МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение  
высшего образования

**«Южно-Уральский государственный университет»**

**(национальный исследовательский университет)**

**Высшая школа электроники и компьютерных наук**

**Кафедра системного программирования**

|  |  |
| --- | --- |
| РАБОТА ПРОВЕРЕНА  Рецензент  Генеральный директор  ООО «Наполеон Айти»  \_\_\_\_\_\_\_\_\_\_\_\_ П.С. Подкорытов  «\_\_\_»\_\_\_\_\_\_\_\_\_\_\_ 2017 г. | ДОПУСТИТЬ К ЗАЩИТЕ  Заведующий кафедрой, д.ф.-м.н., профессор  \_\_\_\_\_\_\_\_\_\_ Л.Б. Соколинский  «\_\_\_»\_\_\_\_\_\_\_\_\_\_\_ 2017 г. |

**РАЗРАБОТКА И ТЕСТИРОВАНИЕ МЕТОДОВ ПРОБЛЕМНО-ОРИЕНТИРОВАННОГО ПЛАНИРОВАНИЯ ПОТОКОВ РАБОТ В РАСПРЕДЕЛЕННЫХ ВЫЧИСЛИТЕЛЬНЫХ СРЕДАХ**

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

ЮУрГУ – 02.03.02.2017.13-018-1382.ВКР

|  |  |
| --- | --- |
|  | Научный руководитель кандидат физ.-мат. наук, доцент  \_\_\_\_\_\_\_\_\_\_ Г.И. Радченко  Автор работы, студент группы КЭ-401  \_\_\_\_\_\_\_\_\_\_И.А. Лыжин  Ученый секретарь  (нормоконтролер)  \_\_\_\_\_\_\_\_\_\_\_\_\_ О.Н. Иванова  «\_\_\_»\_\_\_\_\_\_\_\_\_\_\_ 2017 г. |

Челябинск-2017

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение  
высшего образования

**«Южно-Уральский государственный университет»**

**(национальный исследовательский университет)**

**Высшая школа электроники и компьютерных наук**

**Кафедра системного программирования**

УТВЕРЖДАЮ

Зав. кафедрой СП

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Л.Б. Соколинский

09.02.2017

**ЗАДАНИЕ**

**на выполнение выпускной квалификационной работы бакалавра**

студенту группы КЭ-401

Лыжину Ивану Александровичу,

обучающемуся по направлению

02.03.02 «Фундаментальная информатика и информационные технологии»

1. **Тема работы** (утверждена приказом ректора от 28.04.2017 № 835)

Разработка и тестирование методов проблемно-ориентированного планирования потоков работ в распределенных вычислительных средах.

1. **Срок сдачи студентом законченной работы:** 01.06.2017.
2. **Исходные данные к работе**
3. Официальный сайт симулятора WorkflowSim. [Электронный ресурс] URL: http://www.workflowsim.org/ (дата обращения: 09.02.2017).
4. Радченко Г.И. Модель проблемно-ориентированной облачной вычислительной среды. // Труды Института системного программирования РАН, 2015. – Том 27. – Выпуск 6. – С. 275-284.
5. Deelman E., Gannon D., Shields M., Taylor I. Workflows and e-Science: An overview of workflow system features and capabilities. // Journal Future Generation Computer Systems, 2009. – Vol. 25 – Issue 5. – P. 528-540.
6. Smanchat S., Viriyapant K. Taxonomies of workflow scheduling problem and techniques in the cloud. // Journal Future Generation Computer Systems, 2015. – Vol. 52 – P. 1–12.
7. **Перечень подлежащих разработке вопросов**
8. Анализ существующих алгоритмов планирования ресурсов в облачных системах.
9. Изучение методов и систем для моделирования алгоритмов планирования.
10. Реализация алгоритма планирования PO-HEFT.
11. Разработка прототипа проблемно-ориентированной системы исполнения потоков работ.
12. Сравнительное тестирование алгоритмов.
13. **Дата выдачи задания:** 09.02.2017.

**Научный руководитель**

Доцент кафедры СП,

кандидат физико-математических наук Г.И. Радченко

**Задание принял к исполнению** И.А. Лыжин

**ОГЛАВЛЕНИЕ**

[ГЛОССАРИЙ 6](#_Toc483920433)

[ВВЕДЕНИЕ 7](#_Toc483920434)

[1. КОНЦЕПЦИЯ ПОТОКОВ РАБОТ 10](#_Toc483920435)

[2. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ 12](#_Toc483920436)

[2.1. Обзор алгоритмов планирования 12](#_Toc483920437)

[2.1.1. Виды планирования 12](#_Toc483920438)

[2.1.2. Алгоритмы для потоков работ 12](#_Toc483920439)

[2.1.3. Алгоритмы для независимых задач 14](#_Toc483920440)

[2.1.4. Критерии оптимизации планирования 15](#_Toc483920441)

[2.2. Проблемно-ориентированная среда 15](#_Toc483920442)

[2.3. Обзор подходов к оценке характеристик выполнения задач 17](#_Toc483920443)

[3. МОДЕЛЬ ВЫЧИСЛИТЕЛЬНОЙ СИСТЕМЫ 19](#_Toc483920444)

[3.1. Поток работ 19](#_Toc483920445)

[3.2. Проблемно-ориентированная облачная вычислительная среда 19](#_Toc483920446)

[3.3. Система планирования потоков работ 20](#_Toc483920447)

[3.4. Требования к системе планирования потоков работ 20](#_Toc483920448)

[3.5. Варианты использования системы планирования потоков работ 21](#_Toc483920449)

[4. АРХИТЕКТУРА СИСТЕМЫ И АЛГОРИТМЫ ПЛАНИРОВАНИЯ 23](#_Toc483920450)

[4.1. Общее описание архитектуры системы 23](#_Toc483920451)

[4.2. Описание компонентов, составляющих систему 24](#_Toc483920452)

[4.3. Схема базы данных истории запусков 25](#_Toc483920453)

[4.4. Используемые алгоритмы 26](#_Toc483920454)

[5. РЕАЛИЗАЦИЯ СИСТЕМЫ МОДЕЛИРОВАНИЯ 29](#_Toc483920455)

[5.1. Архитектура WorkflowSim 29](#_Toc483920456)

[5.2. База данных 30](#_Toc483920457)

[5.3. Реализация алгоритма k-ближайших соседей 31](#_Toc483920458)

[5.4. Реализация алгоритма планирования PO-HEFT 32](#_Toc483920459)

[6. ТЕСТИРОВАНИЕ АЛГОРИТМОВ 33](#_Toc483920460)

[6.1. Тестирование алгоритма прогнозирования на отдельных задачах 33](#_Toc483920461)

[6.2. Сравнительное тестирование алгоритмов планирования 34](#_Toc483920462)

[ЗАКЛЮЧЕНИЕ 38](#_Toc483920463)

[ЛИТЕРАТУРА 40](#_Toc483920464)

[ПРИЛОЖЕНИЯ 43](#_Toc483920465)

[Приложение 1 43](#_Toc483920466)

[Приложение 2 45](#_Toc483920467)

[Приложение 3 52](#_Toc483920468)

# ГЛОССАРИЙ

*Поток работ (Scientific Workflow)* – набор задач, связанных между собой зависимостями.

*Scientific Workflow Management System* – система, позволяющая исполнять потоки работ.

*Алгоритм планирования* – алгоритм, упорядочивающий задачи потока и распределяющий их по доступным вычислительным ресурсам.

*Вычислительный ресурс* – компьютер, вычислительный узел, виртуальная машина или контейнер, способные выполнять задачи потока.

*HEFT* – алгоритм планирования для гетерогенных сред, распределяющий задачи в соответствии с вычисленными приоритетами задач. Приоритеты вычисляются на основе вычислительной сложности задачи и объема передачи данных.

*Проблемно-ориентированная среда* – вычислительная среда, в которой набор типов задач фиксирован, и информация об особенностях типа задачи используется для повышения эффективности планирования.

# ВВЕДЕНИЕ

За последние два десятилетия компьютерные вычисления стали устоявшейся «третьей ветвью» науки наряду с теорией и экспериментом [11]. Использование методов суперкомпьютерного моделирования и интеллектуального анализа данных обеспечивает получение новых результатов во всех областях знаний, позволяя проводить численные исследования физических, биологических, социальных и других процессов, обеспечивая реальную альтернативу дорогостоящим (или невозможным) экспериментам [10].

Scientific Workflow Management Systems (SWMS) [11] – специализированный класс программных систем, которые поддерживают решение проблем вычислительной науки в прикладных исследованиях и инженерных задачах. Потоки работ позволяют ученым легко выразить многошаговый вычислительный процесс, состоящий из нескольких задач, которые могут выполняться параллельно или последовательно. Такие задачи могут представлять, например, извлечение данных из базы данных, преобразование данных, выполнение анализа и т.д. Поток работ описывает зависимости между задачами и в большинстве случаев описывается как направленный ациклический граф (DAG), в котором узлы являются задачами, а ребра – зависимостями между задачами.

Очень важной является задача планирования потоков работ, т.е. распределения задач между доступными ресурсами. От решения этой задачи зависят время выполнения потока, количество использованных ресурсов, стоимость выполнения. Существует множество алгоритмов планирования, и каждый из них нацелен на оптимизацию одного или нескольких параметров [1].

Для повышения эффективности использования вычислительных ресурсов создаются проблемно-ориентированные вычислительные среды. Их особенность заключается в том, что они обеспечивают решение задач в рамках конкретной предметной области (биоинформатика, вычислительная гидродинамика, вычислительная химия и др.). В рамках конкретной области набор задач, из которых строятся потоки, является предопределенным, что дает преимущество при планировании таких потоков [28].

Актуальность данной темы обусловлена тем, что потоки работ являются основным подходом в проведении современных научных вычислительных экспериментов и их эффективное планирование крайне важно. Проблемно-ориентированный подход к решению проблемы планирования может дать качественно новые результаты.

*Целью данной работы* является реализация и тестирование методов проблемно-ориентированного планирования в распределенных вычислительных средах.

Для достижения цели работы, необходимо решить следующие задачи:

* изучить существующие методы и алгоритмы планирования;
* изучить существующие подходы к оценке характеристик выполнения задач;
* изучить системы для моделирования алгоритмов планирования;
* разработать и протестировать подсистему прогнозирования характеристик выполнения задач;
* реализовать систему моделирования проблемно-ориентированной среды исполнения потоков работ;
* провести сравнительное тестирование алгоритмов.

**Структура и объем работы**

Работа состоит из введения, шести разделов, заключения, библиографии и трех приложений. Объем работы составляет 42 страницы, объем библиографии – 28 источников, объем приложений – 10 страниц.

Первая глава содержит описание концепции потоков работ, их актуальность и необходимость для проведения научных экспериментов.

Во второй главе проводится анализ предметной области, обзор существующих работ по теме планирования потоков работ и подходов к повышению эффективности планирования путем оценки характеристик задач.

В третьей главе описана модель потока работ и проблемно-ориентированной вычислительной среды, определены требования к системе планирования потоков работ и варианты использования.

В четвертой главе приведены архитектура системы, модель базы данных и используемые в работе алгоритмы.

Пятая глава описывает детали реализации прототипа проблемно-ориентированной системы исполнения потоков работ.

В шестой главе приводятся результаты тестирования алгоритмов в разработанной системе моделирования.

В заключении сделаны выводы о проделанной работе.

Приложение 1 содержит спецификацию вариантов использования системы.

Приложение 2 содержит листинг реализованного алгоритма PO‑HEFT.

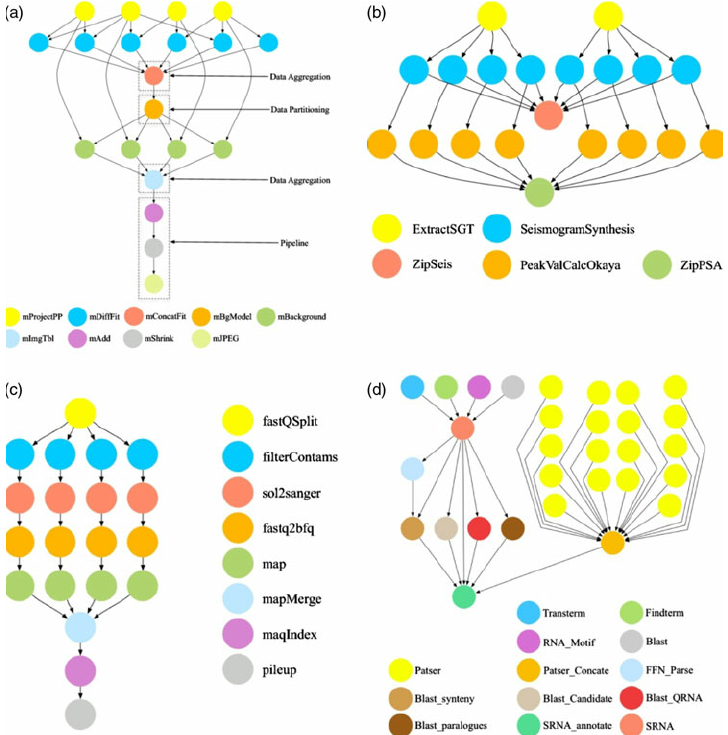
Приложение 3 содержит пример описания потока работы.

# КОНЦЕПЦИЯ ПОТОКОВ РАБОТ

Использование методов суперкомпьютерного моделирования и интеллектуального анализа данных обеспечивает получение качественно новых результатов во всех отраслях знаний, позволяя проводить численные исследования физических, биологических, социальных и др. явлений, предоставляя реальную альтернативу дорогостоящим (или невозможным) экспериментам.

Этот тренд привел к появлению особой научной дисциплины, названной «Computational Science» («Вычислительная наука»). Вычислительная наука – это быстро развивающаяся мульти-дисциплинарная научная дисциплина, использующая передовые вычислительные методы для решения сложных задач, объединяющая в себе методы, алгоритмы и программное обеспечение для компьютерного моделирования, разработанные для решения сложных научных и инженерных задач; а также аспекты фундаментальной информатики и информационных технологий, обеспечивающие развитие аппаратных, программных, сетевых компонентов и СУБД, необходимых для решения таких вычислительно-сложных задач [8].

В рамках Computational Science рассматриваются как отдельные задачи, так и наборы вычислительных задач, которые связаны зависимостями друг с другом. Такие наборы называют Scientific Workflows (потоки работ). Потоки работ позволяют ученым легко выразить многошаговый вычислительный процесс, состоящий из нескольких задач, которые могут выполняться параллельно или последовательно. Такие задачи могут представлять, например, извлечение данных из базы данных, преобразование данных, выполнение анализа и т.д. Поток работ описывает зависимости между задачами, и в большинстве случаев описывается как направленный ациклический граф (DAG), в котором узлы являются задачами, а ребра – зависимостями между задачами. На рисунке 1 представлены примеры реальных потоков работ, применяемых для решения научных задач: Montage (a), CyberShake (b), Epigenomics (c), Sipht (d).



**Рис. 1.** Примеры потоков работ

Зависимости между задачами обычно предполагают, что выходной файл одной задачи поступает на вход другой задаче, поэтому она не может начать выполняться до завершения родительской задачи. При этом, если две связанные задачи выполняются на разных узлах, необходимо передавать соответствующие файлы по сети между этими узлами. В связи с этим некоторые алгоритмы планирования нацелены на то, чтобы распределить задачи по узлам с минимальными расходами на передачу данных.

# АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

## Обзор алгоритмов планирования

Задача планирования потока работ заключается в распределении задач потока по доступным вычислительным ресурсам. В классической постановке задачи предполагается, что набор ресурсов задан изначально и не меняется с течением времени. В последние несколько лет большую популярность получила технология облачных вычислений, которая примечательна тем, что с точки зрения пользователя ресурсы абсолютно не ограничены и в любой момент могут быть добавлены или освобождены.

В общем случае, как и в большинстве частных случаев, задача планирования является NP-полной [25]. Поэтому все существующие алгоритмы используют различные эвристики, чтобы получить результат, близкий к оптимальному. Самые распространенные эвристики – это вычисление приоритетов задач, дублирование и кластеризация. Каждая новая эвристика может дать улучшение в некоторых случаях, поэтому существует немалое множество различных алгоритмов планирования как для отдельных независимых задач, так и для потоков работ в различных вычислительных средах.

### Виды планирования

Планирование бывает двух видов:

* статическое – до начала выполнения известна информация о доступных ресурсах, которые не меняются в процессе исполнения, каждая задача назначается на определенный ресурс до начала исполнения;
* динамическое – происходит в процессе исполнения, набор доступных ресурсов может меняться с течением времени, задача назначается на ресурс непосредственно перед выполнением.

### Алгоритмы для потоков работ

Алгоритмы планирования по типу применяемых эвристик могут быть разбиты на четыре основные категории: list scheduling, duplication based, clustering based и random guided search.

List scheduling алгоритмы являются самыми распространенными и состоят обычно из нескольких фаз. Основные фазы: сортировка задач по глубине в графе потока работ, приоритизация и назначение на ресурсы. Назначение на ресурсы происходит в порядке уменьшения приоритета и обычно задача назначается на ресурс, который обеспечит ее скорейшее выполнение. В эту категорию попадают такие алгоритмы, как HEFT [24], CPOP [24], HCPT [13], HPS [14], PETS [15], Lookahead [2], MH [17].

Duplication based алгоритмы основаны на идее дублирования задач в потоке для уменьшения затрат на передачу данных, таким образом одна задача может быть выполнена на нескольких ресурсах. Такие алгоритмы дают результаты лучше других при планировании одного потока работ. Если в системе появляется несколько потоков, то дублирование задач может привести к увеличению среднего времени выполнения. К тому же применение такой эвристики обычно ведет к увеличению платы за использованные ресурсы. Дублирование задач используется в алгоритмах DBUS [3], TDS [22], DUPS [4].

Clustering based алгоритмы также пытаются снизить затраты на передачу данных между задачами и делают это путем объединения нескольких задач в кластеры. Все задачи одного кластера назначаются на один и тот же вычислительный ресурс, за счет чего между этими задачами не будет происходить обмен данными по сети. Объединение в кластер особенно хорошо применимо для критического пути в графе потока работ. В эту категорию попадают такие алгоритмы, как CHS [12], Triplet [7], LG [18].

Guided random search алгоритмы основаны на случайном поиске оптимального решения в поле всех возможных решений. Однако такие алгоритмы отличаются от обычного случайного поиска тем, что используют информацию с предыдущей итерации. Каждый раз они стараются случайным образом улучшить текущее решение или набор решений. Примерами таких алгоритмов служат генетические алгоритмы [5, 20], а также алгоритмы, описанные в [21]: PSO, Simulated Annealing (SA), Ant Colony Optimization (ACO) и Tabu Search. Алгоритмы этой категории дают весьма хорошие результаты, однако обычно для этого требуется немалое количество итераций поиска, что ведет к большой вычислительной сложности алгоритма.

### Алгоритмы для независимых задач

Алгоритмы для независимых задач могут применяться и для потоков работ, если распределение задач по ресурсам осуществляется динамически во время выполнения потока. При этом планирование происходит только для тех задач, которые на момент планирования уже готовы к исполнению, то есть все родительские задачи уже завершены.

В книге [16] проводится сравнение алгоритмов планирования для независимых задач.

*Opportunistic Load Balancing (OLB)* – данный алгоритм пытается поддерживать все доступные машины в максимально загруженном состоянии. Как только машина освобождается, OLB дает ей следующую задачу, выбирая случайным образом. Преимущество алгоритма заключается в его простоте и отсутствии дополнительных вычислений. Недостатком же является то, что алгоритм никак не учитывает время выполнения задач на конкретных машинах. Этот алгоритм хорошо подходит для систем с одинаковыми вычислительными узлами.

*Minimum Execution Time (MET)* – в противоположность OLB, данный алгоритм для каждой задачи выбирает машину, на которой эта задача будет выполняться наименьшее количество времени. Очевидный недостаток алгоритма заключается в отсутствии какой-либо балансировки нагрузки между вычислительными узлами.

*Minimum Completion Time (MCT)* – объединяет преимущества OLB и MET. Алгоритм выбирает для каждой задачи машину, на которой задача раньше всего сможет завершиться с учетом ее загруженности. Таким образом, общее время выполнения всех задач уменьшается.

*The Min-Min Heuristic* – вычисляет для каждой задачи минимальное время, когда задача может завершиться. Из всех задач выбирает ту, у которой время завершения меньше. Очень похож на алгоритм MCT, за исключением того, что MCT обрабатывает задачи по очереди, а Min-Min рассматривает каждый раз все задачи и выбирает лучшую.

*The Max-Min Heuristic* – в первую очередь выбирает задачи с максимальным временем выполнения и распределяет их на лучшие машины. Это дает позволяет избавиться от недостатка алгоритма Min-Min, когда после выполнения всех маленьких задач несколько больших задач начинают выполняться, при этом оставляя некоторые вычислительные узлы простаивающими.

### Критерии оптимизации планирования

При планировании потоков работ алгоритмы не обязательно нацелены на кратчайшее время выполнения. В качестве критерия оптимизации может выступать стоимость решения, особенно если мы говорим про облачную среду, или потребление энергии, если мы хотим оптимизировать затраты провайдера. Существует ряд алгоритмов, которые выполняют многокритериальное планирование, пытаясь найти золотую середину между оптимизацией нескольких параметров.

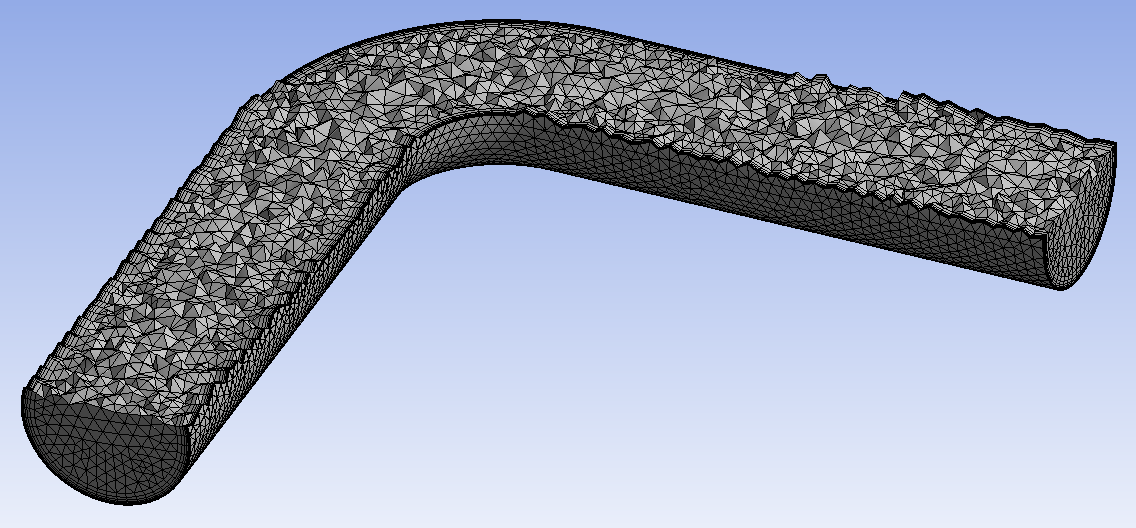
## Проблемно-ориентированная среда

Большинство современных вычислительных систем при постановке задач на выполнение запрашивают у пользователя верхнюю оценку времени выполнения. Эта оценка, во-первых, используется для прерывания зацикленных программ, а во-вторых, для планирования. Зачастую для пользователя сложно дать такую оценку, вследствие чего оценка почти всегда получается сильно завышенной. Это безусловно влияет на качество планирования.

Особенностью проблемно-ориентированных вычислительных сред является то, что они обеспечивают решение заданий в рамках конкретной предметной области. При этом, в рамках предметной области, набор задач, из которых строятся потоки работ, является предопределенным. Задачи могут быть сгруппированы в конечное множество классов. Класс задач представляет собой множество задач, имеющих одну и ту же семантику, а также одинаковые наборы входных параметров и выходных данных.

С одной стороны, это накладывает ограничения на классы задач, которые могут быть решены в рамках вычислительной среды. С другой стороны, такое ограничение позволяет использовать информацию о предметной области для прогнозирования вычислительных характеристик задач при планировании и распределении заданий, увеличивая эффективность использования доступных вычислительных ресурсов [28].

Например, при проведении инженерных расчетов с использованием пакетов численного моделирования OpenFOAM и ANSYS CFX первым шагом является построение сетки (рис. 2). Пользователь задает параметры модели и качество требуемой сетки. Зная эти параметры, проблемно-ориентированная система на основании прошлых запусков может оценить время, необходимое на построение сетки, и размер выходного файла.



**Рис. 2.** Пример сетки, построенной в Ansys

Кроме того, вычислительная система может иметь специальную информацию о типе задачи, которая также позволит более эффективно спланировать выполнение потока работ.

## Обзор подходов к оценке характеристик выполнения задач

Далеко не всегда можно хотя бы приближенно сказать, сколько времени будет выполняться задача, т.к. обычно это зависит от входных данных и параметров запуска. Поэтому совместно с алгоритмами планирования применяются различные техники и алгоритмы прогнозирования характеристик задач. В качестве объекта прогнозирования могут быть: время выполнения задачи или потока работ, количество ресурсов, которое потребуется для выполнения задачи, объем входных и выходных данных и другие параметры.

Существующие подходы и техники оценивания времени выполнения задачи можно разделить на три категории: статические, динамические и гибридные. В статических подходах оценка всего потока и отдельных его задач производится до его непосредственного выполнения. В динамических подходах оценка времени выполнения задачи производится непосредственно перед ее выполнением с учетом текущего состояния системы и доступных ресурсов. Однако, наилучшим подходом следует считать гибридный, который совмещает в себе преимущества статического и динамического.

В статье [8] также вводится понятие MAPE-K цикла (Monitoring, Analysis, Planning, Execution and Knowledge), связанного с динамическим планированием. Изначально оценивается весь поток и строится какой-то план его выполнения. По завершении каждой задачи оцененные значения заменяются реальными, производится переоценка потока и, возможно, перепланировка. Это позволяет сразу исправлять ошибки оценки и не распространять их на весь поток.

В статье [26] для оценки времени выполнения задачи сначала строится регрессионная модель, показывающая зависимость количества операций CPU от размера входных данных. Затем эта модель в комбинации со статическими (частота CPU) и динамическими (например, объем свободной оперативной памяти) параметрами позволяет оценить количество времени, которое потребуется для выполнения задачи на узле.

В статье [9] сначала находится корреляция между размером входных данных и временем выполнения задачи. Для задач с низкой корреляцией и большим стандартным отклонением производится кластеризация алгоритмом DBSCAN. Целью кластеризации является разбиение набора входных данных на более мелкие группы, в которых уже можно будет отследить высокую корреляцию между размером входных данных и временем выполнения задачи.

# МОДЕЛЬ ВЫЧИСЛИТЕЛЬНОЙ СИСТЕМЫ

## Поток работ

Поток работ представляет собой набор задач, связанных между собой зависимостями по данным. Поток работ описывается ориентированным ациклическим графом , где есть множество вершин (задач потока), есть множество ребер (зависимостей между задачами). Задача описывается кортежем , где – тип задачи (один из зарегистрированных в системе), – параметры запуска задачи на выполнение, – набор входных файлов, – набор выходных файлов. Зависимость между задачами описывается парой вершин и означает, что задача не может начать свое выполнение до завершения выполнения задачи .

## Проблемно-ориентированная облачная вычислительная среда

Облачная система исполнения потоков работ в самом общем виде состоит из вычислительного кластера, облачной платформы, развернутой на этом вычислительном кластере и работающей по принципу Infrastructure as a Service (IaaS), и платформы управления потоками работ, обеспечивающей запуск задач в определенном порядке на вычислительном кластере.

Будем считать, что в рамках проблемно-ориентированной облачной вычислительной среды также определены [28]:

*–* множество всех типов задач , которые реализуются в предметной области проблемно-ориентированной облачной вычислительной среды;

– множество образов виртуальных вычислительных машин , доступных для развертывания на узлы;

– характеристика производительности образа виртуальной машины;

– оператор оценки ожидаемого времени выполнения задачи типа при заданном множестве входных информационных объектов на машине с характеристикой производительности ;

,) – оператор оценки общего размера в байтах всех выходных информационных объектов при выполнении задачи типа при значении входных параметров .

## Система планирования потоков работ

Система планирования потоков работ предназначена для составления плана выполнения задач в потоке, т.е. для установления очередности выполнения задач и распределения задач по виртуальным машинам. Алгоритм планирования можно описать как функцию , принимающую на вход поток работ и множество доступных виртуальных машин и возвращающую для каждой виртуальной машины упорядоченный список задач потока работ, которые необходимо выполнить на этой виртуальной машине:

При этом объединение всех задач из результата планирования должно в точности с исходным набором задач потока работ.

Для планирования необходимо знать время выполнения каждой задачи и объем передаваемых данных между задачами. Для этого используются операторы и ,), определенные в проблемно-ориентированной среде.

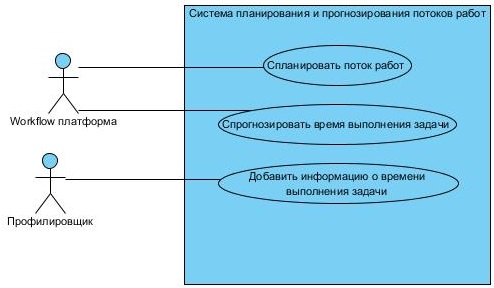
## Требования к системе планирования потоков работ

На основании анализа предметной области и описания модели системы были определены следующие функциональные требования к проектируемой системе:

* система должна оценивать время выполнения задачи при конкретных параметрах запуска и размерах входных данных на конкретных мощностях вычислительного узла;
* система должна планировать потоки работ, т.е. распределять задачи по доступным виртуальным машинам;
* система должна собирать информацию о предыдущих запусках задач для реализации прогнозирования.

## Варианты использования системы планирования потоков работ

В ходе анализа планируемой системы были выявлены основные варианты использования, которые представлены на рисунке 3.

****

**Рис. 3.** Диаграмма вариантов использования системы планирования   
и прогнозирования потоков работ

Были выделены следующие основные актеры, взаимодействующие с системой планирования потоков работ:

* «Workflow платформа» (или платформа потоков работ) – это компонент, отвечающий за подготовку, планирование и выполнение потоков работ в распределенной вычислительной системе.
* «Профилировщик» – это компонент, отвечающий за мониторинг выполнения вычислительных задач и сбор статистики с узлов распределенной вычислительной системы.

Для данных актеров были определены следующие основные варианты использования системы планирования потоков работ:

* «Спланировать поток работ» – назначить каждой задаче из потока работ вычислительный узел, на котором она будет выполняться.
* «Спрогнозировать время выполнения задачи» – оценить время выполнения задачи на конкретных входных данных и конкретном вычислительном узле.
* «Добавить информацию о времени выполнения задачи и размере выходных данных» – добавить в базу данных новую информацию о выполнении задачи, чтобы в будущем точнее предсказывать время ее выполнения и размер выходных данных.

Спецификация вариантов использования приведена в приложении 1.

# АРХИТЕКТУРА СИСТЕМЫ И АЛГОРИТМЫ ПЛАНИРОВАНИЯ

## Общее описание архитектуры системы

Система исполнения потоков работ в самом общем виде состоит из вычислительного кластера, облачной платформы, развернутой на этом вычислительном кластере и работающей по принципу Infrastructure as a Service (IaaS), и Workflow-платформы, обеспечивающей запуск задач в определенном порядке на вычислительном кластере. Общую схему можно видеть на рисунке 4.



**Рис. 4.** Общая схема системы исполнения потоков работ

Workflow-платформа состоит из нескольких компонентов: парсер DAX, подсистема прогнозирования, планировщик, workflow engine, база данных, хранящая статистику с предыдущих запусков. Кроме того, на каждой виртуальной машине на вычислительном кластере запущен профилировщик, который собирает всю необходимую статистику и сохраняет ее в базу данных. Более детальную схему всей системы можно видеть на рисунке 5.



**Рис. 5.** Детальная схема системы исполнения потоков работ

В рамках дипломной работы реализуется подсистема прогнозирования, которая затем используется для реализации алгоритма PO-HEFT.

## Описание компонентов, составляющих систему

Разберем более подробно отдельные компоненты, составляющие программную систему.

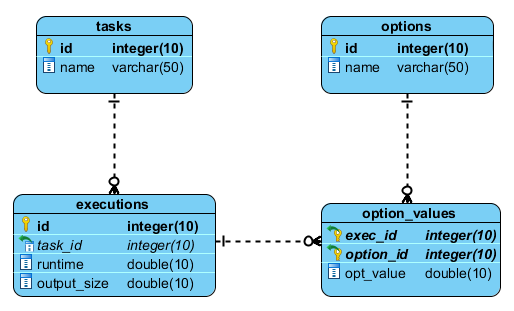
1. *Парсер DAX* – компонент, получающий на вход файл XML, содержащий информацию о задачах, входящих в поток, и их зависимостях, и выдающий список задач с зависимостями в некотором внутреннем представлении, с которым работают остальные компоненты системы.
2. *База предыдущих запусков* – хранит статистику о предыдущих запусках отдельных задач. Статистика по задаче включает в себе параметры запуска, размер выходных данных и время исполнения.
3. *Подсистема прогнозирования* – используя информацию из базы предыдущих запусков, оценивает время выполнения и размер выходных данных при конкретных параметрах запуска.
4. *Планировщик* – получает на вход список задач с зависимостями и временными оценками, запрашивает у облачной платформы информацию о доступных ресурсах и строит план выполнения потока работ.
5. *Workflow Engine* – действует по предоставленному плану и отправляет задачи на выполнение только при условии, что все родительские задачи уже завершены.
6. *Профилировщик* – работает на каждой виртуальной машине и собирает статистику о выполнении задач.

## Схема базы данных истории запусков

Для того чтобы прогнозировать время выполнения задачи, необходимо иметь статистику по предыдущим запускам задач данного типа. Для хранения этой статистики необходима база данных, которая будет наполняться профилировщиками и использоваться системой прогнозирования. В базе данных по каждому запуску задачи необходимо хранить параметры запуска, размеры входных файлов, время выполнения и размер выходного файла. Соответственно, необходимы следующие таблицы:

* tasks – содержащая имена задач, которые могут выполняться в нашей системе.
* options – содержащая параметры, с которыми могут запускаться задачи.
* executions – хранит статистику о запусках задач.
* option\_values – для каждого запуска хранит значения параметров, с которыми запускалась задача.

В итоге получаем базу данных, схема которой изображена на рисунке 6.



**Рис. 6.** Схема базы данных истории запусков

## Используемые алгоритмы

Для планирования потоков работ используется списочный алгоритм проблемно-ориентированного планирования приложений в облачных средах с учетом их вычислительных профилей PO-HEFT [19]. Предложенный алгоритм основан на алгоритме Heterogeneous Earliest-Finish-Time (HEFT), но содержит модификации при вычислении уровня узла задачи и учитывает входящую коммуникационную стоимость его родительских задач.

Отдельная задача потока – это отдельный экземпляр типа задачи с определенным набором входных информационных объектов :

|  |
| --- |
| . |

Определим R – как множество доступных для развертывания виртуальных машин со средней производительностью мощностью:

|  |
| --- |
|  |

В этом случае, для оценки времени выполнения можно применить оператор оценки ожидаемого времени выполнения функции:

|  |
| --- |
| . |

В рамках модели проблемно-ориентированных сервисов необходимо учитывать объем данных, возвращаемых каждой задачей . Для этого может быть использован оператор оценки ожидаемого выхода ,). Следовательно, в рамках проблемно-ориентированной модели для оценки времени передачи данных между двумя задачами может быть использована формула:

|  |
| --- |
|  |

где: представляет собой пропускную способность канала передачи данных в облачной вычислительной системе.

Таким образом, приоритет вычислительного блока может быть определен, как:

|  |
| --- |
| , |

где: – множество всех задач, которые зависят от задачи .

Псевдокод алгоритма PO-HEFT представлен в листинге 1.

**Листинг 1.** Псевдокод алгоритма планирования PO-HEFT

**ПРОЦЕДУРА:** PO-HEFT

**ВХОДНЫЕ ДАННЫЕ:** ГрафЗадач G(T, E), **значения входных параметров** СписокРаспределенийЗадач, МножествоРесурсов R

**НАЧАЛО**

**для каждого** t T из графа задач G

Оценить время выполнения задачи

**для каждого** eE из графа задач G

Оценить время передачи данных

Запустить обход в ширину в обратной последовательности задач и рассчитать ранг для каждой задачи

**пока** в T есть незавершенные задачи

СписокЗадач получить завершенные задачи из графа задач G

Назначить Задачу (СписокЗадач, R)

Обновить СписокРаспределенийЗадач

**КОНЕЦ**

**ПРОЦЕДУРА**: НАЗНАЧИТЬ ЗАДАЧУ

**ВХОДНЫЕ ДАННЫЕ:** СписокЗадач, Множество Ресурсов R

**НАЧАЛО**

Отсортировать СписокЗадач в порядке убывания значения ранга задачи

**для каждого** t СписокЗадач

r получить ресурс из R, способный завершить задачу t раньше

распределить задачу t на ресурс r

обновить статус r

**КОНЕЦ**

Для реализации операторов и применим эвристический метод прогнозирования, основанный на анализе результатов предыдущих запусков аналогичных задач на основе алгоритма k-ближайших соседей.

**Шаг 1.** Выбрать из базы данных информацию о всех предыдущих запусках задач типа .

**Шаг 2.** Для каждого предыдущего запуска вычислить расстояние относительно прогнозируемого запуска по значениям входных параметров. Для расчета расстояния между значениями входных параметров будем применять стандартное Евклидово расстояние:

*,*

где и – размеры -го параметра для текущего и -го запуска соответственно.

**Шаг 3.** Выбрать предыдущих запусков, наиболее близких по значениям параметров в текущему запуску.

**Шаг 4.** Вычислить среднее время выполнения и средние размеры выходных файлов для выбранных предыдущих запусков.

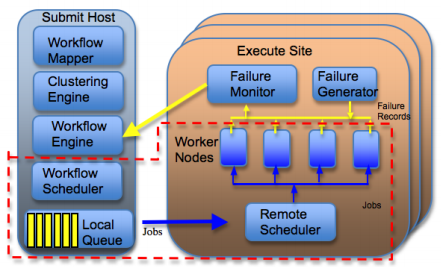
# РЕАЛИЗАЦИЯ СИСТЕМЫ МОДЕЛИРОВАНИЯ

Для проверки эффективности проблемно-ориентированного подхода и используемых алгоритмов в соответствии с архитектурой, описанной в предыдущем разделе, была реализована система моделирования на базе симулятора системы исполнения потоков работ WorkflowSim.

В качестве основного языка программирования для реализации системы был выбран язык Java, т.к. симулятор WorkflowSim реализован на этом языке. В качестве вспомогательного был использован язык Python.

## Архитектура WorkflowSim

WorkflowSim является расширением симулятора CloudSim, обеспечивая поддержку потоков работ с симуляцией ошибок и задержек, которые могут возникать в ходе выполнения потока [27].WorkflowSim состоит из следующих компонентов (рисунок 7) [6].

****

**Рис. 7.** Схема WorkflowSim (компоненты, обведенные красной линией, поддерживаются CloudSim)

*Workflow Mapper (Planner)* – используется для импортирования файлов, содержащих описание графа, представляющего поток работ. Описание содержится в XML-файле, называемом в WorkflowSim DAX (Directed Acyclic Graph + XML). Кроме того, на этапе Workflow Planner может применяться Planning алгоритм (статический алгоритм планирования потока работ), который оценивает весь поток и сразу сопоставляет задачи в потоке с конкретными вычислительными узлами.

*Clustering Engine* – объединяет задачи в работы, где под работой понимается совокупность задач, которые должны быть выполнены на одном вычислительном узле. Работа – это элементарная единица для вычислительной системы. В WorkflowSim реализованы следующие алгоритмы кластеризации: вертикальное, горизонтальное и блочное.

*Workflow Engine* – управляет работами, основываясь на их зависимостях. Как только все родительские работы будут выполнены, Workflow Engine передаст работу в Workflow Scheduler.

*Workflow Scheduler* – сопоставляет работы с конкретными ресурсами, если это не было сделано на этапе планирования. На этом этапе могут применяться динамические алгоритмы планирования или алгоритмы планирования независимых задач.

*Failure Generator* – служит для генерации исключительных ситуаций.

*Failure Monitor –* собирает информацию о ошибках, которую в последующем использует Clustering Engine.

## База данных

В соответствии с разработанной схемой, была реализована база данных для хранения информации о предыдущих запусках задач. Для реализации была использована СУБД PostgreSQL. SQL-скрипт создания таблиц представлен в листинге 2.

**Листинг 2.** SQL-скрипт создания таблиц в базе данных

CREATE TABLE tasks

(

id serial NOT NULL,

name character varying(50) NOT NULL,

CONSTRAINT tasks\_pkey PRIMARY KEY (id)

);

CREATE TABLE options

(

id serial NOT NULL,

name character varying(50) NOT NULL,

CONSTRAINT options\_pkey PRIMARY KEY (id)

);

CREATE TABLE executions

(

id serial NOT NULL,

task\_id serial NOT NULL,

runtime double NOT NULL,

output\_size double NOT NULL,

CONSTRAINT exec\_pkey PRIMARY KEY(id),

CONSTRAINT taskid\_fkey FOREIGN KEY(task\_id) REFERENCES tasks(id) ON   
DELETE CASCADE ON UPDATE CASCADE

);

CREATE TABLE option\_values

(

exec\_id serial NOT NULL,

option\_id serial NOT NULL,

opt\_value double NOT NULL,

CONSTRAINT ov\_pkey PRIMARY KEY(exec\_id, option\_id),

CONSTRAINT execid\_fkey FOREIGN KEY(exec\_id) REFERENCES executions(id) ON DELETE CASCADE ON UPDATE CASCADE,

CONSTRAINT optid\_fkey FOREIGN KEY(option\_id) REFERENCES options(id) ON DELETE CASCADE ON UPDATE CASCADE

);

## Реализация алгоритма k-ближайших соседей

В листинге 3 представлена реализация функции предсказания на основе алгоритма k-ближайших соседей, написанную на языке Java.

**Листинг 3.** Реализация алгоритма k-ближайших соседей

public static double predictRuntime(Map<String, Double> curOptions, List<TaskExecution> prevExecutions) {  
 List<Pair<Double, Double>> dists = new ArrayList<>(prevExecutions.size());  
 for(TaskExecution exec : prevExecutions)  
 dists.add(new Pair<>(*getDist*(curOptions, exec.options), exec.runtime));  
 dists.sort((o1, o2) -> {  
 if(o1.getKey()<o2.getKey()) return -1;  
 if(o1.getKey()>o2.getKey()) return 1;  
 return 0;  
 });  
 double sumRuntime = 0;  
 for(int i=0; i < *K*; ++i)  
 sumRuntime += dists.get(i).getValue();  
 return sumRuntime / *K*;  
}

private static double getDist(Map<String, Double> opts1, Map<String, Double> opts2) {  
 double dist = 0;  
 for(String optName : opts1.keySet()) {  
 double delta = opts1.get(optName) - opts2.get(optName);  
 dist += delta\*delta;  
 }  
 return Math.sqrt(dist);  
}

## Реализация алгоритма планирования PO-HEFT

Для реализации алгоритма планирования PO-HEFT были объединены воедино алгоритм HEFT и система прогнозирования. Полная реализация алгоритма планирования PO‑HEFT приведена в приложении 2.

# ТЕСТИРОВАНИЕ АЛГОРИТМОВ

## Тестирование алгоритма прогнозирования на отдельных задачах

Для тестирования системы прогнозирования был использован набор из 45 потоков работ проекта Pegasus [23]. Каждый из потоков работ представлен XML-файлом с расширением «dax», что является аббревиатурой от «Directed Acyclic graph Xml». Пример dax-файла с описанием потока работ приведен в приложении 3. Общее количество задач во всех собранных потоках работ составило около 24 000. Этот набор был разбит на два блока. Первый блок был загружен в базу предыдущих запусков, на втором блоке был запущен алгоритм оценки характеристик задачи. Значение параметра было установлено равным 10. Для каждого запуска вычислялась относительная погрешность по формуле:

В результате была вычислена средняя относительная погрешность по всем прогнозируемым запускам. Все собранные данные о задачах разбивались в разном соотношении на два блока, и для каждого разбиения было произведено тестирование на предсказание отдельных задач. Результаты тестирования приведены в таблице 1 и на рис. 8.

**Табл. 1.** Результаты тестирования алгоритма прогнозирования

|  |  |  |  |
| --- | --- | --- | --- |
| № теста | Отношение разбиения на блоки | Относительная погрешность оценки времени работы (runtime), % | Относительная погрешность оценки объема выходных данных (outsize), % |
| 1 | 50/50 | 6.7 | 6.3 |
| 2 | 60/40 | 6.2 | 5.8 |
| 3 | 70/30 | 6.0 | 5.6 |
| 4 | 80/20 | 5.5 | 5.3 |
| 5 | 90/10 | 5.0 | 4.8 |

Как видно из результатов тестирования, точность предсказания времени работы задачи и объема выходных данных на представленной выборке составляет от 7 до 5 %, уменьшаясь с увеличением размера выборки.

**Рис. 8.** Результаты тестирования алгоритма прогнозирования

## Сравнительное тестирование алгоритмов планирования

Была произведена симуляция исполнения потоков работ с использованием следующих алгоритмов:

* *HEFT*, реализованный с использованием точных значений времени выполнения и размера выходных данных, полученных из DAX-файлов;
* *PO-HEFT,* реализующий оценку времени выполнения и объема выходных данных на основе информации, полученной от системы прогнозирования;
* *RANDOM*, обеспечивающий случайное распределение задач по узлам.

Была использована стандартная реализация алгоритмов *HEFT* и *RANDOM* из пакета WorkflowSim. Для тестирования были взяты потоки работ CyberShake (на 30, 50, 100 и 1000 задач), Epigenomics (на 24, 46, 100 и 997 задач) и Inspiral (на 30, 50, 100 и 1000 задач) [23]. Для каждого потока в качестве тестового брался самый большой поток, а меньшие потоки загружались в базу предыдущих запусков для оценки качества прогнозирования.

В первой серии тестовых запусков, исследование проводилось на наборе из 5 одинаковых виртуальных машин со следующими параметрами:

* MIPS (million instructions per second) = 1000;
* Bandwidth (пропускная способность) = 1000 MB/s.

Для получения консистентных результатов, алгоритм RANDOM запускался 5 раз, после чего в качестве результата определялось среднее значение. Результаты тестирования алгоритмов HEFT, PO-HEFT и RANDOM представлены в таблице 2 и на рисунке 9.

**Табл. 2.** Сравнение алгоритмов в гомогенной вычислительной среде

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Алгоритм  Поток работ | Время исполнения потока работ  (в скобках указано относительное замедление по сравнению с HEFT) | | | | |
| HEFT | PO-HEFT | | RANDOM | |
| CyberShake\_1000 | 4 754 | 4 957 | (+4.3%) | 5 368 | (+12.9%) |
| Epigenomics\_997 | 776 051 | 789 115 | (+1.7%) | 885 290 | (+14.1%) |
| Inspiral\_1000 | 45 716 | 4 6791 | (+2.3%) | 50 860 | (+11.3%) |

**Рис. 9.** Сравнение алгоритмов планирования потоков работ в гомогенной вычислительной среде

Для оценки влияния значения параметра К, была проведена серия экспериментов с использование алгоритма PO-HEFT и различными значениями параметра K (1, 3, 5, 10) в алгоритме K-ближайших соседей. Результаты тестирования алгоритма PO-HEFT в зависимости от количества информации в базе данных и от параметра K представлены в таблице 3.

**Табл. 3.** Результаты тестирования алгоритма PO-HEFT на потоках работ

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Поток | K=1 | K=3 | K=5 | K=10 |
| CyberShake\_1000 | 4 843 | 4 843 | 4 942 | 5 034 |
| Epigenomics\_997 | 789 076 | 789 053 | 789 394 | 789 087 |
| Inspiral\_1000 | 47 612 | 46 524 | 47 301 | 46 910 |

Результаты первой серии эксперимента позволяют сделать следующие выводы:

* на анализируемых потоках работ, даже случайное распределение задач (алгоритм RANDOM) по вычислительным узлам в гомогенной вычислительной среде дает относительно хорошие результаты планирования, всего на 10 – 15 % хуже, чем алгоритм HEFT;
* результаты алгоритма PO-HEFT, в среднем, показывают результаты на 2–4 % хуже, чем результаты алгоритма HEFT;
* задачи в анализируемых потоках работ характеризуются сильной прямой зависимостью между объемом входных данных, временем исполнения задачи и объемом выходных данных. В связи с этим, в среднем, алгоритм PO‑HEFT показывает одинаково хорошие результаты при использования значений параметра k = 1..10 в алгоритме k-ближайших соседей.

Во второй серии тестирования было создано 5 виртуальных машин с параметрами, представленными в таблице 4.

**Табл. 4.** Параметры виртуальных машин второй серии экспериментов

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | VM\_0 | VM\_1 | VM\_2 | VM\_3 | VM\_4 |
| MIPS | 200 | 400 | 600 | 800 | 1 000 |
| Bandwidth (MB/s) | 200 | 400 | 600 | 800 | 1 000 |

Для второй серии экспериментов, значения параметра k для алгоритма PO-HEFT было принято равным 10. Усредненные результаты тестирования алгоритмов HEFT, PO-HEFT и RANDOM представлены в таблице 5 и на рисунке 10.

**Табл. 5.** Сравнение алгоритмов во второй серии тестирования

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Алгоритм  Поток работ | Время исполнения потока работ  (в скобках указано относительное замедление по сравнению с HEFT) | | | | |
| HEFT | PO-HEFT | | RANDOM | |
| CyberShake\_1000 | 7 795 | 8 106 | (+3.8%) | 23 716 | (+204%) |
| Epigenomics\_997 | 1 294 702 | 1 331 352 | (+2.8%) | 3 995 491 | (+209%) |
| Inspiral\_1000 | 79 130 | 79 051 | (-0.1%) | 230 608 | (+191%) |

**Рис. 10.** Сравнение алгоритмов во второй серии тестирования

На основании полученных результатов можно сделать вывод, что разрабатываемый алгоритм PO-HEFT показал незначительно более низкое (1 ‑ 4 %) время выполнения потока работ по сравнению с алгоритмом HEFT. При этом, алгоритм HEFT в процессе планирования использует априорную информацию о времени выполнения задачи и объеме выходных данных. В то же время эффективность алгоритма PO-HEFT в несколько раз превосходит алгоритм RANDOM в случае, когда в облачной вычислительной среде возможно использование виртуальных машин, обладающих различным уровнем вычислительных возможностей.

# ЗАКЛЮЧЕНИЕ

В рамках дипломного проекта был разработан прототип системы прогнозирования потоков работ в проблемно-ориентированных вычислительных средах и протестирован алгоритм PO-HEFT. Также был разработан прототип проблемно-ориентированной облачной системы исполнения потоков работ. В ходе разработки прототипа были решены следующие задачи:

* изучены существующие алгоритмы планирования ресурсов в облачных системах;
* найдены и изучены методы и системы для моделирования алгоритмов планирования;
* реализован алгоритм планирования PO-HEFT;
* разработан прототип проблемно-ориентированной вычислительной среды для выполнения потоков работ;
* проведено сравнительное тестирование алгоритмов.

Данная работа выполнялась в рамках Гранта РФФИ № 15-29-07959 офи-м (2015-2017 гг.): «Разработка методов и алгоритмов планирования выполнения потоковых приложений при решении задач инженерного анализа в распределенных вычислительных средах».

В рамках работы были опубликованы 2 статьи на международной конференции:

1. Radchenko G., Lyzhin I., Nepovinnyh E. Implementation and Evaluation of the PO-HEFT Problem-Oriented Workflow Scheduling Algorithm for Cloud Environments. // Supercomputing. RuSCDays, Communications in Computer and Information Science, Springer, 2016. – Vol. 687. – P. 91-105.
2. Радченко Г.И., Лыжин И.А., Неповинных Е.А. Имплементация и сравнительное тестирование алгоритма проблемно-ориентированного планирования потоковых приложений в облачных средах PO-HEFT. // Суперкомпьютерные дни в России: Труды международной конференции. М: Московский государственный университет им. М.В. Ломоносова, 2016. – С. 165‑179.

В ходе работы была выполнена одна апробация:

1. Радченко Г.И., Лыжин И.А., Неповинных Е.А. Имплементация и сравнительное тестирование алгоритма проблемно-ориентированного планирования потоковых приложений в облачных средах PO-HEFT. Международная конференция «Суперкомпьютерные дни в России», (Москва, 26-27 сентября 2016 г.).

# **ЛИТЕРАТУРА**

1. Bala A., Chana I. A Survey of Various Workflow Scheduling Algorithms in Cloud Environment. // International Journal of Computer Applications, 2011. – P. 26-30.
2. Bittencourt L.F., Sakellariou R., Madeira E.R.M. DAG Scheduling Using a Lookahead Variant of the Heterogeneous Earliest Finish Time Algorithm. // Parallel, Distributed and Network-Based Processing (PDP), 2010. – P. 27-34.
3. Bozdag D., Catalyurek U., Ozguner F. A task duplication based bottom-up scheduling algorithm for heterogeneous environments. // IPDPS'06 Proceedings of the 20th international conference on Parallel and distributed processing, 2006. – P. 160-160.
4. Bozdag D., Ozguner F., Ekici E., Catalyurek U. A task duplication based scheduling algorithm using partial schedules. // International Conference on Parallel Processing (ICPP'05), 2005. – P. 630-637.
5. Butakov N., Nasonov D. Co-evolutional genetic algorithm for workflow scheduling in heterogeneous distributed environment. // 8th International Conference on Application of Information and Communication Technologies (AICT) Conference Proceedings, 2014. – P. 1-5.
6. Chen W., Deelman E. WorkflowSim: A Toolkit for Simulating Scientific Workflows in Distributed Environments. // E-Science, 2012. – P. 1-8.
7. Cirou B., Jeannot E. Triplet: a Clustering Scheduling Algorithm for Heterogeneous Systems. // Parallel Processing Workshops, 2001. – P. 231-236.
8. Da Silva R., Juve G., Deelman E., Livny M. Online Task Resource Consumption Prediction for Scientific Workflows. // Parallel Processing Letters, 2015. – Vol. 25. – No. 3. – P. 10-35.
9. Król D., Da Silva R., Deelman E., Lynch V. Workflow Performance Profiles: Development and Analysis. // Euro-Par 2016: Parallel Processing Workshops, 2017. – P. 108-120.
10. Davis P.K., Henninger A.E. Analysis, Analysis Practices, and Implications for Modeling and Simulation. – Rand Corporation, 2007. – 54 p.
11. Deelman E., Gannon D., Shields M., Taylor I. Workflows and e-Science: An overview of workflow system features and capabilities. // Future Generation Computer Systems, 2009. – Vol. 25. – No. 5. – P. 528-540.
12. Fotohi R., Effatparvar M. CHS: A Cluster based Job Scheduling algorithm for grid computing. // I.J. Information Technology and Computer Science, 2013. – P. 70-77.
13. Hagras T., Janecek J. A simple scheduling heuristic for heterogeneous computing environments. // Proceedings Second International Symposium on Parallel and Distributed Computing, 2003. – P. 104-110.
14. Ilavarasan E., Thambidurai P., Mahilmannan R. High Performance Task Scheduling Algorithm for Heterogeneous Computing System. // In: Distributed and Parallel Computing. Springer Berlin Heidelberg, 2005. – P. 193-203.
15. Ilavarasan E., Thambidurai P., Mahilmannan R. Performance Effective Task Scheduling Algorithm for Heterogeneous. // The 4th International Symposium on Parallel and Distributed Computing (ISPDC'05), 2005. – P. 28-38.
16. Kannan R., Raihan U.R., Jin H., Balasundaram S.R. Managing and Processing Big Data in Cloud Computing. // IGI Global, 2016. – 307 p.
17. Lewis T.G., El-Rewini H. Scheduling Parallel Program Tasks onto Arbitrary Target Machines. // Journal of Parallel and Distributed Computing, 1990. – Vol. 9. – No. 2. – P. 138-153.
18. Lin W.M., Gu Q. An Efficient Clustering-Based Task Scheduling Algorithm for Parallel Programs with Task Duplication. // Journal of Information Science and Engineering, 2007. – Vol. 23. – No. 2. – P. 589-604.
19. Nepovinnykh E.A., Radchenko G.I. Problem-Oriented Scheduling of Cloud Applications: PO-HEFT Algorithm Case Study. // Proceedings of the 39th International Convention on Information and Communication Technology, Electronics and Microelectronics, 2016. – P. 196-201.
20. Omara F.A., Arafa M.M. Genetic algorithms for task scheduling problem // Journal of Parallel and Distributed Computing, 2010. – Vol. 70. – No. 1. – P. 13-22.
21. Rajesh K.A. Optimization: Algorithms and Applications. // Chapman and Hall/CRC, 2015. – 466 p.
22. Ranaweera S., Agrawal D.P. A task duplication based scheduling algorithm for heterogeneous systems. // Proceedings 14th International Parallel and Distributed Processing Symposium, 2000. – P. 445-450.
23. Synthetic Workflows. [Электронный ресурс] URL: https://confluence.pegasus.isi.edu/display/pegasus/WorkflowGenerator (дата обращения: 20.04.2016).
24. Topcuoglu H., Hariri S., Wu M. Performance-Effective and Low-Complexity Task Scheduling for Heterogeneous Computing. // IEEE Transactions on Parallel and Distributed Systems, 2002. – Vol. 13. – No. 3. P. 260-274.
25. Ullman J.D. NP-complete scheduling problems. // Journal of Computer and System Sciences, 1975. – Vol. 10. – No. 3. – P. 384-393.
26. Wu Q., Datla V.V. On Performance Modeling and Prediction in Support of Scientific Workflow Optimization. // IEEE World Congress on Service, 2011. – P. 161-168.
27. Документация CloudSim. [Электронный ресурс] URL: http://www.cloudbus.org/cloudsim/ (дата обращения: 20.03.2016).
28. Радченко Г.И. Модель проблемно-ориентированной облачной вычислительной среды. // Труды ИСП РАН, 2015. – Т. 27. – № 6. – P. 275‑284.

# ПРИЛОЖЕНИЯ

Приложение 1

Спецификация вариантов использования

|  |
| --- |
| *UseCase:*Спланировать поток работ |
| *ID:* 1 |
| *Аннотация:* назначить каждой задаче вычислительный узел, на котором она будет выполняться |
| *Главные актеры:* 1 |
| *Второстепенные актеры:* - |
| *Предусловия:* - |
| *Основной поток:*   1. Workflow платформа предоставляет схему потока работ и запрашивает план его выполнения. 2. В соответствии с заданным алгоритмом назначить каждой задаче вычислительный узел. При необходимости воспользоваться системой прогнозирования. 3. Вернуть Workflow платформе построенный план выполнения. |
| *Постусловия:* - |
| *Альтернативные потоки:* - |
|  |
| *UseCase:*Спрогнозировать время выполнения задачи |
| *ID:* 2 |
| *Аннотация:* оценить время выполнения задачи на конкретных входных данных и конкретном вычислительном узле |
| *Главные актеры:* 1 |
| *Второстепенные актеры:* - |
| *Предусловия:* - |
| *Основной поток:*   1. Workflow платформа запрашивает оценку времени выполнения конкретной задачи на конкретных размерах входных данных. 2. В соответствии с заданным алгоритмом и информацией, накопленной за время предыдущего использования системы, оценить время выполнения. 3. Вернуть полученную оценку. |
| *Постусловия:* - |
| *Альтернативные потоки:* - |
|  |
| *UseCase:*Добавить информацию о времени выполнения задачи |
| *ID:* 3 |
| *Аннотация:* добавить в базу данных новую информацию о задаче, чтобы в будущем более точно предсказывать время ее выполнения |
| *Главные актеры:* 2 |
| *Второстепенные актеры:* - |
| *Предусловия:* - |
| *Основной поток:*   1. Профилировщик предоставляет информацию о запуске задачи и реальном времени ее выполнения, а также о загрузке CPU, потреблении памяти и другую полезную информацию. 2. Сохранить полученную информацию в базе данных. |
| *Постусловия:* - |
| *Альтернативные потоки:* - |

Приложение 2

Листинг алгоритма планирования PO-HEFT

*/\*\*  
 \* Алгоритм PO-HEFT  
 \*/***public class** POHEFTPlanningAlgorithm **extends** BasePlanningAlgorithm {  
  
 **private** Map<Task, Map<CondorVM, Double>> **computationCosts**;  
 **private** Map<Task, Map<Task, Double>> **transferCosts**;  
 **private** Map<Task, Double> **rank**;  
 **private** Map<CondorVM, List<Event>> **schedules**;  
 **private** Map<Task, Double> **earliestFinishTimes**;  
 **private double averageBandwidth**;  
  
 **private** WorkflowDbHelper **dbHelper** = **new** WorkflowDbHelper(**"jdbc:postgresql://localhost:5432/workflow"**, **"WorkflowUser"**, **""**);  
  
 **private class** Event {  
  
 **public double start**;  
 **public double finish**;  
  
 **public** Event(**double** start, **double** finish) {  
 **this**.**start** = start;  
 **this**.**finish** = finish;  
 }  
 }  
  
 **private class** TaskRank **implements** Comparable<TaskRank> {  
  
 **public** Task **task**;  
 **public** Double **rank**;  
  
 **public** TaskRank(Task task, Double rank) {  
 **this**.**task** = task;  
 **this**.**rank** = rank;  
 }  
  
 @Override  
 **public int** compareTo(TaskRank o) {  
 **return** o.**rank**.compareTo(**rank**);  
 }  
 }  
  
 **public** POHEFTPlanningAlgorithm() {  
 **computationCosts** = **new** HashMap<>();  
 **transferCosts** = **new** HashMap<>();  
 **rank** = **new** HashMap<>();  
 **earliestFinishTimes** = **new** HashMap<>();  
 **schedules** = **new** HashMap<>();  
 }  
  
 @Override  
 **public void** run() {  
 Log.*printLine*(**"POHEFT planner running with "** + getTaskList().size()  
 + **" tasks."**);  
  
 **averageBandwidth** = calculateAverageBandwidth();  
  
 **for** (Object vmObject : getVmList()) {  
 CondorVM vm = (CondorVM) vmObject;  
 **schedules**.put(vm, **new** ArrayList<>());  
 }  
  
 *// Прогнозирование* predictAll();  
  
 *// Вычисление приоритетов* calculateComputationCosts();  
 calculateTransferCosts();  
 calculateRanks();  
  
 *// Назначение на ресурсы* allocateTasks();  
 }  
  
 */\*\*  
 \* Проставляет у каждой задачи оценку времени выполнения и объема выходных данных  
 \*/* **private void** predictAll() {  
 **for** (Task task : getTopSorted(getTaskList())) {  
 **double** inputSize = 0;  
 **for**(Task parent : task.getParentList())  
 inputSize += parent.getPredictedOutputSize();  
 Map<String, Double> options = **new** HashMap<>();  
 options.put(**"insize"**, inputSize);  
 task.setPredictedTaskLength(predictTaskLength(task.getType(), options));  
 task.setPredictedOutputSize(predictOutputSize(task.getType(), options));  
 }  
 }  
  
 **private long** predictTaskLength(String taskName, Map<String, Double> options) {  
 **try** {  
 List<TaskExecution> taskExecutions = **dbHelper**.getExecutionsByTaskName(taskName);  
 **return** (**long**)Predictor.*predictRuntime*(options, taskExecutions);  
 } **catch** (Exception ex) {  
 System.***err***.println(ex.getMessage());  
 ex.printStackTrace();  
 **return** 0;  
 }  
 }  
  
 **private double** predictOutputSize(String taskName, Map<String, Double> options) {  
 **try** {  
 List<TaskExecution> taskExecutions = **dbHelper**.getExecutionsByTaskName(taskName);  
 **return** Predictor.*predictOutsize*(options, taskExecutions);  
 } **catch** (Exception ex) {  
 System.***err***.println(ex.getMessage());  
 ex.printStackTrace();  
 **return** 0;  
 }  
 }  
  
 */\*\*  
 \* Упорядочивает задачи в порядке топологической сортировки  
 \*/* **private** List<Task> getTopSorted(List<Task> taskList) {  
 List<Task> topSortedTasks = **new** LinkedList<>();  
 Map<Task, Integer> degree = **new** HashMap<>();  
 **for** (Task task : taskList)  
 degree.put(task, task.getParentList().size());  
 **for**(**int** i=0; i<taskList.size(); ++i) {  
 **for**(Task task : degree.keySet())  
 **if**(degree.get(task)==0) {  
 topSortedTasks.add(task);  
 **for** (Task childTask : task.getChildList())  
 degree.put(childTask, degree.get(childTask)-1);  
 degree.remove(task);  
 **break**;  
 }  
 }  
 **return** topSortedTasks;  
 }  
  
 */\*\*  
 \* Вычисляет среднюю пропускную способность среди доступных ВМ  
 \*/* **private double** calculateAverageBandwidth() {  
 **double** avg = 0.0;  
 **for** (Object vmObject : getVmList()) {  
 CondorVM vm = (CondorVM) vmObject;  
 avg += vm.getBw();  
 }  
 **return** avg / getVmList().size();  
 }  
  
 */\*\*  
 \* Вычисляет время выполнения задачи на ВМ  
 \*/* **private void** calculateComputationCosts() {  
 **for** (Task task : getTaskList()) {  
 Map<CondorVM, Double> costsVm = **new** HashMap<>();  
 **for** (Object vmObject : getVmList()) {  
 CondorVM vm = (CondorVM) vmObject;  
 **if** (vm.getNumberOfPes() < task.getNumberOfPes()) {  
 costsVm.put(vm, Double.***MAX\_VALUE***);  
 } **else** {  
 costsVm.put(vm,  
 task.getPredictedTaskLength() / vm.getMips());  
 }  
 }  
 **computationCosts**.put(task, costsVm);  
 }  
 }  
  
  
 */\*\*  
 \* Вычисляет время передачи данных между задачами  
 \*/* **private void** calculateTransferCosts() {  
 *// Initializing the matrix* **for** (Task task1 : getTaskList()) {  
 Map<Task, Double> taskTransferCosts = **new** HashMap<>();  
 **for** (Task task2 : getTaskList()) {  
 taskTransferCosts.put(task2, 0.0);  
 }  
 **transferCosts**.put(task1, taskTransferCosts);  
 }  
  
 *// Calculating the actual values* **for** (Task parent : getTaskList()) {  
 **for** (Task child : parent.getChildList()) {  
 **transferCosts**.get(parent).put(child,  
 calculateTransferCost(parent, child));  
 }  
 }  
 }  
  
 */\*\*  
 \* Вычисляет время передачи данных между двумя задачами  
 \*/* **private double** calculateTransferCost(Task parent, Task child) {  
 List<FileItem> parentFiles = parent.getFileList();  
 List<FileItem> childFiles = child.getFileList();  
  
 **double** acc = 0.0;  
  
 **for** (FileItem parentFile : parentFiles) {  
 **if** (parentFile.getType() != Parameters.FileType.***OUTPUT***) {  
 **continue**;  
 }  
  
 **for** (FileItem childFile : childFiles) {  
 **if** (childFile.getType() == Parameters.FileType.***INPUT*** && childFile.getName().equals(parentFile.getName())) {  
 acc += childFile.getPredictedSize();  
 **break**;  
 }  
 }  
 }  
  
 acc = acc / Consts.*MILLION*;  
 **return** acc \* 8 / **averageBandwidth**;  
 }  
  
 */\*\*  
 \* Вычисление рангов  
 \*/* **private void** calculateRanks() {  
 **for** (Task task : getTaskList()) {  
 calculateRank(task);  
 }  
 }  
  
 */\*\*  
 \* Вычисление ранга для задачи  
 \*/* **private double** calculateRank(Task task) {  
 **if** (**rank**.containsKey(task)) {  
 **return rank**.get(task);  
 }  
  
 **double** averageComputationCost = 0.0;  
  
 **for** (Double cost : **computationCosts**.get(task).values()) {  
 averageComputationCost += cost;  
 }  
  
 averageComputationCost /= **computationCosts**.get(task).size();  
  
 **double** max = 0.0;  
 **for** (Task child : task.getChildList()) {  
 **double** childCost = **transferCosts**.get(task).get(child)  
 + calculateRank(child);  
 max = Math.*max*(max, childCost);  
 }  
  
 **rank**.put(task, averageComputationCost + max);  
  
 **return rank**.get(task);  
 }  
  
 */\*\*  
 \* Назначение задач на виртуальные машины  
 \*/* **private void** allocateTasks() {  
 List<TaskRank> taskRank = **new** ArrayList<>();  
 **for** (Task task : **rank**.keySet()) {  
 taskRank.add(**new** TaskRank(task, **rank**.get(task)));  
 }  
  
 Collections.*sort*(taskRank);  
 **for** (TaskRank rank : taskRank) {  
 allocateTask(rank.**task**);  
 }  
  
 }  
  
 */\*\*  
 \* Назначение задачи на самый лучший ресурс  
 \*/* **private void** allocateTask(Task task) {  
 CondorVM chosenVM = **null**;  
 **double** earliestFinishTime = Double.***MAX\_VALUE***;  
 **double** bestReadyTime = 0.0;  
 **double** finishTime;  
  
 **for** (Object vmObject : getVmList()) {  
 CondorVM vm = (CondorVM) vmObject;  
 **double** minReadyTime = 0.0;  
  
 **for** (Task parent : task.getParentList()) {  
 **double** readyTime = **earliestFinishTimes**.get(parent);  
 **if** (parent.getVmId() != vm.getId()) {  
 readyTime += **transferCosts**.get(parent).get(task);  
 }  
 minReadyTime = Math.*max*(minReadyTime, readyTime);  
 }  
  
 finishTime = findFinishTime(task, vm, minReadyTime, **false**);  
  
 **if** (finishTime < earliestFinishTime) {  
 bestReadyTime = minReadyTime;  
 earliestFinishTime = finishTime;  
 chosenVM = vm;  
 }  
 }  
  
 findFinishTime(task, chosenVM, bestReadyTime, **true**);  
 **earliestFinishTimes**.put(task, earliestFinishTime);  
  
 task.setVmId(chosenVM.getId());  
 }  
  
 */\*\*  
 \* Поиск свободного слота для выполнение задачи  
 \*/* **private double** findFinishTime(Task task, CondorVM vm, **double** readyTime,  
 **boolean** occupySlot) {  
 List<Event> sched = **schedules**.get(vm);  
 **double** computationCost = **computationCosts**.get(task).get(vm);  
 **double** start, finish;  
 **int** pos;  
  
 **if** (sched.isEmpty()) {  
 **if** (occupySlot) {  
 sched.add(**new** Event(readyTime, readyTime + computationCost));  
 }  
 **return** readyTime + computationCost;  
 }  
  
 **if** (sched.size() == 1) {  
 **if** (readyTime >= sched.get(0).**finish**) {  
 pos = 1;  
 start = readyTime;  
 } **else if** (readyTime + computationCost <= sched.get(0).**start**) {  
 pos = 0;  
 start = readyTime;  
 } **else** {  
 pos = 1;  
 start = sched.get(0).**finish**;  
 }  
  
 **if** (occupySlot) {  
 sched.add(pos, **new** Event(start, start + computationCost));  
 }  
 **return** start + computationCost;  
 }  
  
 start = Math.*max*(readyTime, sched.get(sched.size() - 1).**finish**);  
 finish = start + computationCost;  
 **int** i = sched.size() - 1;  
 **int** j = sched.size() - 2;  
 pos = i + 1;  
 **while** (j >= 0) {  
 Event current = sched.get(i);  
 Event previous = sched.get(j);  
  
 **if** (readyTime > previous.**finish**) {  
 **if** (readyTime + computationCost <= current.**start**) {  
 start = readyTime;  
 finish = readyTime + computationCost;  
 }  
  
 **break**;  
 }  
 **if** (previous.**finish** + computationCost <= current.**start**) {  
 start = previous.**finish**;  
 finish = previous.**finish** + computationCost;  
 pos = i;  
 }  
 i--;  
 j--;  
 }  
  
 **if** (readyTime + computationCost <= sched.get(0).**start**) {  
 pos = 0;  
 start = readyTime;  
  
 **if** (occupySlot) {  
 sched.add(pos, **new** Event(start, start + computationCost));  
 }  
 **return** start + computationCost;  
 }  
 **if** (occupySlot) {  
 sched.add(pos, **new** Event(start, finish));  
 }  
 **return** finish;  
 }  
}

Приложение 3

Пример описание потока работ в dax-файле.

<?xml version="1.0" encoding="UTF-8"?>

<!-- Cancer Data Standards Repository workflow (Figure 12 in Ramakrishnan and Gannon) -->

<adag name="cadsr" jobCount="4" fileCount="0" childCount="3">

<job id="findClassesInProjects" namespace="cadsr" name="findClassesInProjects" runtime="5" cores="1">

<uses file="projects\_out.dat" link="input" size="10485760"/>

<uses file="classes\_out.dat" link="output" size="15728640"/>

</job>

<job id="findSemanticMetadata" namespace="cadsr" name="findSemanticMetadata" runtime="5" cores="1">

<uses file="projects\_out.dat" link="input" size="10485760"/>

<uses file="classes\_out.dat" link="input" size="15728640"/>

<uses file="metadata\_out.dat" link="output" size="10485760"/>

</job>

<job id="searchLogicConcept" namespace="cadsr" name="searchLogicConcept" runtime="5" cores="1">

<uses file="metadata\_out.dat" link="input" size="10485760"/>

<uses file="search\_out.dat" link="output" size="15728640"/>

</job>

<job id="findProjects" namespace="cadsr" name="findProjects" runtime="5" cores="1">

<uses file="projects\_in.dat" link="input" size="10485760"/>

<uses file="projects\_out.dat" link="output" size="10485760"/>

</job>

<child ref="findClassesInProjects">

<parent ref="findProjects"/>

</child>

<child ref="findSemanticMetadata">

<parent ref="findClassesInProjects"/>

<parent ref="findProjects"/>

</child>

<child ref="searchLogicConcept">

<parent ref="findSemanticMetadata"/>

</child>

</adag>