РАЗРАБОТКА АЛГОРИТМА ДИСПЕТЧЕРИЗАЦИИ В МОБИЛЬНОМ ВЫЧИСЛИТЕЛЬНОМ КЛАСТЕРЕ

СкопченкоА., ФроловС.Г.
Научный руководитель Ботыгин И.А.
Томский политехнический университет
aas83@tpu.ru

Введение

На сегодняшний день существует необходимость в распараллеливании вычислений.

Одним из подходов для распараллеливания выполнения вычислительно-емких задач являются GRID-системы [1]. То есть, решением проблемы с распараллеливанием может являться подключение нескольких компьютеров в одну сеть. Сеть из нескольких компьютеров, в дальнейшем, будем называть мобильным вычислительным кластером (МВК), центральный компьютер обозначим как диспетчер, а компьютер из кластера – агент. Даже собрав кластер, мы не решаем проблему распараллеливания, так как работать модули будут как угодно, но не так как нужно.

Как вариант GRID-система, сконфигурированная на основе персональных компьютеров, является удобным и простым инструментом для решения таких задач. Поскольку время обмена между узлами МВК достаточно большое (нет высокоскоростных шин обмена данными), то будем рассматривать задачи, не требующие обработки в реальном времени.

Как известно, для полного распараллеливания нужно придерживаться правила «1 ядро = 1 поток (модуль)». Конечно, даже при работе на МВК с одним ядром (в общем случае 1 ПК) технически мы можем создать 100 и более параллельных потоков, но в таком случае распараллеливанием будет управлять операционная система, запуская потоки на ядрах ПК, передавая управление то одному, то другому потоку. Хорошо, если имеется под рукой суперкомпьютер, в инфраструктуре которого 100 и более ядер. В этом случае можно запустить 100, 200 и более потоков параллельно, но, к сожалению, не у всех такой суперкомпьютер есть под рукой. Однако, в реальной жизни сформированный МВК включает ограниченное число ядер, меньшее, чем число возможных параллельных потоков. Поэтому возникает проблема распределения модулей между ядрами МВК. Одной из важных задач в этой проблеме является обеспечение правильного порядка выполнения вычислительных модулей (в одних узлах данные от предыдущего узла нужны, в других нет), а также обеспечение синхронизации и взаимодействия между модулями, особенно выполняющихся на разных ПК.

Алгоритм диспетчеризации

Пусть, вычислительная схема, реализуемая на МВК представлена следующим ориентированным графом (рис. 1).

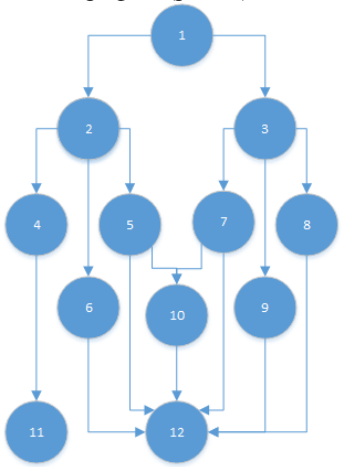

Рис. 1. Схема вычислений

Из данной схемы можно получить следующую информацию:

приоритет в выполнении модулей;

зависимость между модулями;

какие модули могут работать параллельно.

Схему вычислительного процесса представим в виде модифицированной матрицы смежности (таблица).

Таблица. Визуализация схемы вычислительного процесса

1	2	3	4	5	6	7	8	9	10	11	12	
0	0	0	0	0	0	0	0	0	0	0	0	1
1	0	0	0	0	0	0	0	0	0	0	0	2
1	0	0	0	0	0	0	0	0	0	0	0	3
0	1	0	0	0	0	0	0	0	0	0	0	4
0	1	0	0	0	0	0	0	0	0	0	0	5
0	1	0	0	0	0	0	0	0	0	0	0	6
0	0	1	0	0	0	0	0	0	0	0	0	7
0	0	1	0	0	0	0	0	0	0	0	0	8
0	0	1	0	0	0	0	0	0	0	0	0	9
0	0	0	0	1	0	1	0	0	0	0	0	10
0	0	0	1	0	0	0	0	0	0	0	0	11
0	0	0	0	1	1	1	1	1	1	0	0	12

Из данной таблицы можно получить:

Порядок выполнения вычислительных модулей и потоки данных между ними. Например, первый модуль (по горизонтали) не зависит ни от каких модулей и должен выполниться первым, а, например, девятый модуль (по горизонтали) зависит от третьего модуля (по вертикали). Соответственно, он может выполниться только после того, как отработает третий модуль.

Какие модули могут выполниться раньше. Например, так как первый модуль никого не ждет он может быть выполнен сразу, а второй и третий модуль, дождавшись выполнения первого модуля могут выполняться параллельно.

Диспетчер, получив и обработав данную таблицу, сможет понять, как он должен работать и сформировать пакеты команд для отправки на другие агенты.

Заметим, что для полноценного распараллеливания диспетчеру не хватает данных. Таблица не является полным решением проблемы, так как даже с имеющимися данными диспетчер не имеет представления о доступных агентах, количестве ядер у агентов и др. Поэтому после завершения выполнения каждого вычислительного модуля диспетчер опрашивает подключенных к нему агентов на предмет:

доступности (свободен ли данный агент);

вычислительной мощности (сколько доступно физических ядер).

После получения данной информации диспетчер заносит ее в файл конфигурации и в дальнейшем будет работать с ней, периодически обновляя информацию путем повторного опроса агента на изменения. Данные в файле хранятся в следующем виде: «1;192.168.0.4;0;4», где 1 – идентификационный номер(id) агента; 192.168.0.4 – IP-адрес агента; 0 – агент свободен (1 – агент занят); 4 – количество доступных физических ядер.

Полученные выше данные обеспечивают выполнение вычислительных модулей на ограниченном числе ядер.

Синхронизация и взаимодействие между агентами и выполняемыми на них модулями обеспечивается «объектами занятости», в структуре которых диспетчер записывает: на какой компьютер и какое задание отправлено. Запись имеет следующий вид, например, «1;3», где 1 – это id агента; 3 – имя модуля на выполнение.

Диспетчер, отправив задание агенту на выполнение модуля и получив ответ от агента о принятии, вновь опрашивает агента и отмечает у себя, что данный агент занят (изменяет флаг с 0 на 1 в файле конфигурации). То же самое диспетчер повторяет с другими агентами, которые могут выполняться согласно матрице смежности.

Дополнительно, диспетчер постоянно проводит опрос агентов на предмет наличие соединения. Если в течение некоторого периода времени от агента не поступает ответ, то диспетчер считает, что у агента произошел сбой. Это диспетчер отмечает в лог-файле и передает задание этого агента другому свободному агенту.

Диспетчер заканчивает работу при исчерпании выполняемых модулей в матрице смежности.

Алгоритм работы диспетчера включает в себя следующие основные шаги:

1. Осуществляется просмотр матрицы смежности вычислительной схемы на предмет определения начальных модулей выполнения (поиск нулевых строк в матрице смежности).
2. Анализируются подключенные агенты на предмет доступности и вычислительной мощности.
3. Полученная информация от агентов заносится в файл конфигурации.
4. Запуск начальных модулей на доступных агентах и изменение соответствующей записи в файле конфигурации.
5. Опрос агентов на предмет наличия соединения.
6. При потере соединения с агентом осуществляется возврат на шаг 4.
7. После выполнения текущего модуля изменяется запись в файле конфигурации.
8. Осуществляется анализ матрицы смежности на предмет определения модулей, которые могут быть выполнены следующими.
9. Если таких модулей нет, то диспетчер заканчивает работу.
10. Запуск модуля на доступных агентах и изменение соответствующей записи в файле конфигурации.
11. Опрос агентов на предмет наличия соединения.
12. При потере соединения с агентом осуществляется возврат на шаг 8.

Диспетчер разработан в среде IntelliJ IDEA на языке Java [2 - 4]. Одним из решающих факторов в пользу языка Java является его платформная независимость.

Заключение

Разработан алгоритм, позволяющий диспетчировать выполнение произвольной вычислительной схемы на мобильном вычислительном кластере с произвольной инфраструктурой.

Список использованных источников

1. Grid-технология [Электронный ресурс] //Linux. Кластер. – URL: http://cluster.linux-ekb.info/grid.php (дата обращения: 15.10.2016).

2. Java 8. Полное руководство Девятое издание/ Герберт Шилдт., 2015. – 1376 с.

3. Обзор Java.util.concurrent[Электронный ресурс] Блог Habrahabr – URL: https://habrahabr.ru/company/luxoft/blog/157273/

4. Многопоточность в Java [Электронный ресурс] Блог Habrahabr – URL: https://habrahabr.ru/post/164487/ (дата обращения: 21.10.2016).