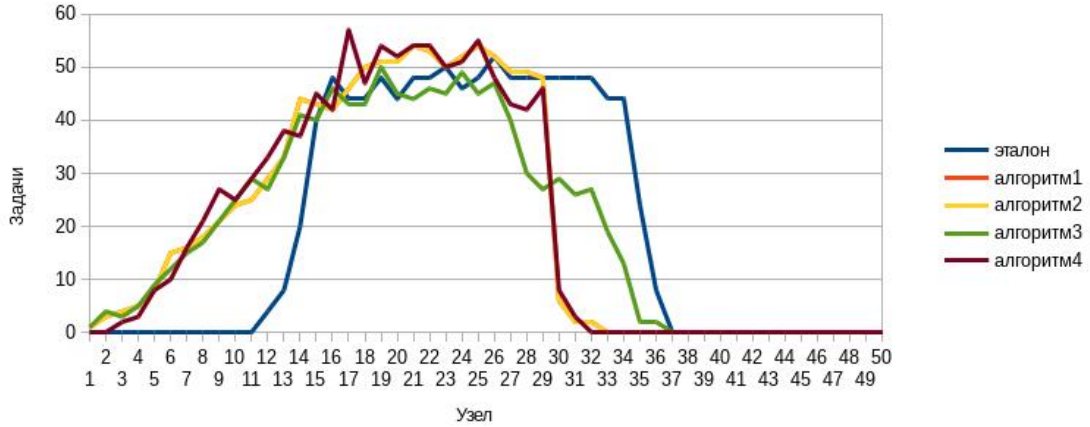

Рис. 17: Конфигурация 2. Первичный запуск

Обучение нейронных сетей, составляющих прогноз производилось по профилю работы диффузионного балансировщика.

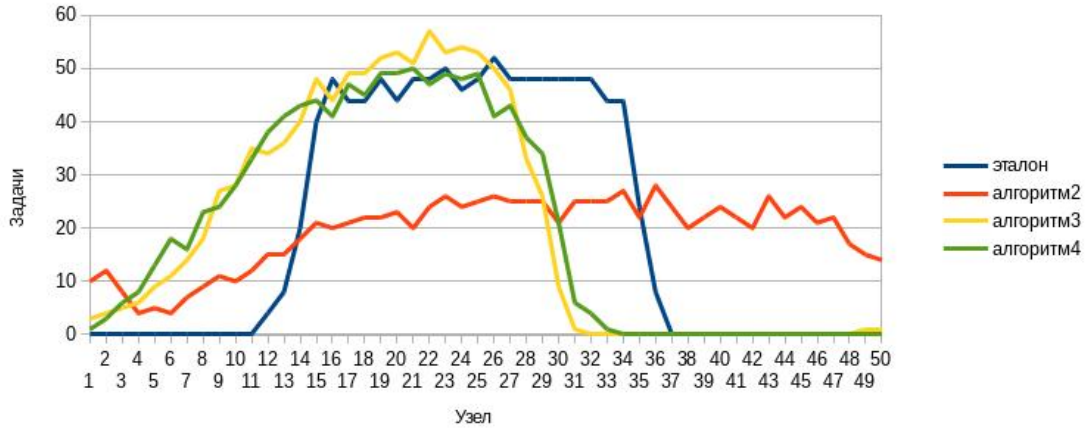

Рис. 18: Конфигурация 2. Вторичный запуск

Использовался профиль предыдущего запуска. Алгоритм 1 не участвовал в повторном тестировании, на текущий момент он не использует профиль для обучения.

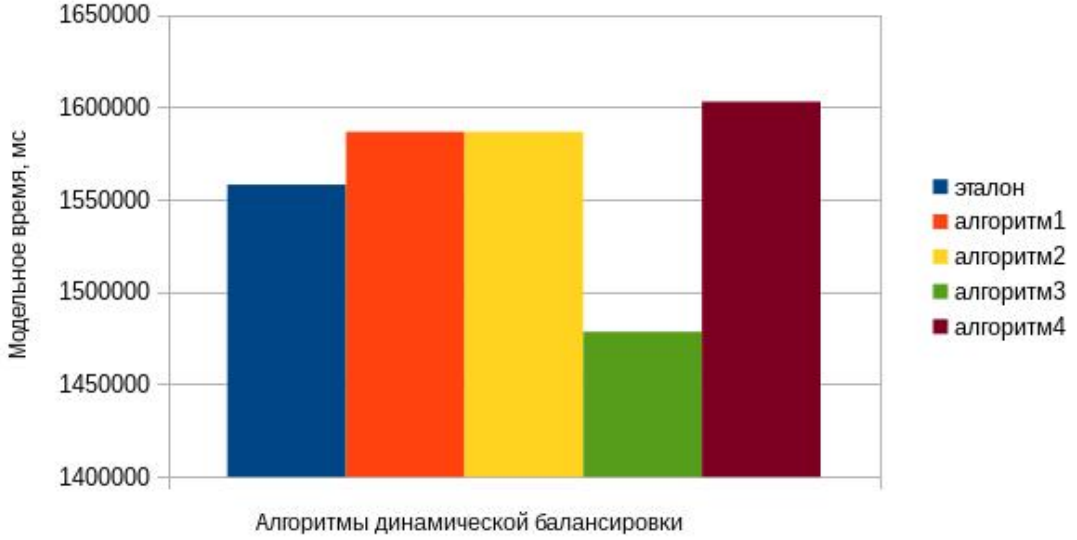

Рис. 19: Конфигурация 2. Сравнение времени работы имитационной модели для первого запуска

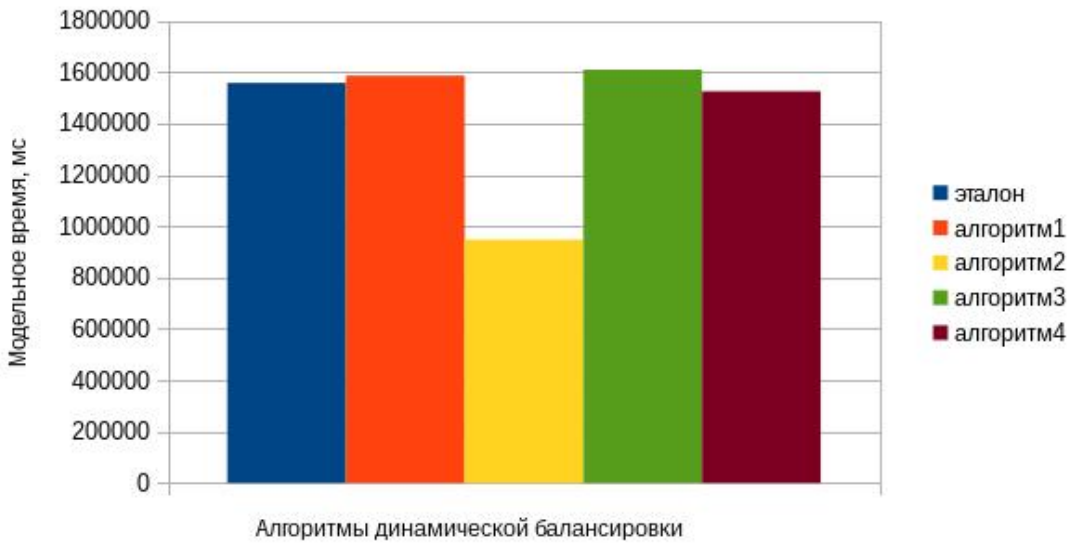

Рис. 20: Конфигурация 2. Сравнение времени работы имитационной модели для второго запуска

Следует отметить, что на диаграмме в рисунке 20 шкала модельного времени начинается с нуля. Данная диаграмма показывает существенную разницу работы балансировщиков.

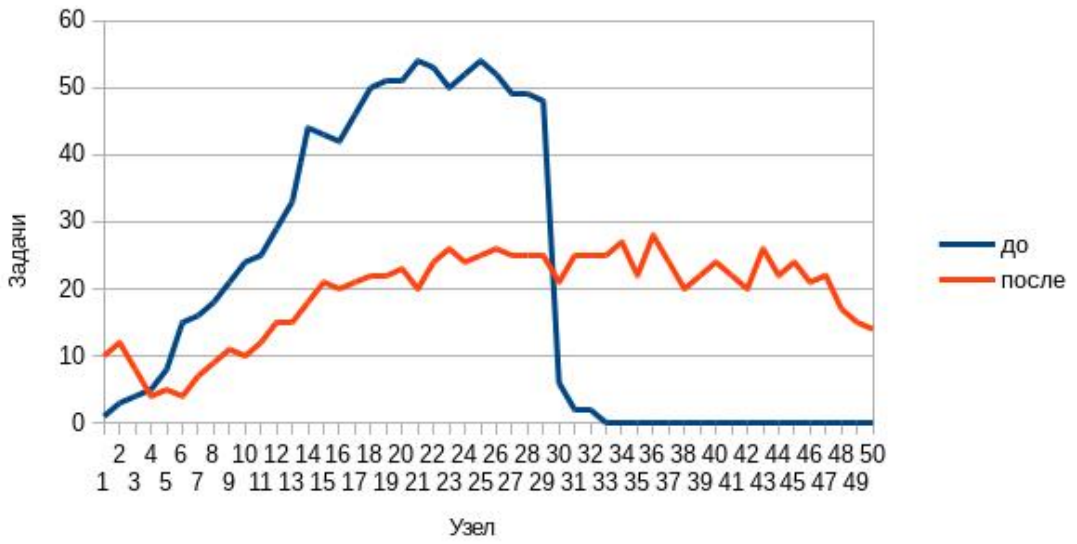

Рис. 21: Конфигурация 2. Адаптация для гибридного варианта динамического балансировщика на основе прогнозирования многослойным персептроном

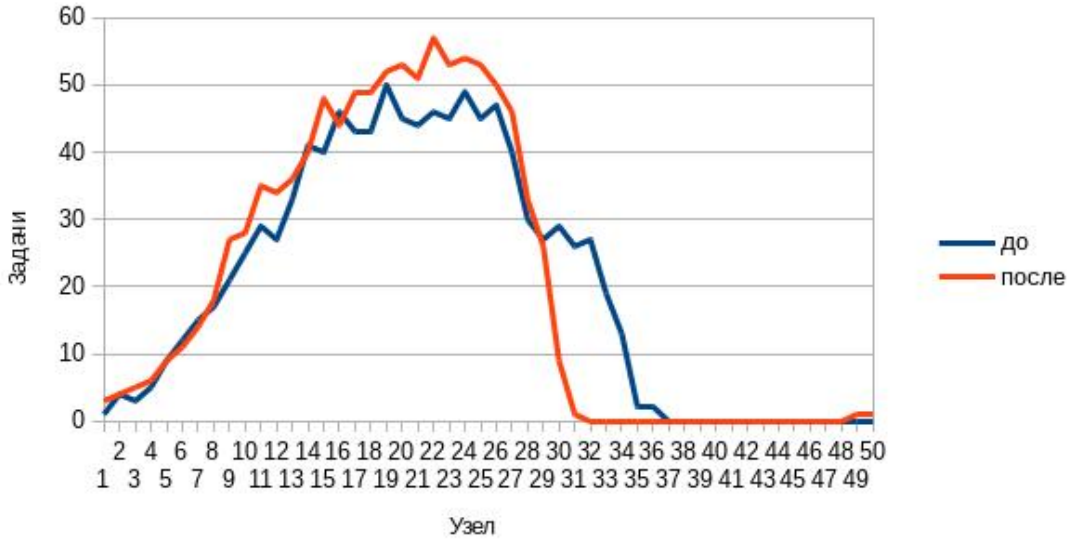

Рис. 22: Конфигурация 2. Адаптация для гибридного варианта динамического балансировщика на основе прогнозирования ESOINN

Рис. 23: Конфигурация 2. Адаптация работы гибридного варианта динамического балансировщика на основе прогнозирования SOM

2.3 Анализ полученных результатов

Анализируя полученные результаты можно точно сказать, что гибридный вариант динамической балансировки использует адаптацию к предыдущим запускам программы. Существенные особенности адаптации для прогнозирующих сетей:

– Многослойный персептрон - результаты запуска на двух конфигурациях показывают улучшение работы балансировщика при использовании профиля программы со своими же результатами. Обучение по результатам работы диффузионного балансировщика (первый запуск) либо не приносило результата, либо он был работал немного лучше эталонного варианта. Стоит отметить значительное улучшение балансировки во второй конфигурации тестирования (примерно в два раза быстрее выполнение программы)

– ESOINN - отмечается кардинальное отличие от работы многослойного персептрона. Обучение на своих же результатах приводят к ухудшению его работы, но при этом обучение на профиле работы программы с диффузионным балансировщиком оказывается весьма эффективным. В этом случае во всех конфигурациях ESOINN показал лучший результат при первом запуске.

– SOM - имеет наименее выразительные черты адаптации. В целом, данная сеть так же способна подстраиваться к задаче, что показано во второй конфигурации тестирования.

Используемый адаптивный алгоритм диффузионной балансировки в гибридном варианте динамической балансировки использовался одинаковый при каждом запуске, более того, идентичный с алгоритм1. Поэтому в анализе адаптации про него не сообщается.

Следует отметить, что ухудшение работы ESOINN может быть связано с тем, что сеть не смогла найти необходимые закономерности в своих результатах работы с заданными внутренними параметрами. Это не означает, что её нельзя использовать, обучая на этих данных, но необходимо произвести более детальное рассмотрение структуры и поведения данной сети, так как при первых запусках она показывала лучшие результаты.

Работа адаптивного алгоритма диффузионной балансировки на основе нейронной сети при всех конфигурациях тестов показал близкий к эталонному результат, но с худшей стороны. Это объясняется тем, что данный алгоритм обучается по работе диффузионной балансировки и не использует профили запуска в текущей реализации. Но даже такие результаты для адаптивного алгоритма диффузионной балансировки являются положительными из-за возможности к адаптации.

Заключение

Была исследована проблема использования нейронных сетей для обеспечения динамической балансировки нагрузки на вычислительные узлы распределённого вычислителя. Предложены и реализованы алгоритмы, решающие поставленную задачу. Проведён их анализ и сравнительное тестирование. Результаты показали применимость нейронных сетей для динамической балансировки нагрузки.

На защиту выносится:

1. Разработана и реализована имитационная модель исполнения фрагментированной программы на мультикомпьютере.
2. Разработаны и реализованы алгоритмы динамической балансировки нагрузки на узлы мультикомпьютера на основе нейронных сетей
3. Проведено сравнительное тестирование предложенных алгоритмов на имитационной модели

Результаты работы были представлены на конференции:

- Международная научная студенческая конференция 2014 “Студент и научно-технический прогресс” (название доклада - “Применение нейронных сетей для динамической балансировки работы мультикомпьютера в системе фрагментированного программирования”)

В дальнейших планах:

- Анализ полученной деградации эффективности прогнозирования ESOINN при обучении на собственных результатах
- Рассмотрение гибридных нейронных сетей для решения задачи динамической балансировки
- Интеграция разработанных алгоритмов в систему фрагментированного программирования

Список литературы

[1] Khan Z. Performance Analysis of Dynamic Load Balancing Techniques for Parallel and Distributed Systems /Z. Khan, R. Singh, J. Alam, and R. Kumar // International Journal of Computer and Network Security. - 2010. - Vol. 2. - No. 2.

[2] Tang X. Optimizing Static Job Scheduling in a Network of Heterogeneous Computers /X. Tang, S. T. Chanson // Proc. of the Intl. Conf. on Parallel Processing. - August 2000. - pp. 373-382.

[3] Bryant R. M. A Stable Distributed Scheduling Algorithm / R. M. Bryant, R. A. Finkel // Proc. 2nd Int. Conf. Dist. Comp. - April 1981. - pp. 323-341.

[4] Chowdhury S. The Greedy Load Sharing Algorithms // J. Parallel and Distributed Comput. - May 1990. - Vol. 9. - pp. 93-99.

[5] Eager D.L. Adaptive Load Sharing in Homogeneous Distributed Systems / D. L. Eager, E. D. Lazowski, J. Zahorjan // IEEE Trans. Software Eng. - May 1986. - Vol. SE-12. - No. 5. - pp. 662-675.

[6] Efe K. Heuristic Models of Task Assignment Scheduling in Distributed Systems // Computer. - June 1982. - Vol. 15. - No. 6. - pp. 50-56.

[7] Goscinski A. Distributed Operating Systems. - Sydney: Addison-Wesley, 1991.

[8] Livny M. Load Balancing in Homogeneous Broadcast Distributed Systems. /M. Livny, M. Melman // Proc. ACM Comput. Network Performance Symp. - 1982. - pp. 47-55.

[9] Karimi A. A New Fuzzy Approach for Dynamic Load Balancing Algorithm /A. Karimi, F. Zarafshan, A. b. Jantan, A. R. Ramli, M. I. Saripan // International Journal of Computer Science and Information Security. - October 2009. - Vol. 6. - No. 1. - pp. 001-005.

[10] Mirchandaney R. Adaptive Load Sharing in Heterogeneous Distributed Systems / R. Mirchandaney, D. Towsley, J. Stankovic // Journal of Parallel and Distributed Computing. - 1990. - No. 9. - pp. 331-346.

[11] Ni L. Optimal Load Balancing in a Multiple Processor System with Many Job Classes / L. Ni, K. Hwang // IEEE Transactions on Software Engineering. - May 1985. Vol. SE-11. - pp. 491-496.

[12] Shivaratri N. Load Distributing for Locally Distributed Systems / N. Shivaratri, P. Krueger, M. Singhal // Computer. - December 1992. - Vol. 25. - No. 12. - p. 33-44.

[13] Stankovic J. A. An Adaptive Bidding Algorithm for Processes, Cluster and Distributed Groups / J. A. Stankovic, I. S. Sidhu // Proc. 4th Int. Conf. Distributed Compu. Sys. - 1984. - pp. 49-59.

[14] Stankovic J. Simulations of Three Adaptive, Decentralized Controlled, Task Scheduling Algorithms // Computer Networks. - June 1984. - Vol. 8. - No. 3. - pp. 199-217.

[15] Stone H. S. High-Performance Computer Architecture // 2nd ed., MA: Addison Wesley, Reading. - 1990.

[16] Zhou S. A Trace-Driven Simulation Study of Dynamic Load Balancing // IEEE Transactions on Software Engineering. - September 1988. - Vol. SE-14. - No. 9. - pp. 1327-1341.

[17] Barak A. A Distributed Load-balancing Policy for a Multicomputer / A. Barak, A. Shiloh // Software-Practice and Experience. - September 1985. - Vol. 15. - No. 9. - pp. 901-913.

[18] Kruger P. The Diverse Objectives of Distributed Scheduling Policies / P. Kruger, P. and M. Livny // Proceedings of the Seventh International Conference in Distributed Computing Systems. - 1987. - pp. 242-249.

[19] Grosu D. Load Balancing in Distributed Systems: An approach Using Cooperative Games. / D. Grosu, Ming-Yeung Leung // IEEE Transactions on Systems, Man, and Cybernetics. - February 2004. vol 34. - pp. 77-84. - DOI: 10.1109/TSMCB.2002.805812.

[20] Malyshkin V. E. LuNA Fragmented Programming System, Main Functions and Peculiarities of Run-Time Subsystem / V. E. Malyshkin, V. A. Perepelkin // Proceedings of the 11th Conference on Parallel Computing Technologis. - 2011. - pp. 53-61.

[21] Малышкин В. Э. Параллельная Реализация Алгоритмов Численного Моделирования // сборник трудов Сибирской школы-семинара по параллельным вычислениям. - 2002.

[22] Khayyat Z. Mizan: A System for Dynamic Load Balancing in Large-scale Graph Processing / Z. Khayyat, K. Awara, A. Alonazi, H. Jamjoom, D. Williams, P Kalnis // Proc. of the 8th ACM European Conference on Computer Systems. - 2013. - pp. 169-182. DOI: 10.1145/2465351.2465369.

[23] Тарков М. С. Отображение Параллельных Программ на Многоядерные Компьютеры Рекуррентными Нейронными Сетями // ПДМ. - 2013. No 2. - сс 50-58.

[24] Furao S. An enhanced self-organizing incremental neural network for online unsupervised learning / S. Furao, T. Ogura, O. Hesegawa // Neural Networks. - 2007. - Vol. 20. - No. 8. - pp 893-903.