ПРОГРАММНАЯ СИСТЕМА НЕАГРЕССИВНОГО ТЕСТИРОВАНИЯ КОММУНИКАЦИОННОЙ СРЕДЫ СУПЕРКОМПЬЮТЕРОВ

Д.А. Гаврусь, А.Н. Сальников

Актуальность задачи. Количество узлов в суперкомпьютерах на текущий момент времени может достигать десятков тысяч, но несмотря на высокое развитие технологий обслуживание суперкомпьютера без человека невозможно, а уследить за состоянием всех узлов и коммуникаций в вычислительном кластере не является простой задачей. Существует два подхода отслеживания внутреннего состояния кластера. Первый представляет собой «пассивное» отслеживание состояния узлов специальными программными компонентами для этих целей (Например работы[1,2]). Примерами таких являются BSSC (bullx supercomputer suite components), Supercomputer Management System, Ganglia. Этот способ используется в большинстве случаев, но не всегда показывает нам полную картину внутреннего состояния суперкомпьютера. Аппаратная часть и программная часть суперкомпьютера образуют сложную систему (см. работу [3]), что делает невозможным предсказание поведения коммуникаций в нём. Такого рода непредсказуемость не может отслеживаться с помощью мониторинга аппаратной части системы. Для решения такой проблемы существуют программные инструменты, основанные на передаче сообщений между процессорами. К таким инструментам относятся Intel MPI Benchmarks, SkaMPI, MPIBlib, network_test2. Приложение network_test2[4] разработано на факультете ВМК МГУ, способно тестировать коммуникационную среду с использованием библиотеки MPI путём отправки сообщений между узлами и анализа времени передачи этих сообщений. Его особенностью является то, что оно осуществляет отправку сообщений между всеми парами узлов суперкомпьютера на которых оно запущено. Если мы хотим получить полную картину о коммуникационной среде, то необходимо осуществить запуск такого инструмента на всех вычислительных узлах кластера, иначе можно упустить проблемную область.

Предлагаемый подход к решению задачи. В большинстве случаев суперкомпьютеры являются многопользовательскими, а это означает, что в какой-либо момент времени на нём могут быть одновременно запущены тысячи задач и осуществлять работу сотни людей. Если требуется осуществить запуск приложения на всех узлах кластера, то нужно дождаться, пока не будет выполняться ни одной пользовательской задачи, или просто закрыть на время тестирования доступ пользователей к ресурсам вычислительной системы. Чтобы осуществить полное тестирование приложению нужно отправить сообщения между всеми парами вычислительных узлов, а это может занять время. Учитывая количество узлов в современных машинах, кластер целиком будет занят одним приложением долгое время, а это не лучший выход из ситуации. Другим подходом является тестирование некоторых подмножеств узлов, которые вместе составят весь кластер. Тогда не придется ждать момента освобождения всего кластера от пользовательских задач, и не придется закрывать пользователям доступ для работы. Приложение будет запускаться несколько раз на небольшом количестве доступных узлов, что уменьшит общее время тестирования. Также стоит отметить, что в предложенном подходе для нас будут не столь существенны возможные администраторские ограничения такие как, например, количество выделенных узлов для задачи или количество времени, выделенное на выполнение задач.

Планировщики задач.

Чтобы пользователям и администраторам было удобно работать с вычислительной системой, разработаны системы управления ресурсами суперкомпьютера, так называемые «планировщики». К таким относятся LoadLeveler, Slurm, Inferno-Owen. Планировщик выполняет три основные функции:

1. Предоставление ресурсов пользователям для выполнения их задач в рамках выделенного для них времени.
2. Предоставляет среду для запуска, выполнения и мониторинга работы на наборах узлов суперкомпьютера
3. Управление очередью задач, выделение ресурсов для них в автоматическом режиме
4. Slurm

В данной работе будет использоваться популярный планировщик Slurm (The Simple Linux Utility for Resource Management), в частности, установленный на суперкомпьютере «Ломоносов».

Slurm представляет собой систему управления задачами на вычислительном кластере и предоставляет обширный набор команд для пользователей и администраторов.

Рассмотрим 3 основные пользовательские команды, которые будут использованы в дальнейшем:

1. sinfo — предоставляет информацию о разделах, узлах суперкомпьютера и их состояниях на текущий момент. Команда sinfo выводит список с информацией обо всех разделах суперкомпьютера, в нём представлены такие данные как имя раздела, его доступность, ограничения по времени на одну задачу, количество узлов в разделе, текущее состояние узлов и, непосредственно, список узлов.

   Раздел может находиться в состоянии down, drain, alloc, idle.

   Рабочими состояниями являются alloc и idle, первый случай означает то, что узлы раздела выделены для какой-либо задачи, второй случай означает что все узлы раздела свободны.

При этом раздел с одним именем, но разными состояниями в выводе представляется разными записями, это нужно учесть в дальнейшей работе. Команда имеет множество различных форматов вывода, но списки узлов в разделах она выводит таким образом, что если имеется несколько последовательно именованных узлов, то она выведет их имена в формате диапазонного представления. К примеру, если имеются узлы с одинаковым текущим состоянием и именами:
node1-028-01, node1-028-02, node1-028-03, node1-029-05, node1-030-04, node1-030-06, node1-030-07,
то slurm отдаст список в виде:
node1-028-[01-03], node1-029-05, node1-030-[04,06-07]

2. sbatch — осуществляет постановку задачи в очередь в пакетном режиме, предоставляет файл с информацией о запущенной задаче. Пользователь может указать в параметрах число узлов, которое требуется выделить для задачи, раздел суперкомпьютера, на котором он хочет выполнять свою задачу, список имен узлов для задачи, скрипт запуска задачи и запускаемое приложение.
3. squeue — текущее состояние очереди кластера, для каждой задачи предоставляет ее номер, имя пользователя, запустившего ее, текущее состояние задачи, время выполнения, количество и список выделенных для задачи узлов. (рис. 1).

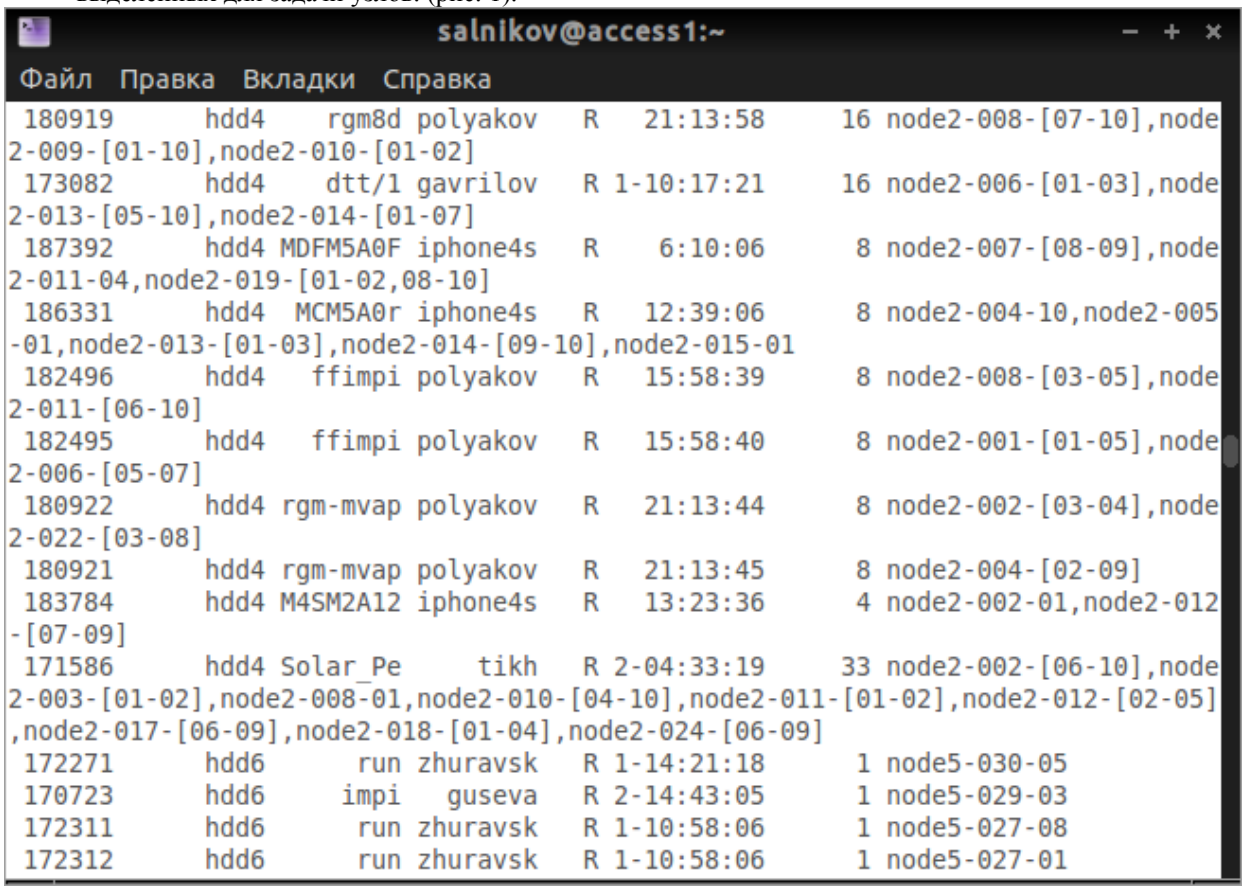

Рис. 1. Пример вывода команды squeue Рис. 1. Пример вывода команды squeue

network_test2

В работе будет использоваться тестирующее приложение network_test2, разработанное на факультете ВМК МГУ. Оно осуществляет последовательный перебор всех пар узлов для выявления проблемных областей в коммуникациях суперкомпьютера. Приложение в процессе выполнения создает 5 файлов с результатами работы, один из которых содержит в себе список имен узлов, на которых производилось тестирование, а четыре других содержат результаты тестов в виде различных статистических характеристик. Описание приложения и подробная информация о типах тестов содержится в [4].

Программная реализация

Программа была реализована на языке Python 2.6. На вход она требует от пользователя имя раздела, количество узлов для одной задачи, максимальное количество запущенных задачи и диапазон длин сообщений, посылаемый приложением network_test2. Также может быть построен граф топологии выбранного раздела. В результате работы программы будут созданы файлы, с результатами каждого запуска network_test2.

В процессе реализации было написано несколько классов и основной модуль, в процессе работы программы создается несколько вспомогательных файлов, которые впоследствии автоматически удаляются.

Описание основных классов и вспомогательных модулей.

Было реализовано несколько вспомогательных классов для удобства дальнейшей работы:

Класс Partition, хранящий в своих полях информацию о разделе кластера.

Метод класса Partition get_all_nodes() осуществляет раскрытие диапазонного представления имен узлов, о котором было сказано в описании Slurm, и возвращает список из отдельных узлов.

Класс ClusterInfo, осуществляющий работу с данными, полученными из вывода команды sinfo. Класс осуществляет выбор узлов из заданного пользователем раздела суперкомпьютера, соответственно основными составляющими данного класса являются объекты типа Partition. С помощью данного класса производится выборка и сортировка узлов определенного раздела. Поскольку некоторые узлы могут принадлежать одновременно нескольким разделам, при выборе всех узлов из всех разделов класс отсеивает повторные вхождения какого-либо узла. При выборе узлов раздела создаются несколько объектов Partition, каждый из которых принадлежит к одному разделу, но с разными состояниями. Метод get_partition_nodes() возвращает отсортированный список всех узлов, независимо от их состояния.

Модуль miniutils, содержит небольшие методы общего назначения, для удаления временных файлов, для получения информации о пользователе.

Описание работы алгоритма основного модуля программы

С помощью класса subprocess осуществляется вызов команды sinfo, а посредством класса ClusterInfo возвращается отсортированный список раздела, на котором будет осуществляться тестирование.

Происходит последовательный выбор узлов в количестве number_of_nodes_for_run / 2 - первая половина тестируемых узлов. Во время выбора проверяется доступность узла в текущий момент времени, соответствующий узел будет выбран, только если он доступен. Если во время выбора узлов встречаются недоступные, то накапливается смещение относительно начала списка узлов раздела. Следующим шагом является подбор второй половины узлов, из оставшихся после выбора первой, для образования целой порции для запуска, выбор происходит по таким же правилам. (см. пример)

Сформированная порция записывается во временный файл с префиксом nodefile и номером, изменяющимся относительно количества запущенных задач. Аналогично выбираются порции до тех пор, пока не будет готово количество порций, равное максимальному числу указанному пользователем запущенных задач. Для каждого сформированного файла с помощью класса subprocess выполняется запуск задачи командой sbatch с необходимыми параметрами. После чего файлы с префиксами nodefile удаляются, и происходит анализ очереди задач, программа проверяет количество пользовательских задач, запущенных на данном разделе до тех пор, пока оно не станет меньше максимально установленного значения. Происходит обновление до актуального состояния узлов суперкомпьютера. Это необходимо, так как нет гарантий, что узел будет доступен всё время с момента запуска Python-программы, а некоторые недоступные после запуска программы узлы могут оказаться доступными. Происходит следующая выборка узлов, и запускается соответствующее количество задач. Таким образом, в результате работы алгоритма будут получены файлы с результатами тестов для каждой выбранной порции.

Небольшой пример работы алгоритма:

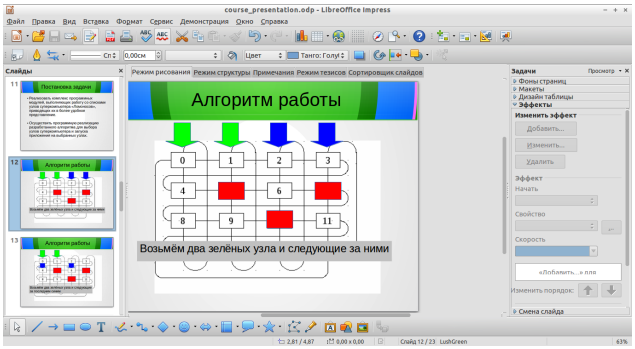

Рис. 2. Узлы 0 1 зафиксированы с ними выбраны 2 3

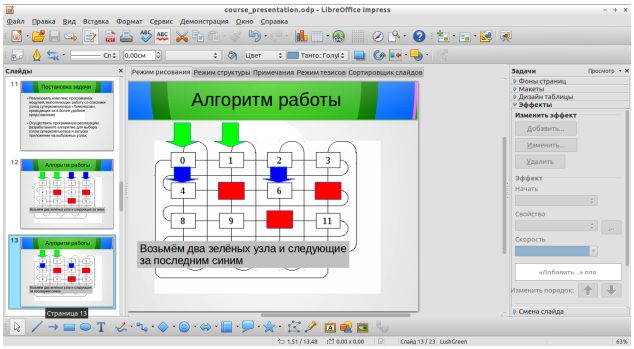

Рис. 3. Узлы 0 1 фиксированы, с ними выбраны 4 и 6, так как 5 не отвечает

Сперва выбрались узлы 0 1 2 3, причем 0 1 зафиксировались для дальнейшего выбора до конца списка, а затем к 0 1 требуется присоединить еще 2 следующих. После присоединения узла 4, алгоритм обнаружит, что узел 5 недоступен, но это и был второй узел, который требовалось взять. Алгоритм его пропускает и увеличивает счетчик смещения на 1, будто бы изначально хотелось взять не 2, а 3 узла. Попытка взятия узла 6 заканчивается успешно, получается нужное количество узлов с номерами 0 1 4 6. Учитывая смещение, алгоритм попытается взять узел 7, но ввиду того, что он недоступен, смещение увеличится на еще единицу.

Класс Graph

Как некоторый побочный эффект выполняемой работы со списком узлов суперкомпьютера «Ломоносов», на основе классов Python-программы представилась возможность реализовать класс, формирующий файл формата *.dot (*.gv) с описанием ребер и вершин графа топологии всего суперкомпьютера «Ломоносов», или отдельно взятого его раздела. Формирование графа осуществляется на основе именования узлов суперкомпьютера и представляет собой эталонную топологию, с которой в дальнейшем можно сравнивать

топологии, полученные экспериментальными путями. Классу было найдено практическое применение в продолжении исследований по статьям [5], [6]. В дальнейшем можно оптимизировать такого рода программный продукт, который на основе заранее известных правил именования узлов кластера будет строить граф его топологии.

Результаты тестирования.

Программа была запущена и протестирована на суперкомпьютере «Ломоносов», для примера работы программы был выбран раздел суперкомпьютера, состоящий из 64 узлов. На момент старта выполнения программы доступными оказались 61 узел. Было выбрано количество узлов для одной задачи равное 8, а максимальное количество таких задач 4. Диапазон длин сообщений 0 - 1000

Для запуска программы использовались команды:

python main.py

partition=<имя_раздела>

--number_of_nodes=<количество_узлов>

--tasks=<максимальное_количество_задач>

-e <начальная_длина_сообщения> -b <конечная_длина_сообщения>

В результате работы получилось 120 запусков программы и соответственно, пятикратное количество файлов с результатами, ввиду того, что программа после каждого запуска создаёт 5 файлов.

Выводы.

Был разработан алгоритм выбора узлов суперкомпьютера для многократной постановки в очередь «Ломоносова» тестирующего приложения network_test2. В дальнейшем планируется усовершенствовать алгоритм выбора узлов исходя из анализа очереди в разделе, и осуществлять запуск в первую очередь на доступных в ближайшее или текущее время узлах, это может еще сильнее ускорить время тестирования. Работа велась при частичной поддержке гранта РФФИ 12-07-3108.

ЛИТЕРАТУРА:

1. А.В. Созыкин, М.Л. Гольдштейн, М.А. Черноскутов The system of operative monitoring of supercomputer’s «uran» temperature and power consumption // Software and Systems № 4 за 2011 год. с.120-123
2. М.Г. Курносов, А.А. Пазников Децентрализованные алгоритмы диспетчеризации пространственно-распределенных вычислительных систем // Вестник Томского государственного университета, №1 2012 год
3. Jiuxing Liu, B. Chandrasekaran, Jiesheng Wu, W. Jiang, Sushmitha Kini, Weikuan Yu, Darius Buntinas, Peter Wyckoff, D.K. Panda Performance comparison of MPI implementations over InfiniBand, Myrinet and Quadrics // SC '03 proceedings of the 2003 ACM/IEEE conference on supercomputing, page 58
4. А.Н. Сальников, Д.Ю. Андреев, Р. Д. Лебедев Инструментальная система для анализа характеристик коммуникационной среды вычислительного кластера на основе функций стандарта MPI // Вестник Московского университета серия 15 Вычислительная математики и кибернетика № 1, 2012, - Москва, издательство МГУ, с.39-48
5. П.С. Банников, А.Н. Сальников Система визуализации результатов MPI-тестирования коммуникационной среды вычислительных комплексов // Вестник Пермского университета. Математика. Механика. Информатика. Выпуск 3(11) 2012г. Редакционно-издательский отдел Пермского государственного национального исследовательского университета, с. 80-85
6. П.С. Банников, А.Н. Сальников Трёхмерная визуализация результатов MPI-тестирования коммуникационной среды вычислительного кластера. // Труды Всероссийской научной конференции "Научный сервис в сети Интернет: поиск новых решений", сентябрь 2012г., г. Новороссийск, Издательство Московского государственного университета им. М.В. Ломоносова, Москва, 2012 с. 517-521