САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
Математико-механический факультет
Кафедра информатики

Дипломная работа

Распределение наборов неоднородных по размеру заданий в кластерных системах на основе ClassAd механизма

Студент 542 группы
Голубев Александр Юрьевич

Допущен к защите:
Заведующий кафедрой:
д. ф.-м. н., профессор Косовский Н. К.

Научный руководитель:
аспирант Вахитов А. Т.

Рецензент:
д. ф.-м. н., профессор Граничин О. Н.

Санкт-Петербург
2010

ST. PETERSBURG STATE UNIVERSITY
Faculty of Mathematics and Mechanics
Chair of Computer Science

Graduate paper

Assignment policy for sets of jobs with highly-variable sizes on cluster systems based on ClassAd mechanism

Student of group 542
Alexander Yurievich Golubev

Admitted to defence:
Head of the Chair:
Dr. of Phys. and Math. Sci., Professor N. K. Kosovskiy

Academic advisor:
Post-graduate student A. T. Vakhitov

Reviewer:
Dr. of Phys. and Math. Sci., Professor O. N. Granichin

St. Petersburg
2010

Оглавление

1. Введение. 4
1.1. Суть и актуальность работы 4
1.2. Распределенные вычисления и грид 5
1.3. High Throughput Computing и Condor 6
1.4. Механизм ClassAd 7
2. Существующие методы и постановка задачи
2.1. Существующие методы распределения заданий в кластерных системах 8
2.2. Постановка задачи 9
3. Интервальный алгоритм распределения заданий и ClassAd 10
3.1. Архитектура Condor 10
3.1.1. Роли машин и основные демоны 11
3.1.2. ClassAd 14
3.1.3. Распределение заданий в Condor 16
3.2. Математическое обоснование 18
3.3. Реализация в ClassAd системе Condor 19
4. Рабочий пример 21
4.1. Развертывание Condor 21
4.2. Настройка ClassAd механизма 22
4.3. Запуск набора заданий ACO 23
4.4. Анализ полученных данных 24
5. Результаты работы 25
6. Список литературы 26
7. Приложения 27
7.1. График фукции распределения Парето и интервалы 27
7.2. График симуляции средней задержки 27

Введение

Суть и актуальность работы

В настоящее время распределенные вычисления широко используются для решения задач в самых разных сферах, начиная от науки и кончая финансовыми рынками. Физические задачи требуют сложных расчетов, в медицине требуется моделировать лекарства с прогнозируемыми свойствами, расчет химических реакторов позволяет избежать ошибок в испытаниях, аэромеханические расчеты плохо обтекаемых тел, все эти задачи требуют не только большие вычислительные мощности, но и качественную инфраструктуру для надежности и скорости работы [2]. В рамках такой инфраструктуры параллелизация многих из подобных задач подразумевает создание большого числа неоднородных по размеру заданий, рассылающихся по узлам сети.

Согласно [3], одним из самых эффективных алгоритмов распределения неоднородных по размеру заданий является алгоритм, основанный на том, что каждой машине или группе машин посылаются задания размером только из определенного интервала. Проблема заключается в том, что невозможно заранее предугадать размеры наборов заданий, поступающих на выполнения, и, соответственно, задать оптимальные размеры этих интервалов. Имея неоптимальные размеры интервалов, большинство вычислительных мощностей простаивает, сводя преимущества в эффективности упомянутого алгоритма распределения заданий к нулю. В связи с этим была предложена реализация алгоритма по подстройке размеров интервалов к каждому конкретному набору заданий.

Одной из самых развитых и популярных распределенных инфраструктур является система Condor [6], разработанная в университете Wisconsin-Madison. Condor обладает весьма гибкой системой распределения ресурсов, основанной на ClassAd, которая предоставляет широкую вариативность для создания и реализации собственных алгоритмов распределения ресурсов. Механизм ClassAd дает возможность учитывать

любою деталь, начиная от количества оперативной памяти и операционной системы на машине и кончая статистическими данными по средней загрузке процессора. Именно этого и требуют от систем современные алгоритмы распределения ресурсов.

В связи с этим именно система Condor была выбрана для работы по реализации алгоритма распределения заданий на ресурсы. В Cluster классе на математико-механическом факультете была развернута экспериментальная грид инфраструктура под управлением Condor. На основе встроенной в Condor системы ClassAd был реализован упомянутый алгоритм. В качества примера реальной задачи, порождающей большое множество неоднородных по размеру задачий, была взята важная задача биоинформатики по построению оптимальных и субоптимальных эволюционных сценариев между геномами на основе Ant Colony Optimization (ACO).

Распределенные вычисления и грид

На сегодняшний день распределенные вычисления, и в частности грид, являются одной из наиболее развивающихся областей в компьютерной индустрии. Распределенные вычисления – способ решения трудоёмких вычислительных задач с использованием двух и более компьютеров. Что же такое грид? Грид – географически распределенная инфраструктура, объединяющая множество ресурсов разных типов (процессоры, долговременная и оперативная память, хранилища и базы данных, сети), доступ к которым пользователь может получить из любой точки, независимо от места их расположения. Грид предполагает коллективный разделяемый режим доступа к ресурсам и к связанным с ними услугам в рамках глобально распределенных виртуальных организаций, состоящих из предприятий и отдельных специалистов, совместно использующих общие ресурсы. В каждой виртуальной организации имеется своя собственная политика поведения участников, которые должны соблюдать установленные правила. Виртуальная организация может образовываться динамически и иметь

ограниченное время существования. Алгоритмы для грид рассматриваются в [5].

High Throughput Computing и Condor

Для многих экспериментальных ученых качество исследований тесно связано с пропускной способностью их вычислительной среды. Иными словами, большинству ученых более важно решения скольких последовательных массивных задач в месяц или в год они могут ожидать от своей вычислительной среды, чем то, какое количество операций с плавающей точной в секунду(FLOPS) эта среда им может предоставить (что характеризует High Performance Computing (HPC). Вычисления с целью предоставления большего количества мощностей в течение длительного периода время и есть High Throughput Computing (HTC). Ключом к HTC является эффективное управление и эксплуатация всех доступных вычислительных ресурсов. Поскольку вычислительные потребности ученых сейчас могут быть удовлетворены общедоступными процессорами и памятью, высокая эффективность не играет основную роль в среде HTC.

Система Condor – это ПО для поддержки среды HTC, образованной станциями на платформе UNIX и NT. Как и термин HTC, Condor был разработал командой из университета Wisconsin-Madison. Несмотря на то, что Condor может управлять специализированными кластерами из рабочих станций, его ключевое преимущество – способность распределять обычные компьютерные ресурсы, доступные в любой лаборатории или офисе. Иногда еще Condor называют «охотник за свободными станциями» [4] – вместо того чтобы запускать задания на своей машине, пользователь обращается к системе, которая ищет временно свободные машины в сети и запускает на них задания. Когда машина перестанет быть свободной, Condor прерывает выполнение заданий, осуществляет миграцию на другую свободную машину и перезапускает задание на ней с прерванного места. Если нет свободных машин, то задание помещается в очередь и ждет свободных ресурсов. Предусмотренный в Condor механизм управления заданиями предполагает

ведение очередей, составление расписаний, схему приоритетов и классификацию ресурсов, что позволяет пользователю быть в курсе дел о состоянии своих заданий, не заботясь лично об их судьбе.

Механизм ClassAd

Для управления распределения ресурсов в Condor используется рассматриваемый в [6] механизм ClassAd , который предоставляет гибкую структуру по сопоставлению запросов на ресурсы (работа) с предложениями ресурсов (машины). Механизм ClassAd основан на идее рынка с покупателями и продавцами. Первые имеют некоторые условия: чего они хотят купить, сколько готовы потратить и т.д., вторые же имеет условия по тому, за сколько минимум они могут продать свой товар и кому они готовы продавать. Роль продавца играют ресурсы (ядра на компьютере), роль покупателя играет работа. Структура ClassAd представляет из себя набор выражений вида:

```
Name_of_Entity = Expression
```

Эти выражения вычисляются индивидуально для каждого отдельного сопоставления работы и ресурса.

Далее следует рассматривать термины “работа”, “запрос на ресурс” и “задание” как взаимозаменяемые.

Существующие методы и постановка задачи

Существующие методы распределения заданий в кластерных системах

Существует несколько методов распределения в системе с центральным менеджером, у которого есть очередь заданий, и множеством машин, способных эти задания выполнить. Самые известные из них рассматриваются в [3] и [7]:

- **Случайный.** Задания раздаются имещимся в доступе машинам так, что каждое задание имеет вероятность $1/n$ распределения на каждую из n машин. Такое распределение уравнивает ожидаемое количество заданий, в итоге выполненных на каждой машине.
- **Round-Robin.** Задания раздаются по одному на свободную машину по очереди. Это распределение также уравнивает количество заданий, выполненых на каждой машине.
- **Интервальный.** Каждой машине или группе машин достается определенный интервал размеров заданий, которые они имеют право брать на исполнение. Этот метод позволяет маленьким по размеру заданиям не застрять за спиной больших заданий.
- **Динамический.** Каждое следующее задание назначается машине с наименьшим ожидаемым количеством оставшейся работы. С точки зрения заданий, это распределение оптимально.

Рассматриваемый в этой работе алгоритм является прямым наследником и спецификацией интервального алгоритма и называется Size-Based Task Assignment with Equal load (SITA-E). Целью ставится предоставить механизм распределения неоднородного по размеру набора заданий. Предложенный алгоритм использует уравнивание нагрузок для этого.

Согласно [3] случайный и Round-Robin алгоритмы неэффективны в рассматриваемом случае из-за того, что они уравнивают количество заданий, но никак не общий их размер на каждой машине. Что в случае относительно большого количества как больших, так и маленьких заданий приводит к

огромной разнице загрузки между машинами, и как следствие увеличивает общее время исполнения. Согласно авторам [3] динамический метод значительно лучше проявляется себя, чем предыдущие два, однако хуже интервального в случае большого разброса размеров заданий. График средней задержки при запуске 400 заданий (столько же будет запущено в примере) в зависимости от коэффициента вариативности размеров этих заданий (используется упомянутое ниже ограниченное распределение Парето) представлен в Приложении 2.

Постановка задачи

При поступлении набора заданий с неоднородными размерами с целью минимизации общего времени исполнения необходимо следить за тем, чтобы машины имели одинаковую или близкою к таковой нагрузку L_i. Для этого нужно оценить всю нагрузку от набора заданий и поделить ее на количество машин n, чтобы выяснить среднюю нагрузку.

От искомого алгоритма ожидается то, чтобы разница реальных нагрузок на машинах и средней нагрузки были невысоки. Для этого вводится целевая функция:

$$M = E\left(\frac{1}{n} \sum_{i=1}^{n} \left| \frac{1}{n} \sum_{j=1}^{n} L_j - L_i \right| \right)$$

Искомый алгоритм должен минимизировать эту функцию. Необходимо реализовать этот алгоритм на универсальной системе распределения заданий, позволяющей использовать интегрировать любые алгоритмы и, в частности, искомый.

Интервальный алгоритм распределения заданий и ClassAd

Архитектура Condor

Пул Condor состоит из одной машины, играющей роль центрального менеджера, и из произвольного количества других машин, которые присоединяются к этому пулу. Концептуально, пул – это набор ресурсов (машин) и запросов на ресурсы (работ). Роль Condor заключается в том, чтобы сопоставлять ожидающие запросы со свободными ресурсами. Каждый участник системы Condor периодически посылает обновления центральному менеджеру, где находится центральный репозитарий со всей информацией о состоянии пула. Время от времени центральный менеджер оценивает текущее состояние пула и пытается сопоставить запросы с соответствующими ресурсами.

Каждый ресурс имеет своего владельца, пользователя, который работает на машине. Этот человек имеет абсолютную власть над своим ресурсом, и Condor старается уменьшить влияние, которое оказывает его работа на пользователя. Пользователь сам устанавливает политику, когда запрос Condor должен быть принят и когда он должен быть отклонен.

Каждый запрос на ресурсы тоже имеет владельца, того, кто отправил работу. Такие пользователи хотят, чтобы им было предоставлено максимальное количество ресурсов. Таким образом, часто интересы владельцем реурсов и владельцев запросов конфликтуют. Работа администратора кластера Condor заключается конфигурировании системы там, чтобы она сама решала эти конфликты, предоставляла компросисные распределения.

Роли машин и основные демоны

Каждая машина в пуле Condor может выполнять множество ролей. Большинство машин играют несколько ролей одновременно. Определенные роли могут быть исполнены только одной машиной в пуле. Далее рассматриваются эти роли:

- **Центральный менеджер.** В пуле может быть только один центральный менеджер. Он собирает всю информацию и сопоставляет ресурсы с запросами на них. Эти два сервиса исполняются двумя разными демонами. Эта машина играет ключевую роль в пуле Condor, поэтому важно, чтобы она была надежна. Если эта машина даст сбой, дальнейшее сопоставление в системе Condor будет невозможно (хотя все текущие уже сопоставленные пары продолжат работать). Центральный менеджер должен иметь хорошую связь с остальными машинами пула, т.к. Они все посылают ему обновления по сети. Все запросы на ресурсы также идут к центральному менеджеру.
- **Исполнитель работ.** Каждая машина в пуле, включая центрального менеджера, может быть настроена так,что она будет или не будет исполнять работу. Важным ресурсом на исполняющих машинах является дисковое пространство, поскольку, если удаленная работа выгружается, то файл записывается на локальный диск исполняемой машины прежде, чем быть отправленным назад к владельцу этой работы. В общем, чем больше ресурсов (оперативная память, swap, частота процессора и т.д.) имеет машина, тем большие запросы на исполнения она может принимать.
- **Отправитель работ.** Каждая машина в пуле, включая центрального менеджера, может отправлять или не отправлять работы на исполнение. Для такой машины требования гораздо жестче, чем для исполнителя. Каждая работа, исполняемая на удаленной машине, генерирует процесс на машине, отправившей эту работу. Таким образом, если запущено множество работ, необходимо достаточное количество swap и/или оперативной памяти. Кроме того, на этой машине хранится код всех

отправленных работ, что накладывает также ограничения на дисковое пространство. Задания поступают в систему Condor с помощью так называемого submit-файла, пример и описание которого можно увидеть в главе “Рабочий пример”.

Основные демоны и их функции:

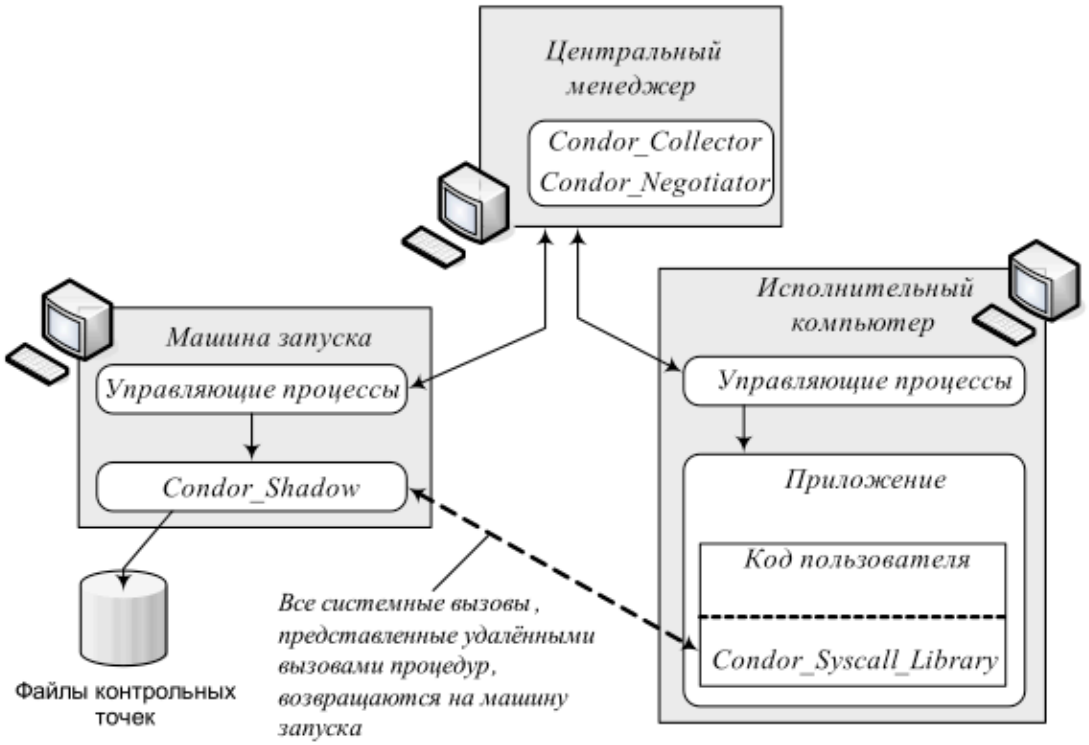
- *condor_master* Этот демон отвечает за поддержание работы всех остальных демонов на каждой машине пула. Он порождает остальные демоны и периодически проверяет наличие обновлений для них. Если находит, то перезапускает этот демон. Если какой-то демон дает сбой, то он посылает письмо администратору и перезапускает этот демон. *condor_master* также поддерживает различные административные команды для старта, остановки и реконфигурирования остальных демонов. Он должен быть запущен на любой машине пула независимо от того, какие функции эта машина выполняет.
- *condor_stard* Этот демон представляет данный ресурс (машину) пулу Condor. Он должен быть запущен на любой машине, способный исполнять работу. Когда этот демон готов выполнить работу, он порождает *condor_starter*.
- *condor_starter* Именно этот демон порождает удаленную работу на данной машине. Он устанавливает среду исполнения и следит за работой, пока она выполняется. Когда работа завершается, он посылает всю необходимую информацию назад на машину, которая отправила эту работу, и выходит.
- *condor_schedd* Этот демон представляет запрос на ресурс в пуле Condor. Он должен быть запущен на любой машине, способной к отправке работ. Когда пользователь отправляет работу, она идет к schedd, где они хранятся в очереди, управляемой schedd. Многие инструменты по просмотру этой очереди и управлению ее (такие как *condor_submit*, *condor_q*, *condor_rm*) должны соединяться с shedd. Он ответственен за требование нужных ресурсов для того, чтобы удовлетворить все

запросы. Когда он находит нужный ресурс для работы, он порождает *condor_shadow* для обслуживания этой работы.

- *condor_shadow* Эта программа исполняется на машине, где данный запрос был отправлен, и играет для него роль менеджера ресурсов.
- *condor_collector* Этот демон ответственен за сбор информации о состоянии пула. Остальные демон периодически посылают ему обновления ClassAd, в которых хранится вся информация о состоянии демонов, ресурсах или запросах. Команда *condor_status* используется для того, чтобы спросить у collector специфическую информацию о различных частях Condor. Кроме того, сами демоны также запрашивают у него важную информацию.
- *condor_negotiator* Этот демон ответственен за все сопоставления внутри системы Condor. Периодически, negotiator начинает цикл, в котором он запрашивает collector о текущем состоянии пула. Он связывается с каждым schedd, имеющим ожидающие запросы в приоритетном порядке, и пытается сопоставить доступные ресурсы с этими запросами. Negotiator заботится о балансе приоритетов пользователей на использование ресурсов (у людей с большими ресурсами приоритет понижается, с меньшими повышается). Он может отобрать ресурс у работы пользователя с низким приоритетом и отдать этот ресурс работе пользователя с высоким приоритетом.

Подробнее про Condor и про представленный далее ClassAd механизм можно прочитать в [6].

На схеме ниже изображены все три роли, которые могут играть машины и необходимые для этого демоны:



ClassAd

ClassAd – это множество выражений с уникальными названиями. Каждое такое выражение с именем называется атрибутом. Выражения ClassAd очень похожи на выражения на языке C, и состоят из литер, подвыражений и ссылок на атрибуты, соединенных операторами и функциями. Разница между ClassAd выражениями и выражениями в C заключается в том, что первые используются в гораздо более динамичной среде. Например, выражение, взятое из ClassAd машины, может ссылаться на атрибут из ClassAd работы. Значение и тип атрибута неизвестны до тех пор, пока выражение не будет вычислено при сопоставлении ClassAd работы и машины. Так в примере, приведенном ниже, `TARGET.Owner` обращается к атрибуту ClassAd работы с названием Owner.

```
MyType          = "Machine"
TargetType      = "Job"
Machine         = "froth.cs.wisc.edu"
Arch            = "INTEL"
OpSys           = "SOLARIS251"
Disk            = 35882
Memory          = 128
KeyboardIdle    = 173
LoadAvg         = 0.1000
Requirements    = TARGET.Owner=="smith" || LoadAvg<=0.3 && \
                  KeyboardIdle>15*60
```

ClassAd выражения справляются с такими неопределенностями, определяя поведение всех своих операторов для любых операндов. Такая функциональность осуществляется с помощью двух различных значений: `UNDEFINED` и `ERROR`. Например, оператор умножения, который обычно работает только с числами, имеет заданное поведение для тех значений, которые умножаться не могут. Так, выражение `10 * "a string"` выдаст значение `ERROR`. Большинство операторов выдает в качестве результата `ERROR`, если хотя бы один из операндов `ERROR`. Аналогичное поведение показывают операторы и для значение `UNDEFINED`.

Поскольку выражение высчитываются в контексте двух ClassAd, существует возможность возникновения двусмысленности в пространстве имен. Следующие правила определяют семантику ссылки атрибута, вызванной ClassAd A и высчитывающейся в контексте другого ClassAd B:

1. Если у ссылки есть префикс:
   - Если префикс `MY.`, атрибут ищется в ClassAd A. Если этот атрибут не находится в A, то значение ссылки становится `UNDEFINED`. Если находится, то значие ссылки становится равным значению выражения атрибута с этим именем.

- Так же, если префикс `TARGET.`, атрибут ищется в ClassAd B. Если не найдет, `UNDEFINED`, иначе присваивается значение выражения атрибута с таким именем.

2. Если у ссылки префикса нет:
   - Если атрибут определен в A, значение ссылки становится равным значению выражения атрибута A.
   - Иначе, если атрибут определен в B, значение ссылки становится равным значению выражения атрибута B.
   - Иначе, если атрибут определен в окружении ClassAd, значение возвращается из этого окружения. Это окружение следует отличать от окружения Unix. В настоящий момент в окружении ClassAd всего один атрибут `CurrentTime`, который высчитывает целое значение, возвращаемое системным вызовом `time(2)`.
   - Иначе, значение становится равным `UNDEFINED`.
3. Наконец, если ссылка обращается в выражению, которое само в процессе вычисления, возникает круговая зависимость и значение становится равным `ERROR`.

Распределение заданий в Condor

В Condor периодически (период можно задать, стандартно 1 минута) демон *condor_negotiator* запускает цикл по сопоставлению работ с ресурсами, способными эти работы исполнить.

Во время этого цикла выполняется следующая последовательность действий:

1. Построение списка всех возможных ресурсов, независимо от их текущего состояния.
2. Получение списка отправляющих работы всего пула.
3. Сортирование списка всех отправляющих работы по приоритетам (в рассматриваемом случае работы отправляются только от одного пользователя).

4. Производить итерации до тех пор, пока не останется либо ресурсов, либо работ.

   Для каждого отправляющего работу от его демона *condor_schedd* получается список работ. Они располагаются по приоритетам (в рассматриваемом случае приоритеты равны, и порядок от старых к новым). Начиная с первой по порядку, для каждой работы:

   - Для каждой машине в пуле, которая может выполнять работу:
     a. Если `machine.requirements` вычисляется равным `False` или `job.requirements` вычисляется равным `False`, пропустить машину.
     b. Если машина в состоянии Claimed, но не выполняет работу, пропустить машину.
     c. Если машина не исполняет никакой работы, добавить машину в список потенциальных сопоставлений с причиной No Preemption.
     d. Если машина исполняет работу:
        - Если `machine.RANK` у рассматриваемой работы лучше, чем он же у текущей, добавить машину в список потенциальных сопоставлений с причиной Rank.
        - Если приоритет отправляющего рассматриваемую работу выше, чем он же у текущей работы и `PREEMPTION_REQUIREMNTS` равен `True`, а также `machine.RANK` рассматриваемой работы не хуже его же текущей работы, добавить машину в список потенциальных сопоставлений с причиной Priority.
   - Машины списка потенциальных сопоставлений сортируютсяпо порядку по критериям `NEGOTIATOR_PRE_JOB_RANK`, `job.RANK`, `NEGOTIATOR_POST_JOB_RANK`, причина (No Preemtion, Rank, Priority), `PREEMPTION_RANK`.
   - Работа достается первой машине из списка. После этого машина удаляется из списка ресурсов до конца цикла.

Математическое обоснование

Рассматривается следующая модель распределенной системы. Имеется n машин, каждая из которых обладает одинаковой мощностью, плюс один менеджер, управляющий распределением ресурсов. На менеджер поступает набор заданий различных размеров. Математическое распределение размеров заданий характеризуется как ограниченное распределение Парето $B(k, p, a)$, где k – наименьший возможный размер задания, p – наибольший возможный размер задания, и a >1 задает уровень вариативности размеров заданий. Причем, чем ближе a к 1, тем больше эта вариативность. Тогда функция распределения и его плотность следующие:

$$F(x) = \frac{1 - k^a x^{-a}}{1 - (k/p)^a}, k \le x \le p$$

(См. пример графика функции распределения в Приложении 1)

$$f(x) = \frac{a k^a x^{-a-1}}{1 - (k/p)^a}, k \le x \le p$$

Каждой из n машин присваивается свой определенный интервал размеров заданий, который они могут выполнять. В интересах достижения баланса нагрузок необходимо правильным образом подобрать эти интервалы. Для этого надо выбрать набор границ интервалов x_i , $i = 0...n$, где $k = x_0 <$ $< x_1 < \ldots < x_n = p$, следующим образом:

$$\int_{x_0=k}^{x_1} x\,dF(x) = \int_{x_1}^{x_2} x\,dF(x) = \ldots = \int_{x_{n-1}}^{x_n = p} x\,dF(x) = \frac{1}{n}\int_{k}^{p} x\,dF(x)$$

Каждая i-тая машина получает интервал от x_{i-1} до x_i . Стоит отметить, что подобным образом целевая функция M обращается в 0, т. к. средняя нагрузка приравнивается к нагрузке всех машин. Решая данное уравнение, получаем (т. к. a >1):

$$x_1^{-a+1} - x_0^{-a+1} = x_2^{-a+1} - x_1^{-a+1} = ... = x_n^{-a+1} - x_{n-1}^{-a+1} = \frac{p^{-a+1} - k^{-a+1}}{n}$$

Отсюда получаем значения границ интервалов:

$$x_i = \sqrt[-a+1]{\frac{ip^{-a+1}}{n} + \frac{(n-i)k^{-a+1}}{n}}, i = 0..n$$

(См. пример значений границ интервалов в Приложении 1)

Имея набор заданий $\{X_t\}$, $t=1...m$, приравниваем k к размеру наименьшего задания, p к размеру наибольшего задания. Значение параметра a можно получить, посчитав математическое ожидание размера имеющихся заданий и приравняв его к теоретическому математическому ожиданию:

$$E(X_t) = \frac{1}{m}\sum_{t=1}^{m} X_t \qquad E(X) = \frac{kp(p^{a-1} - k^{a-1})}{p^a - k^a} \cdot \frac{a}{a-1}$$

Согласно исследованиям в [1] при достаточно больших значениях m (>1000)вычисленное значение a будет максимально соответствовать данному распределению. И значения получившихся границ интервалов сбалансируют нагрузку.

Реализация в ClassAd системе Condor

На центральный менеджер кластера поступает набор заданий. Необходимо с помощью ClassAd сделать так, чтобы эти задания успешно выполнились и загрузка машин была уравнена с помощью разбиения заданий на интервалы по размеру, каждый интервал отдается одной машине. Любая информация относительно заданий должна быть помещена в ClassAd работы. Информация относительно размера пула и того, какие задания может выполнять определенная машина должны быть указаны в ClassAd машины. За этими высказываниями стоит логика, основанная на том, что все динамически меняющиеся несистемные параметры должны быть указаны в

ClassAd работы, т. к. пользователь может влиять на ClassAd машины только с помощью редактирования конфигурационных файлов, загрузка информации из которых происходит в процессе запуска и/или конфигурирования системы. Это означает, что необходимо каждый раз перезапускать всю систему для изменения ClassAd машин. Что неоптимально.

С другой стороны, в ClassAd работы поместить можно информацию, общую для всего набора заданий, и только ее, поскольку наборы рассматриваемых заданий велики, и запись в submit-файл данных для каждой отдельной работы займет много времени. Изменения ClassAd как локальных так и глобальных (*condor_collector, condor_negotiator*) демонов также вступают в силу только после перезагрузки системы.

В sumbit-файл набора заданий следует поместить следующее:

```
Min_size_of_job     = k
Max_size_of_job     = p
Pareto_variability  = a
```

В локальный конфигурационный файл *i*-той машины необходимо добавить следующие строки:

```
Powered_Size        = pow(TARGET.ImageSize, 1- \
                    TARGET.Pareto_variability)
Powered_Min_Size    = pow(TARGET.Min_size_of_job, 1- \
                    TARGET.Pareto_variability)
Powered_Max_Size    = pow(TARGET.Max_size_of_job, 1- \
                    TARGET.Pareto_variability)
Requirements        = Powered_Size >= ((i-1)*Powered_Max_Size + \
                    (n-i+1)*Powered_Min_Size)/n && Powered_Size \
                    <= (i*Powered_Max_Size + (n-i) \
                    *Powered_Min_Size)/n
```

`pow(x,y)` - операция возведения x в степень y.

`\` - перенос на следующую строку.

Рабочий пример

Реализация рабочего примера была проделана в Cluster классе на математико-механическом факультете СПбГУ. Работа разделилась на несколько этапов:

- развертывание Condor;
- настройка ClassAd механизма;
- запуск набора заданий ACO и анализ.

Ниже приведено подробное описание проведенных действий.

Развертывание Condor

В Cluster классе математико-механического факультета СПбГУ установлена операционная система openSUSE. Эта операционная система официально не поддерживается системой Condor. Поэтому с сайта университета Wisconsin-Madison www.cs.wisc.edu/condor/ был скачен исходный код версии 7.4.2, которая на дату написания дипломной работы является текущей стабильной версией. На каждой из 9 рабочих станций класса был создан специальный локальный Condor-user, под которым был сконфигурирован и скомпилирован исходный ход.

Ввиду того, что Condor официально не поддерживается под openSUSE, скомпилированная версия в Cluster классе лишена ряда функциональных возможностей, таких как механизм контрольных точек и удаленных системных вызовов. Также было проведено конфигурирование полученной инфраструктуры и подготовка к комфортной работе пользователей. Таким образом был создан так называемый пул из 9 машин с центральным менеджером на рабочей станции № 1. Все станции, в том числе и центральный менеджер, способны отсылать работу в пул и выполнять любою из работ, находящихся в пуле. Глобальное администрирование способен выполнять только пользователь, управляющий центральным менеджером.

Настройка ClassAd механизма

Каждая рабочая станция имеет имя вида l{номер}. l01 - центральный менеджер. Он не будет исполнять работу, для этого из списка запускаемых при инициализации демонов в конфигурационном файле убирается *condor_stard*. Оставшиеся 8 станций делятся на 4 группы по 2. Каждой группе выдается свой интервал заданий. Для этого в конфигурационные файлы каждой машины было записано следующее:

```
Powered_Size        = pow(TARGET.ImageSize, 1- \
                    TARGET.Pareto_variability)
Powered_Min_Size    = pow(TARGET.Min_size_of_job, 1- \
                    TARGET.Pareto_variability)
Powered_Max_Size    = pow(TARGET.Max_size_of_job, 1- \
                    TARGET.Pareto_variability)
Num_of_interval     = floor (substr(Machine, 8, 1)/2)
Requirements        = Powered_Size >= ((Num_of_interval-1) \
                    *Powered_Max_Size + (5-Num_of_interval) \
                    *Powered_Min_Size)/4 && Powered_Size <= \
                    (Num_of_interval*Powered_Max_Size + (4- \
                    Num_of_interval*Powered_Min_Size)/4
```

`Num_of_interval` задает номер интервала следующим образом:

`Machine` – полное название машины. Компьютеры в Cluster классе математико-механического факультета двухядерные, поэтому название имеет вид `slot1@l02.math.spbu.ru`, здесь рассматривается первое ядро второго компьютера.

`substr(s,x,y)` делает обрезание строчки s начиная с позиции x и длиной, равной y.

`floor` округляет в нижнюю сторону, предварительно переводя любой тип выражения в `Real`.

Запуск набора заданий ACO

На центральный менеджере была создана папка aco, внутри которой находятся все необходимый файлы для запуска построения эволюционных сценариев, а именно: исполняемый файл aco.o, файл начального сценария input.txt и сам submit-файл для Condor, который выглядит следующим образом:

```
executable      = aco.o
arguments       = 10 10 0.9
input           = input.txt
output          = aco_out.$(procid)
universe        = vanilla
queue           10
```

Executable – указывает приложение для запуска.

Arguments – параметры для приложения, указываемые в командной строке.

Input – файл, использующийся вместо стандартного входа.

Output – файл, использующийся вместо стандартного выхода.

Universe – среда окружения, в которой будет выполнено приложение.

Queue – количество раз, которое приложение будет запущено.

Первый и второй аргументы отвечают за качество построения, причем средний ожидаемый размер заданий линейно связан с каждым из аргументов. Кроме того, каждое из заданий (из 10 в примере выше) тоже отличается по размеру от других, в силу выбора путей с вероятностями внутри ACO алгоритма.

Задание, описанное в примере исполняется 1 секунду на одном ядре компьютера в Cluster классе и имеет размер 12,500 байт. Всего согласно ограниченному распределению Парето с границами 10,000 и 10,000,000 и коэффициентом 1.01 (большой разброс размеров заданий) было запущено 400 заданий (см. график этого распределения в Приложении 1):

12,500 байт	- 80 штук	200,000 байт	- 20 штук
25,000 байт	- 160 штук	400,000 байт	- 12 штук
50,000 байт	- 80 штук	800,000 байт	- 4 штуки
100,000 байт	- 40 штук	3,200,000 байт	- 4 штуки

Согласно ClassAd соответствующих работ задания первых трех размеров были распределены на 2 и 3 компьютеры и были выполнены за 180 секунд, задания размерами 100,000b и 200,000b попали на 4 и 5 компьютеры и были выполнены за 160 секунд. Задания размером 400,000b и 800,000b попали на 6 и 7 машины, время исполнения 160 секунд. Наконец, на компьютеры 8 и 9 попали 4 задания размером $3.2*10^6$b. Каждое из 4 ядер забрало по заданию, каждое выполнялось 256 секунд.

При этом общая нагрузка составляет $37.8*10^6$b. Соответственно, средняя нагрузка составляет $9.45*10^6$b. Это позволяет нам посчитать значение функции M для данного распределния заданий на данной системе. $M = 1.675$. В итоге разбалансированность на нашем примере составляет примерно 18 % от средней нагрузки.

Анализ полученных данных

Полученный показатель является весьма неплохим особенно в условиях сильной неоднородности размером заданий. Стоит отметить, что выбранное распределение является оптимальным на данном примере. Быстрее 256 секунд выполниться задача не могла в силу атомарности задания размером $3.2*10^6$b. А, соответственно, на самом деле был достигнут максимальный реальный показатель мощности рассматриваемой распределенной системы.

Несмотря на то, что динамический метод хорошо себя проявляет для случая заданий, не сильно отличающихся по размеру, разбалансированность динамического метода при подобных заданиях может составлять до 50 %. Случайный и Round-Robin алгоритмы сопоставили бы одинаковое количество заданий каждому из 4 классов, независимо от размеров заданий. Процентное соотношение целевой функции к средней нагрузке может составлять до 200 %. Необходимо также учитывать, что увеличение количества заданий и их возможных размеров приводит к улучшению результатов для интервального подхода с балансировкой нагрузки.

Результаты работы

Был рассмотрен интервальный алгоритм распределения заданий и, в частности, его модификация SITA-E, выявлено, что из существующих алгоритмов для случая множеств неоднородных по размеру заданий он является оптимальным.

Подведена математическая база под задачу нахождения границ интервалов, на которые необходимо разделить область размеров задач для балансировки нагрузки на машины.

С помощью механизма ClassAd метод интервальной балансировки нагрузки был перенесен на реальную распределенную систему Condor, основные принципы которой были описаны.

Развернута экспериментальная инфраструктура Condor на математико-механическом факультете СПбГУ с встроенным в нее с помощью ClassAd механизмом балансировки нагрузки, адаптирующимся к каждому конкретному набору заданий.

Проведено тестирование механизма балансировки нагрузки на примере задачи биоинформатики по построению оптимальных и субоптимальных эволюционных сценариев между геномами на основе ACO. Также произведено оценивание скорости работы системы на конкретном примере.

Список литературы

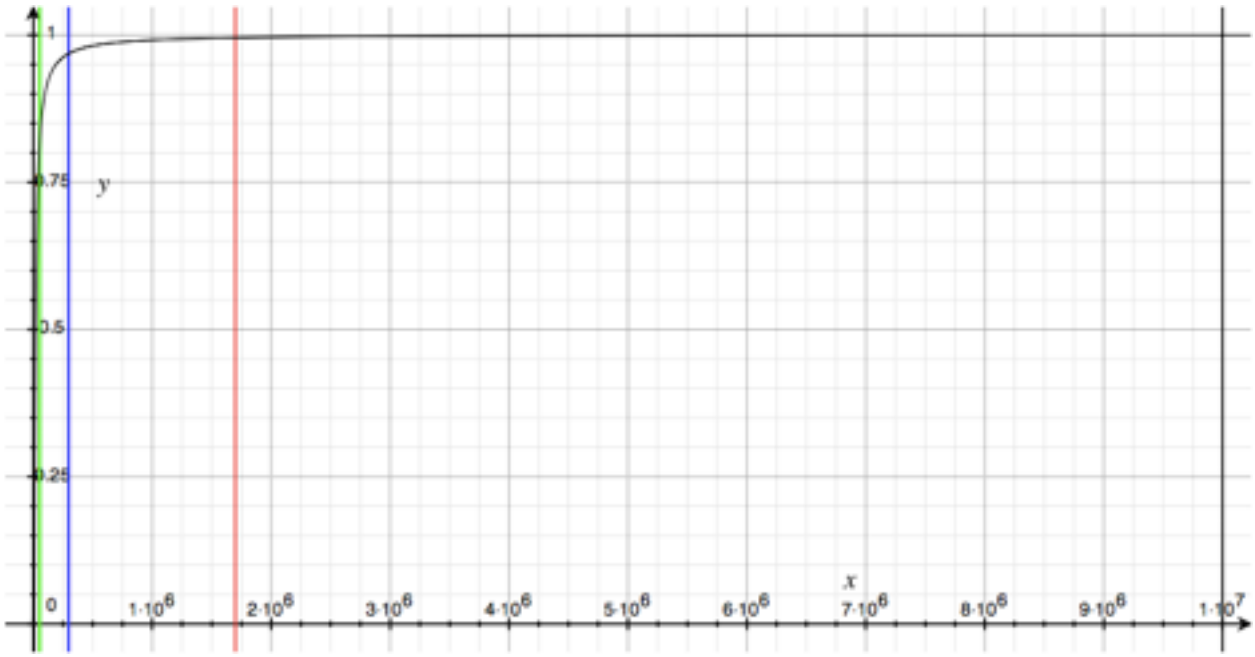
1. Беврани Х., Аничкин К. Оценка параметров распределений с тяжелыми хвостами с помощью эмпирического распределения. МКО, 2005, ч.2, стр. 493-499
2. Воеводин В. В. Решение больших задач в распределенных вычислительных средах // Автоматика и телемеханика, 2005, № 5, стр. 32-45
3. M. Harchol-Balter, M. E. Crovella, C. D. Murta. On Choosing a Task Assignment Policy for a Distributed Server System // In Proceedings of Performance Tools, Lecture Notes in Computer Science, Vol. 1468, pp. 231-242, September 1998
4. M. J. Litzkow, M. Livny, M. W. Mutka. Condor - A Hunter of Idle Workstations // In Proceedings of the 8th International Conference on Distributed Computing Systems, pp. 104-111, 1988
5. F. Dong, S. G. Akl. Scheduling Algorithms for Grid Computing: State of the Art and Open Problems. Technical Report No. 2006-504, January 2006
6. Condor Team, University of Wisconsin-Madison, Condor version 7.4.1 Manual - February 1, 2010
7. M. Harchol-Balter. Task Assignment with Unknown Duration. Journal of the ACM, Vol. 49, Issue 2, pp. 260-288, March 2002

Приложения

1. График функции распределения Парето и интервалы

$k = 10^4$, $p = 10^7$, $a = 1.01$

$n = 4$, x_1 - первая вертикальная линия, x_2 - вторая, x_3 - третья



2. График симуляции средней задержки

$k = 10^3$, $p = 10^{10}$, $n = 8$, $a = alpha$