Программная система для моделирования активности пользователей вычислительного кластера на основе системы ведения очередей SLURM*

А.Н. Сальников[1], А.Н. Бойко[2]

Факультет Вычислительной математики и кибернетики МГУ имени М.В. Ломоносова[1], Черноморский филиал МГУ имени М.В. Ломоносова[2]

Рассматривается система, которая на основе сбора статистики, ведущейся системами ведения очередей, организует имитационную нагрузку на модельный вычислительный кластер, управляемый системой ведения очередей Slurm. В результате, появляется возможность оценить эффективность применяемых политик планирования задач пользователей до применения их на реальном кластере. В работе приводится несколько метрик, использующихся системными администраторами кластера. Предложенный подход опробован на данных с вычислительных кластеров факультета ВМК МГУ и АО «НИКИЭТ».

1. Введение

Современные вычислительные кластеры в большинстве случаев используются коллективно многими пользователями. В отличие от «обычных компьютеров» (рабочих станций пользователей и серверов), пускай даже достаточно производительных, использование ресурсов таких систем происходит в пакетном режиме. Явно присутствует очередь задач пользователей, которая обеспечивается специальным программным обеспечением — системой ведения очередей. Данное программное обеспечение достаточно сложно, как правило состоит из нескольких «демонов» в терминологии UNIX-подобных операционных систем. Некоторыми используемыми системами ведения очередей в России являются: Slurm [1], LoadLeveler [2], СУППЗ [3] (установлено в институтах РАН, в том числе МСЦ РАН).

Демоны запускаются на разных компьютерах: «управляющей машине», «узлах кластера». На управляющей машине, как правило, находится демон, который отвечает за ведение очередей, мониторинг ресурсов, планирование использования ресурсов задачами пользователей. На узлах кластера запускается специальный демон, который по команде от управляющей машины ставит задачи из очереди на выполнение путём запуска соответствующих процессов и смены им прав доступа с прав пользователя root на права обычного пользователя. Также осуществляется снятие задачи со счёта путём выставления соответствующих сигналов процессу, соответствующему запущенной на узле задаче. Обычно демон, запущенный на узле, отслеживает ещё состояние узла кластера и сообщает о нём управляющей машине.

Пользователь обычно попадает на «интерфейсную машину», с которой он может «ставить» задачи в очередь. Интерфейс к системе ведения очередей обычно предоставляется консольной программой, которая по сети взаимодействует с демоном на управляющей машине.

Системные администраторы вычислительного кластера часто сталкиваются с жалобами пользователей на некорректное продвижение их задач через очередь. Как правило, пользователи возмущены чрезмерно большим временем, проведённым задачей в очереди. Большинство современных систем ведения очередей обладают широким набором варьируемых параметров, которые существенно влияют на политику планирования прохождения

*Работа ведётся при частичной поддержке грантов РФФИ: 14-05-00363, 14-07-00037, 14-07-00654.

задач пользователей через очередь. Системные администраторы могут захотеть изменить настройки системы ведения очередей, однако из-за сложности системы и большого потока задач не всегда очевидно, как именно изменения скажутся на группах пользователей и не появятся ли особенно недовольные пользователи, или не приведёт ли это к неэффективному использованию ресурсов и простою дорогостоящего оборудования.

Всё это делает актуальным решение двух задач: *задачи анализа статистики, предоставляемой системами ведения очередей*; *задачи имитационного моделирования прохождения задач через очередь*.

Для решения первой задачи нами предложена программа, которая по статистике в некотором универсальном формате вычисляет различные метрики и отображает их графики. Для решения второй задачи предложено программное обеспечение, которое на основе специальным образом скомпилированной системы ведения очередей Slurm и программы, которая читает файл со статистикой, вычисляет сжатое с варьируемым коэффициентом реальное время в модельное время и ставит некоторые модельные «псевдозадачи» в очередь. В дальнейшем можно производить анализ статистики, разжимая модельное время в реальное.

Однако такой подход связан с некоторыми техническими трудностями. Требуется где-то воспроизвести множество узлов кластера. Очевидно, что построить рядом такой же кластер не представляется возможным, поэтому необходим механизм виртуальных узлов кластера, который предоставляет Slurm. Также возможно, за счёт унификации формата предоставляемой статистики и переноса понятий одной системы ведения очередей на понятия Slurm, моделировать прохождение задач как от разных систем ведения очередей, так и от разных вычислительных кластеров.

Следует отметить, что идея сама по себе не нова и группы коллективов уже решали подобные задачи. В 2011 году на конференции HPC-UA в г. Киеве [4] представлена работа по созданию виртуального кластера, часть представленых в работе метрик реализована в нашей работе. К сожалению, исходный код (вероятно, в силу разовости работы) не представлен в открытом доступе и, возможно, потерян.

Также в 2011 году группой разработчиков из Барселонского суперкомпьютерного центра был представлен симулятор Slurm [5]. Основная идея данного симулятора заключается в том, что задачи реально не отправляются на узлы кластера, даже виртуальные. Вместо этого на управляющей машине в slurmctld производится подмена функций библиотеки libc таким образом, что время для slurmctld меняется скачком на предполагаемое время выполнения задачи на узлах кластера. Мы решили, что использование данного симулятора для нас возможно, однако хотелось более-менее универсального подхода, который позволит легко адаптировать программный код к системе ведения очередей, отличной от Slurm.

2. Описание архитектуры программной системы

Программная система реализована как набор Python-скриптов. Требуется Python версии 2.7. Язык программирования Python был выбран из соображений обеспечения переносимости. Представлено несколько типов скриптов.

1. Группа скриптов (начинаются с `parse_`), которая отвечает за получение статистики от конкретной системы ведения очередей. В текущий момент реализован разбор вывода LoadLeveler и базы данных Slurm.

2. Скрипт отображения статистики (`print_characteristics.py`). Скрипт с применением библиотеки `matplotlib` отображает графики характеристик, вычисляемых по статистике, а также пишет необходимые сведения в текстовый файл.

3. Скрипт, организующий эмуляцию активности пользователя путём постановки модельных заданий в очередь. В текущий момент реализована только постановка в Slurm.

Скрипт называется `run_pseudo_tasks_slurm.py`.

4. Скрипт создания пользователей и групп в системе (кластере-полигоне) для проведения эксперимента. Называется `pseudo_users_and_groups_operations.py`.

Остановимся более подробно на процессе разбора статистики и вопросах, связанных с организацией кластера-полигона, а также запуске задач, моделирующих реальные задачи.

2.1. Извлечение статистики из системы ведения очередей

В данном месте мы столкнулись с несколькими проблемами. Информация о задачах, хранящаяся в разных системах ведения очередей, естественно, разная. Возникла необходимость среди всего множества параметров выбрать некоторое универсальное подмножество. Универсальные параметры мы свели в поля файла формата CSV (Comma-Separated Values), в котором было решено хранить извлечённую статистику. Далее приведены поля данного файла:

1. `id` – идентификатор задачи, под которым он проходил в системе ведения очередей.
2. `name` – имя задачи при постановке в очередь, которое задал пользователь.
3. `time_submit` – дата и время поступления задачи в очередь. Дата в UTC.
4. `time_start` – дата и время отправки задачи на счёт.
5. `time_end` – дата и время снятия задачи со счёта и покидания очереди.
6. `user` – имя пользователя, под которым задача ставилась в очередь или имя псевдопользователя, которое присваивается Python-скриптом при извлечении информации.
7. `group` – имя группы. Семантика аналогична предыдущему полю.
8. `time_limit` – Лимит времени выполнения задачи в минутах, которое выставлял пользователь. Если время было меньше минуты, то сюда пишется 1. Ноль устанавливается в случае, если пользователь не выставил ограничение на длительность своей задачи.
9. `required_cpus` – Этот параметр означает число MPI-процессов в случае запуска MPI-программы.
10. `partition` – раздел в очереди. В идеале, он должен соответствать части кластера, но некоторые системы очередей (например, Sun Grid Engine [6]) часто используют данное понятие для альтернативных способов планирования доступа к одним и тем же ресурсам.
11. `priority` – приоритет задачи, который выставляет пользователь. Автоматическое его определение по базе данных slurm, в текущий момент, нам представляется сложной задачей, поэтому для задач Slurm здесь стоит значение 0.
12. `task_class` – класс, присвоенный задаче. Иногда пользователю предоставляется возможность указать класс своей задачи для выбора применяемых политик планирования. В текущей реализации если класс не указан, то выставляется класс: `pseudo_cluster_default`.
13. `state` – состояние задачи в очереди. Задано отображение состояния, как оно принято в системе ведения очередей на состояние, как оно принято в нашей системе.

14. `other` – список прочих параметров, которые не столь универсальны. Данный список имеет формат `имя_параметра='значение',имя_следующего='значение'`. При запуске моделирующих задач Python-скрипт извлекает оттуда «знакомые» ему параметры.

В текущий момент скрипт разбора поддерживает ключ, который маскирует пользователей, подставляя вместо реальных имён пользователей и групп имена вида `pseudo_cluster_user_100500`.

Одновременно с этим создаются дополнительные файлы с соответствием псевдопользователей и псевдогруп реальным, а также файлы со включённостью псевдопользователей как в реальные группы, так и в псевдогруппы. После разбора статистики можно запустить скрипт, который на тестовом полигоне создаст всех псевдопользователей и псевдогруппы и добавит всех пользователей в нужные группы.

Псевдопользователи необходимы не только из соображений унификации процесса тестирования для разных кластеров, но и для случая, когда статистика берётся за отдалённый период от текущего момента времени, и пользователи и группы за это время могли частично измениться. Также подход с маскированием пользователей позволяет хоть чуть-чуть рассеять опасения системных администраторов о разглашении конфиденциальной информации о функционировании вычислительной системы.

2.2. Организация тестового полигона

Для организации полигона был перекомпилирован Slurm с опцией: `--enable-multiple-slurmd`. Эта опция позволяет в файле конфигурации slurm.conf создавать виртуальные узлы кластера, где на реальном узле можно развернуть множество виртуальных. Каждый виртуальный узел представлен своим экземпляром демона, отвечающего за запуск заданий, который взаимодействует с управляющей машиной по своему порту. (К записи объявления узла добавляются опции `NodeHostname` и `Port`.)

Для разворачивания виртуальных узлов на физических пришлось модифицировать стандартные скрипты запуска демонов Slurm, поскольку они предполагают запуск одного экземпляра slurmd на узле кластера. Пришлось регулярными выражениями извлекать информацию о виртуальных узлах по имени реального узла, для этого пришлось отказаться от принятой в slurm сокращённой записи имён узлов через квадратные скобки.

Поскольку при моделировании вместо реального времени используется модельное, а Slurm при этом работает в реальном времени, то в конфигурационном файле все действующие ограничения на лимиты времени должны быть масштабируемы на коэффициент сжатия, с которым предполагается сжимать модельное время к реальному.

Как было сказано ранее, запуск осуществляется скриптом `run_pseudo_tasks_slurm.py`. Данный скрипт требует себе на вход следующие параметры: путь до .csv файла со статистикой; коэффициент сжатия времени; интервал времени, не чаще которого будут производиться запуски задач в очередь развёрнутого полигона.

Скрипт работает следующим образом. В соотвествии с преобразованым временем выбираются задачи из файла со статистикой, которые должны быть запущены или сняты со счёта в течении указанного в параметрах интервала времени, соответствующие команды отдаются системе ведения очередей. Как именно это происходит описано далее. При помещении задачи в очередь кластера-полигона в имени новой задачи сохраняется идентификатор старой задачи, как она была представлена в файле со статистикой — таким образом сохраняется соответствие задач в очереди реального кластера и кластера-полигона. После проведения всех необходимых операций скрипт засыпает на время, осташееся до начала следующего интервала. Так продолжается до тех пор, пока не будут исчерпаны все необходимые действия со всеми задачами из файла статистики.

Предполагается, из соображений наибольшей честности эмулирования применения политик планирования, что операции с очередью полигона производятся от имени моделиру-

емого пользователя. Это может быть как тот же пользователь, что и на реальной системе, в случае, когда на полигон копируются учётные записи, так и псевдопользователь, полученный в результате процедуры маскирования реального пользователя, описанной ранее. Для обеспечения такого поведения Python-script запускается с правами пользователя root, затем создаётся новый процесс, который с исходным связывается каналом, в порождённом процессе права понижаются до прав нужного пользователя и далее, в зависимости от ситуации, тело процесса подменяется на запуск команды `sbatch` или `scancel`. При этом вывод запущеных команд считывается в процессе-предке и из этого вывода прочитывается идентификатор модельной задачи, который ей присваивается Slurm-ом. В дальнейшем этот идентификатор используется, если нужно по протоколу действий досрочно завершить задачу.

Наконец, можно перейти к тому, как анализируется и отображается статистика.

3. Описание используемых метрик

Имеется набор задач T, каждая из которых запускалась на вычислительном кластере, управляемом системой ведения очередей Slurm. Пусть $U = \{u | \exists task \in T : user(task) = u\}$ — множество всех пользователей, которые ставили задачи из набора задач T, T_u — множество задач из набора задач T, которые были поставлены пользователем $u \in U$.

По этому набору вычисляются числовые характеристики (более подробно описаны далее): «среднее время ожидания в очереди», «средняя заполненость вычислительной системы», «степень разумности запрашиваемого времени исполнения», «количество задач», «среднее отношение времени в очереди к запрошенному времени».

Каждая из характеристик может считаться в трёх вариантах: «по дням», «по каждому пользователю», «в общем». В целом, авторы полагают, что эти метрики открывают простор для анализа работы вычислительной системы. В особенности они полезны для моделирования сценариев вида «что если», когда администратор меняет параметр системы и хочет посмотреть, как бы себя повели существующие задачи в новых условиях.

В программном коде метрики оформлены как подключаемые модули. Каждый желающий может добавить свою метрику записав соответсвующий python файл в каталог `pseudo-cluser/metrics`. В дальнейшем информацию о доступных метриках можно посмотреть указав опцию `--show-available-metrics` и информацию о параметрах определённой метрики:
`--metric METRIC --show-metric-description`.

У каждой задачи $task \in T$ есть несколько параметров, которые представлены в файле со статистикой, нас будут интересовать следующие:

- $t_{start}(task)$ – дата и время, когда задача начала выполнение
- $t_{end}(task)$ – дата и время, когда задача закончила выполнение
- $t_{queue}(task)$ – дата и время, когда задача была поставлена в очередь
- $p(task)$ – запрошенное пользователем количество процессоров для выполнения задачи
- $t_{limit}(task)$ – запрошенное пользвателем ограничение по времени исполнения задачи
- $s(task)$ – статус запуска задачи
- $u(task)$ – пользователь, который поставил задачу

Далее указываются формулы для подсчёта, а также предназначение указанных ранее характеристик. По некоторым из них можно строить график, чтобы более наглядно увидеть распределение характеристики по пользователям или дням. Дабы не загромождать статью Формулы указаны для одного из трёх вариантов: 1.) дни, 2.) пользователи, 3.) общее. Под

вычитанием двух дат будем подразумевать количество минут, которое прошло от уменьшаемой даты до вычитаемой даты.

3.1. Среднее время ожидания в очереди

Среднее время ожидания в очереди для задач пользователя $u \in U$ считается следующим образом:

$$\frac{\sum_{task \in T_u} (t_{start}(task) - t_{queue}(task))}{|T_u|}$$

Эта метрика напрямую выражает «недовольство» пользователей вычислительным кластером — никому не хочется долго ждать, пока задача начнёт выполнение. В идеале, это число должно быть как можно меньше.

3.2. Средняя заполненность вычислительной системы

Пусть d – некая единица измерения времени (например, два часа), указанная в минутах. Также обозначим $len(t_1, t_2) = (t_2 - t_1)/d$, $time_{min} = \min\{t_{start} | task \in T\}$ – время самого раннего старта задачи, $time_{max} = \max\{t_{end} | task \in T\}$ – время самого позднего окончания задачи.

Средняя заполненность R измеряется в процессорах. Она призвана показать, насколько много процессоров в среднем (на определённый квант времени d) было использовано за период времени указанный в параметрах программы, при этом предполагаем набор задач T. Формула для вычисления:

$$R = \frac{\sum_{task \in T} p(task) \cdot len(t_{start}(task), t_{end}(task))}{len(time_{min}, time_{max})}$$

Метрика показывает, насколько «нагружена» вычислительная система оказывается под влиянием приходящих задач, в том числе можно отследить вклад определённого пользователя в некоторые моменты на общую нагрузку системы. Метрика позволяет выявить самых «активных» пользователей с точки зрения заполнения вычислительного кластера.

3.3. Степень разумности запрашиваемого времени исполнения

Для каждого пользователя $u \in U$ считается множество задач, которые успешно выполнились: $SC_u = \{task \in T_u | s(task) = completed\}$. Тогда степень разумности запрашиваемого времени исполнения для данного пользователя вычисляется так:

$$\frac{\sum_{task \in SC_u} \frac{t_{end}(task) - t_{start}(task)}{t_{limit}(task)}}{|SC_u|}$$

На первый взгляд кажется, что данная характеристика не зависит от настроек вычислительной системы, а полностью зависит от пользователей. Но стоит учесть, что на выставление лимита по времени влияет успешность предыдущих запусков программы на кластере. Например, пользователь два раза запустил программу с ограничениями в 15 и 20 минут, обе они не завершились к этому времени. Тогда рядовой пользователь возьмёт времени «с запасом» — 40 минут. Хотя, быть может, хватило бы 25. Такие действия, верные с точки зрения пользователя системы, совершенно неприемлемы для администратора, ведь выставление заведомо больших лимитов времени негативно влияет на этап планирования очерёдности всех задач. В идеальном случае, эта метрика должна быть близка к 1, а количество неуспешных (не попавших в SC_u) задач должно быть как можно меньше. Анализ степени разумности запрашиваемого времени позволит выявить случаи неудачного использования вычислительной системы. После этого можно, к примеру, обсуждать с пользователями «правильность» выставленных ими лимитов времени, и предлагать им другие лимиты основываясь на статистическом анализе их задач.

3.4. Среднее отношение времени в очереди к запрошенному времени

Среднее отношение времени в очереди к запрошенному времени для задач пользователя $u \in U$ считается следующим образом:

$$\frac{\sum_{task \in T_u} \frac{(t_{start}(task) - t_{queue}(task))}{t_{limit}(task)}}{|T_u|}$$

Не каждой задаче одинаково критично находиться в очереди долгое время. Например, если задача сама по себе выполняется 900 минут, то время ожидания в очереди в 15 минут особой роли не сыграет. В то же время задача, которой требуется 1 минута на выполнение, но которая будет стоять в очереди 15 минут, вызовет недовольство пользователя. Эта характеристика — что-то вроде альтернативного взгляда на «среднее время ожидания в очереди», который позволит более тонко регулировать параметры системы.

4. Результаты работы

В рамках данной работы создан проект с открытым исходным кодом. Код программной реализации доступен на сайте GitHub под именем организации *pseudo-cluster* [7].

Также была собрана статистика за декабрь 2013 года с системы Bluegene/P, установленной на факультете ВМК МГУ имени М.В. Ломоносова. Была собрана статистика с вычислительного кластера beta, установленного в АО «НИКИЭТ». Для данного кластера создан виртуальный стенд, на котором исследовалось влияние параметра ограничения длительности задачи по умолчанию (если пользователь длительность не указывает явно) на эффективность планирования ресурсов системы. Всё это оказалось возможным при помощи проекта, предложенного в данной статье.

Авторами отмеченно нежелание системных администраторов вычислительных кластеров делиться информацией о статистике прохождения задач через очередь суперкомпьютера, даже если суперкомпьютер находится в той же организации, в которой работают авторы. В частности, на просьбу предоставить статистику по суперкомпьютеру Ломоносов был получен категорический отказ. Впрочем, авторы надеются, что проект найдёт применение в среде системных администраторов ввиду своей лёгкости в использовании и наглядности, в силу чего не придётся делиться конфиденциальной информацией с кем бы то ни было.

Так-же была отмеченна некоторая тяжеловесность библиотеки `matplotlib`, в следствии чего нами реализована пробная версия отображения графиков средствами библиотеки `pygal` [8].

Литература

1. Сайт системы ведения очередей Slurm (Slurm Workload Manager): URL: `http://slurm.schedmd.com/slurm.html` (дата обращения: 29.11.2014).

2. Kannan S., Roberts M., Mayes P., Brelsford D., Skovira J.F. Workload Management with LoadLeveler // IBM Redbooks, November 2001. pp. 222 ISBN: 9780738422091.

3. А.В. Баранов, Д.С. Ляховец Сравнение качества планирования заданий в системах пакетной обработки SLURM и СУППЗ // Научный сервис в сети Интернет: все грани параллелизма: Труды Международной суперкомпьютерной конференции (23-28 сентября 2013 г., г. Новороссийск). Москва - Изд-во МГУ, 2013. С. 410–414.

4. Голвинський А., Маленко А. Аналіз ефективності планувальників черги задач для суперкомп'ютера з кластерною архітектурою // Труды международной конференции «Высокопроизводительные вычисления» (12-14 октября 2011) HPC-UA'2011 Украина, Киев, 2011. С. 64–69.

5. Lucero A. Simulation of batch scheduling using real production-ready software tools // Ibergrid 2011, 5th Iberian Grid Infrastructure Conference Proceedings, Prtugal 2011. Chapter 1, P. 345–356. ISBN 978-84-9745-884-9.

6. Сайт проекта UNIVA: наследник проекта Sun Grid Engine
URL: `http://www.univa.com/products/grid-engine.php` (дата обращения: 29.11.2014).

7. Сайт проекта pseudo-cluster на хостинге проектов с открытым исходным кодом GitHub: URL: `https://github.com/pseudo-cluster` (дата обращения: 29.11.2014).

8. Сайт проекта отрисовки графиков из Python в svg формате: URL: `http://pygal.org/` (дата обращения 15.02.2015).