Инструмент для моделирования балансировки потока задач пользователей между несколькими независимыми вычислительными кластерами

М.Ю. Воробьев[1], А.А. Рыбаков[2], А.Н. Сальников[3]

[1,2]МСЦ РАН – филиал ФГУ ФНЦ НИИСИ РАН, Москва, Россия
[1,3]МГУ им. М.В. Ломоносова, Москва, Россия

e-mails: [1]mikhail.vorobyov@jscc.ru, [2]rybakov@jscc.ru, [3]salnikov@cs.msu.ru

Аннотация. Работа посвящена организации программного окружения для анализа алгоритмов балансировки нагрузки на ресурсы группы вычислительных кластеров, используемых для высокопроизводительных вычислений. Моделирование производилось путем одновременного запуска нескольких систем ведения очередей SLURM, работающих на нескольких виртуальных машинах в режиме мультиузла. Поток задач создавался посредством инструмента моделирования pseudo-cluster, модифицированного для поддержки одновременной работы с несколькими кластерами. Система апробирована на модельном журнале вычислительных задач пользователей, состоящем из 2000 задач, составленном по типичным паттернам поведения очередей реальных вычислительных кластеров.

Ключевые слова. Распределённые системы, вычислительный кластер, инфраструктура, система ведения очередей, балансировка нагрузки.

Введение

Для распределения ресурсов вычислительного кластера между задачами пользователей используются системы ведения очередей (планировщики), которые выделяют процессорное время вычислительного кластера задачам в зависимости от запрошенных вычислительных ресурсов, времени, приоритета и других параметров.

Специалисту или группе специалистов может быть доступен более чем один вычислительный кластер. В таком случае для выполнения потока задач пользователей разумно использовать все доступные кластеры. Эти кластеры имеют свои независимые системы ведения очередей, возможно построенные на различных программных продуктах. Также они могут управляться различными организациями. Поэтому мы не имеем полной информации о состоянии очередей и не можем влиять на распределение ресурсов [1]. Такая ситуация характерна при функционировании распределенной сети суперкомпьютерных центров коллективного пользования [2].

Для изучения поведения алгоритмов планирования в условиях группы вычислительных кластеров целесообразно применять моделирование для рассмотрения большого количества вариантов за разумное время [3,4]. Известны работы, направленные на решение данной проблемы. В частности в работе [5] описан инструментарий для моделирования систем из нескольких кластеров для изучения алгоритмов планирования. Он основан на системе моделирования процессов SimJava. Но сам SimIC публично не доступен. Также можно отметить следующие существующие программные средства для моделирования работы систем ведения очередей: slurm-sim [6], pseudo-cluster [7]. Имитационный пакет slurm-sim использует модифицированный SLURM [8], позволяющий регулировать скорость течения модельного времени. Программная система pseudo-cluster использует оригинальный SLURM, собранный с поддержкой запуска нескольких демонов вычислительного узла slurmd на одной машине. Для ускорения тестирования скрипт, моделирующий задачу, засыпает на время меньшее реального времени работы задачи в заданное число раз.

Постановка задачи

Пусть имеется множество вычислительных кластеров, на каждом из

которых установлена своя система ведения очередей. Каждый кластер имеет множество узлов и обслуживает свою очередь задач. Также присутствует общая очередь задач и метапланировщик (балансировщик). Метапланировщик решает, в очередь какого вычислительного кластера отправить появившуюся задачу пользователя. Назовём такое множество кластеров мультикластером. Каждый кластер имеет множество узлов и обслуживает свою очередь задач. Каждый узел имеет множество процессоров. Задача в рассматриваемом случае определяется требуемым числом процессоров, требуемым временем работы и реально использованным временем.

В рамках данной работы рассмотрим расширение функционала программной системы pseudo-cluster путем добавления поддержки работы с мультикластерами. Для модификации выбран pseudo-cluster, так как он не требует модифицированной версии SLURM, а позволяет использовать стандартный пакет из репозитория большинства дистрибутивов Linux.

Была выполнена модификация программы запуска потока задач и получения лога потока задач. Для обеспечения удобной работы с модельными кластерами также понадобятся скрипты для запуска SLURM с правильными настройками и для его остановки.

Целью работы является создание программной среды для имитационного моделирования прохождения потока задач через системы очередей нескольких вычислительных кластеров с добавлением свойства сжатия реального времени до модельного.

Описание решения и программная реализация

Как и в однокластерном режиме, будем запускать специально настроенный SLURM. Отличием будет то, что SLURM будет запускаться не на той машине, на которой работает головная часть pseudo-cluster. На одной машине будем запускать один SLURM, симулирующий один кластер. Головная часть pseudo-cluster будет общаться с ними по SSH.

Для тестирования разрабатываемого окружения были использованы следующие простые алгоритмы выбора кластера при распределении задач:

- «Round Robin» – данный алгоритм перебирает кластеры по очереди и каждому назначает следующую задачу из очереди.
- «Наиболее свободный» – в этом алгоритме регулярно собирается информация о числе свободных процессоров. При распределении текущая задача отправляется на кластер с наибольшим числом свободных процессоров.

Рассмотрим устройство pseudo-cluster, выбранного в качестве базового программного средства для расширения функционала. Он написан на скриптовом языке общего назначения Python. Программа работает с журналом в формате csv. Журнал содержит данные о запрошенном числе процессоров и лимите времени, времени отправки, времени начала и завершения работы. Точками входа для пользователя являются следующие скрипты:

- scripts/parse_llsummary_output.py, scripts/parse_slurm_db.py – используются для получения лога запуска задач в формате, понимаемом pseudo-cluster, от LoadLeveler с помощью llsummary и от SLURM напрямую из базы данных MySQL, соответственно;
- scripts/print_characteristics.py – для вычисления метрик по логу;
- scripts/pseudo_users_and_groups_operations.py – реализация работы с именами пользователей и групп в логах;
- scripts/run_pseudo_tasks_slurm.py – симуляция активности пользователей по данным из лога.

Опишем схему работы симулятора. Сначала он загружает лог в память в виде списка задач. Для того, чтобы симуляция проходила быстрее чем реальное выполнение, в качестве программы, запускаемой задачей, используется скрипт, засыпающий на заданное модельное (с учетом сжатия) время. Время, на которое засыпает скрипт, определяется как реальное время выполнения задачи, делённое на коэффициент сжатия comp, задаваемый пользователем опцией --time-compress. Далее раз в период времени T_{model}, заданный пользователем через опцию --time-interval в минутах, симулятор выбирает из списка задачи, отправленные, согласно логу, на выполнение за соответствующий период длительностью $T_{real} = T_{model} \times comp$ и отправляет их системе SLURM одного из виртуальных кластеров.

Для поддержки мультикластерного режима были добавлены следующие классы:

- SlurmCluster
- ClusterSelector
- RoundRobinClusterSelector
- MaxFreeCPUsClusterSelector

и доработаны следующие классы:

- Actions_List
- Scheduled_action
- Extended_task_record

SlurmCluster – предоставляет собой интерфейс взаимодействия с кластером. Команды передаются кластеру через SSH. Для этого была использована библиотека paramiko [9]. В конструктор передаются имя хоста (IP адрес) и, при необходимости, путь к файлу секретного SSH ключа. Для начала работы с кластером нужно вызвать метод connect(), который создаст экземпляр paramiko.SSHClient и создаст соединение с хостом. После этого можно выполнять на удалённой машине команды с помощью метода execute_command(command). Получить текущее состояние SLURM (число свободных, занятых и общее число узлов и процессоров) можно с помощью метода updateUsageInfo().

ClusterSelector – базовый класс для реализаций алгоритмов балансировки. В конструктор передаётся список кластеров SlurmCluster, для которых будет выполняться балансировка. Метод select_clusters_for_tasks(task_list) нужно переопределять в классах-наследниках, реализуя в нём алгоритм балансировки. Результатом работы метода является присвоение полю cluster_to_send элементов task_list выбраного для запуска кластера. Элементы task_list, которым не были назначены кластеры, остаются неотправленными и передаются для распределения снова при следующем запуске.

RoundRobinClusterSelector – реализация алгоритма «Round Robin». На каждой итерации цикла по task_list задаче назначается следующий по порядку кластер. Очерёдность кластеров обеспечивается хранением индекса следующего кластера для назначения в списке. Когда индекс сбрасывается в 0, когда доходит до конца списка.

MaxFreeCPUsClusterSelector – реализация алгоритма «Наиболее свободный». При вызове select_clusters_for_tasks(task_list) состояние всех SlurmCluster из clusters обновляется с помощью метода updateUsageInfo(). На каждой итерации цикла по task_list ищется кластер с максимальным числом свободных процессоров, который и назначается задаче. Кроме этого, в состоянии кластера изменяется число занятых и свободных процессоров на число процессоров, требуемых задачей. Последняя операция выполняется вместо получения реальной информации с кластера из-за больших накладных расходов по времени при использовании сетевого соединения SSH.

Actions_List и Scheduled_action – список действий для выполнения в ближайший период моделирования. Именно эти классы ставят задачи на выполнение и отменяют их. В оба класса добавлено поле логического типа multicluster, определяющее режим работы симулятора. Значение устанавливается одноимённым аргументом конструктора. В методы submit_task() и cancel_task() класса Scheduled_action добавлены альтернативные ветки выполнения для мультикластерного режима. В нём подготовленная команда вместо локального выполнения с помощью execvp() выполняется на соответствующем кластере с помощью метода execute_command().

Extended_task_record – класс, который содержит информацию о задаче. Добавлено поле cluster_to_send, для указания кластера, выбранного для запуска задачи.

Также были внесены изменения в скрипт симулирования потока задач run_pseudo_tasks_slurm.py. В основной цикл был добавлен вызов метода ClusterSelector.select_clusters_for_tasks(task_list) в мультикластерном режиме для распределения задач по кластерам.

Для нового режима добавлены опции командной строки:

- --multicluster – для включения мультикластерного режима;
- --cluster-selector – выбор алгоритма балансировки (RoundRobinn, MaxFreeCPUs);
- --hosts-file – путь к файлу со списком сетевых адресов машин, на которых запущены кластеры SLURM;
- --ssh-key – путь к файлу секретного ключа SSH.

На основе скрипта для получения лога запуска задач напрямую из базы данных parse_slurm_db.py был написан скрипт для получения лога с помощью утилиты sacct parse_slurm_sacct.py. Опции нового скрипта аналогичны опциям исходного за исключением добавления опции --host, с помощью которой указывается сетевой адрес машины, с которой нужно получить лог.

Для быстрой перенастройки SLURM в различные конфигурации были написаны скрипты на языке командной оболочки POSIX shell:

- start-slurm.sh – запуск SLURM с заданным числом узлов и процессоров на заданных машинах. В качестве опций принимает количество узлов (-n), количество процессоров на каждом узле (-c). Также скриптом используется шаблон конфигурационного файла, который должен находиться в текущем каталоге и называться slurm.template.conf. Значения опций -n и -c подставляются вместо $nn и $nc соответственно и получившийся файл копируется на машины.
- stop-slurm.sh – остановка SLURM на заданных машинах.
- clean-accounting-logs.sh – удаление логов запуска задач. Очищает состояние SLURM перед началом следующего теста.

Каждый из этих перечисленных скриптов имеет опцию -h, с помощью которой передаётся путь к файлу, содержащему список сетевых адресов машин, над которыми нужно совершить операцию.

Тестовые запуски

Для проведения тестирования разработанной программной среды на 8-и виртуальных машинах, управляемых системой libvirt [10] (см. рис. 1), была установлена система ведения очередей SLURM 17.11.2. Характеристики виртуальных машин:

- операционная система: Ubuntu 18.04;
- процессор: 1 ядро Intel Core i7-3770K CPU 3.50GHz;
- оперативная память: 1ГБ.

Рис. 1. Пример интерфейса приложения для управления виртуальными машинами virt-manager

В качестве тестовых данных использовался модельный журнал вычислительных задач пользователей, содержащий 2000 задач.

Для тестирования были использованы модели кластеров, состоящих из 256 узлов по 8 процессоров. Характеристики модельных задач выбраны таким образом, чтобы все задачи могли поместиться на модельном кластере. Число моделируемых узлов было ограничено доступными ресурсами виртуальных машин. При большем числе запущенных slurmd SLURM завершался с ошибками, не отработав положенное время. Что означает, что физические ресурсы были использованы полностью.

Было выполнено по 3 запуска с каждым из 2 алгоритмов («Round Robin» и «Наиболее свободный») на 1, 2, 4 и 8 кластерах.

Для осуществления всех намеченных запусков в автоматическом режиме был написан скрипт на POSIX shell, который использовал ранее написанные start-slurm.sh, stop-slurm.sh и clean-accounting-logs.sh. Пример запуска скрипта моделирования потока задач:

```
scripts/run_pseudo_tasks_slurm.py
    --time-compress 1024
    --time-interval 1
    --multicluster
    --path-to-task-script
        ./pseudo_cluster_task.sh
    --hosts-file ./hosts.0-15
    --cluster-selector $sel
> ./exp/${sel}\_16/$i/run.log
```

Объяснение опций:

- --time-compress 1024 – модельное время ускоряется по сравнению с реальным в 1024 раза;
- --time-interval 1 – модельный период равен одной минуте реального времени;
- --multicluster – мультикластерный режим включен;
- --path-to-task-script ./pseudo_cluster_task.sh – путь к скрипту, моделирующему задание путём выполнения sleep с заданным временем;
- --hosts-file ./hosts.0-15 – путь к файлу со списком сетевых адресов машин, отдаваемых под управление pseudo-cluster;
- --cluster-selector $sel – выбор алгоритма балансировки.

Также данный скрипт после каждого теста сохраняет лог с помощью parse_slurm_sacct.py.

Используя скрипт print_characteristics.py, получим метрику среднего времени ожидания задачи для каждого запуска отдельно. При этом кроме исходного модельного потока задач будем использовать еще один поток, в котором нагрузка увеличена в 10 раз (вместо каждой поступающей на выполнении задачи на распределение отправляется 10 ее копий). Объединим полученные данные при помощи shell скрипта в таблицу с колонками: алгоритм, число кластеров, номер запуска, имя кластера, среднее время ожидания задачи. Для дальнейшей обработки данных воспользуемся средой Jupyter Notebook, библиотекой анализа данный pandas и библиотекой построения графиков Matplotlib. Усредним метрики, сгруппировав строки по алгоритму и числу кластеров. По этим данным построим графики среднего времени ожидания задачи в очереди для двух рассмотренных алгоритмов. Графики представлены на рис. 2 и рис. 3.

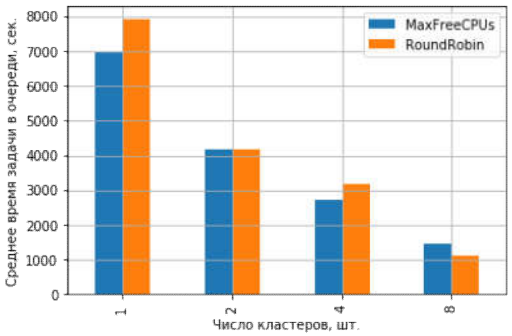

Рис. 2. Зависимость среднего времени ожидания задачи в очереди от числа кластеров для разных алгоритмов балансировки

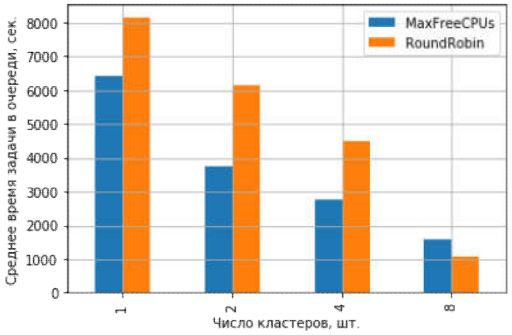

Рис. 3. Зависимость среднего времени ожидания задачи в очереди от числа кластеров для разных алгоритмов балансировки при использовании увеличенной в 10 раз нагрузки

Заключение

В систему моделирования кластеров pseudo-cluster был добавлен режим симулирования мультикластера и средства для автоматического развёртывания инфраструктуры мультикластера, которые позволяют изучать поведение различных алгоритмов балансировки нагрузки в мультикластере. Были добавлены два простые алгоритма балансировки нагрузки, которые могут быть использованы в качестве шаблонов для добавления новых алгоритмов с более сложной логикой. Таким образом разработана система, позволяющая произвести быстрое тестирование разрабатываемых алгоритмов распределения задач пользователей для мультикластера, отдельные кластеры которого обладают произвольными характеристиками. При этом тестирование алгоритмов распределения задач может быть осуществлено как для случайно генерируемого потока задач, так и для реальных логов работы вычислительных систем.

Работа выполнена в МСЦ РАН в рамках государственного задания по теме 0065-2019-0016. При проведении исследований использовался суперкомпьютер МВС-10П, находящийся в МСЦ РАН.

Development of algorithm for user tasks flow balancing over a number of computational clusters

M.Yu. Vorobyov, A.A. Rybakov, A.N. Salnikov

Abstract. The work is devoted to the organization of a software environment for the analysis of load balancing algorithms for a group of computing clusters for high-performance computing. To simulate the cluster, the SLURM queuing system was used, running on several virtual machines in multi-node mode. The task flow was created using the pseudo-cluster modeling tool, modified for supporting multiple clusters. The system was tested using model task flow journal of 2000 tasks based on typical task queue behavioural patterns of real computing clusters.

Keywords. Distributed systems, computational cluster, infrastructure, queue management system, load balancing.

Литература

1. A. Kertesz, Z. Farkas, P. Kacsuk, T. Kiss. Grid interoperability by multiple broker utilization and meta-broking. // Grid Enabled Remote Instrumentation, Springer, 2009, P. 303–312.
2. Б.М. Шабанов, А.П. Овсянников, А.В. Баранов и др. Проект распределенной сети суперкомпьютерных центров коллективного пользования. // Программные системы: Теория и приложения, 2017, 8:4(35), С. 245–262.
3. M. Rezaei, A. Salnikov. Machine learning techniques to perform predictive analytics of task queues guided by slurm. // 2018 Global Smart Industry Conference (GloSIC), IEEE, 2018, P. 1–6.
4. Tanash M. et al. Improving HPC system performance by predicting job resources via supervised machine learning. // Proceedings of the Practice and Experience in Advanced Research Computing on Rise of the Machines (learning), ACM, 2019.
5. S. Sotiriadis, N. Bessis, N. Antonopoulos, A. Anjum. SimIC: Designing a new inter-cloud simulation platform for integrating large-scale resource management. // Proceedings of the 2013 IEEE 27th International Conference on Advanced Information Networking and Applications, 2013, P. 90–97.
6. Репозиторий slurm-sim. https://github.com/marinazapater/slurm-sim // Дата обращения 12.11.2019.
7. А.Н. Сальников, А.Н. Бойко. Программная система для моделирования активности пользователей вычислительного кластера на основе системы ведения очередей SLURM. // Параллельные вычислительные технологии (ПаВТ'2015): труды международной научной конференции, Издательский центр ЮУрГУ Челябинск, 2015, С. 463–470.
8. Slurm Workload Manager – Documentation. https://slurm.schedmd.com // Дата обращения 12.11.2019.
9. Welcome to Paramiko! – Paramiko documentation. http://www.paramiko.org // Дата обращения 12.11.2019.
10. libvirt: The virtualization API. https://libvirt.org // Дата обращения 12.11.2019.