



Московский Государственный Университет имени М.В.Ломоносова
Факультет Вычислительной Математики и Кибернетики
Кафедра Системного Программирования

Курсовая работа

**Разработка и реализация прототипа
системы потоковой обработки данных**

Автор:
гр. 427

Овчинников Дмитрий

Научный руководитель:
д.т.н профессор Кузнецов Сергей Дмитриевич

Москва, 2015

Содержание

1	Постановка задачи	3
1.1	Актуальность	3
1.2	Общие требования	3
1.3	Выбор языка программирования	4
2	Разработка прототипа	6
2.1	Общие сведения о схеме	6
2.2	Взаимодействие с пользователем	6
2.3	Логика приложения в общих чертах	8
3	Реализация прототипа	9
	Заключение	10
	Список литературы	11
	Приложение. Код программы	12

1 Постановка задачи

1.1 Актуальность

Поскольку рост информации коллосален в последнее время, ее нужно как-то быстро обрабатывать, и чем быстрее, тем лучше. Существуют такие технологии, как Hadoop, MapReduce, которые безусловно полезные, но довольно-таки медленные и непригодные для "Big data in realtime". Это сфера, где нужно обрабатывать много (потенциально бесконечно) данных за очень короткий промежуток времени, к таким относятся, например:

1. Системы анализа и мониторинга
2. Задачи оптимизации работы сетей и отдельно каждого устройства в сети
3. Обнаружение подозрительных действий на охраняемых территориях
4. Телефония, обслуживание АТС
5. Финансовый трейдинг

Это сфера ИТ еще нова и только набирает популярность, и до некоторого момента проблема быстрых вычислений решалась путем закупки дорогостоящего оборудования. Однако сейчас решение проблемы все больше и больше ложится на плечи программистов, которые меняют подходы к обработке информации.

Существует тенденция, когда все вычисления (которых слишком много) ставят "на поток" т.е. выделяют отдельно систему, которая обрабатывает непрерывные данные, входящие к ней и передает дальше, не задерживая. О разработке такой системы и пойдет речь в данной курсовой работе.

1.2 Общие требования

Планируется реализовать систему, в виде отдельной библиотеки, которая представляла бы собой один кластер распределенной системы потоковой обработки данных. Для начала разработки необходимо обозначить набор требований, которым должна удовлетворять система:

1. Система не должна использовать работу с жестким диском, чтобы уменьшить время выполнения.
2. Система должна предоставлять инструменты для создания графов(топологий) вычислений.
3. Система должна иметь возможность пользоваться всеми ресурсами процессора, т.е. должна иметь возможно распараллеливания.
4. Система желательно быть кросс-платформенной
5. Система должна гарантировать полный проход графа вычислений.
6. Система не должна заботиться о том, как принимать данные и куда их потом передавать.
Об этом должен позаботиться пользователь-программист, использующий эту библиотеку.
7. Система должна быть достаточно гибкой.
Это означает, что пользователь, имел бы возможность настраивать множество параметров, таких как коэффициент распараллеливания, время жизни топологии, максимальную память, которые занимают данные в системе.

Так выглядят требования в общем виде. В процессе разработке они будут уточняться и поясняться. Сразу следует отметить то, что данная реализация - это прототип системы потоковой обработки данных, а значит в этой системе не будет реализовано то, как кластеры будут взаимодействовать между собой. Будут лишь некоторые замечания и предположения о том, как это будет проходить. А во всем остальном - должен получиться полноценный распараллеливаемый кластер параллельной обработки данных.

1.3 Выбор языка программирования

Есть множество замечательных языков программирования, у каждого есть свои преимущества и недостатки. Автор же остановился на языке Java, поскольку он кросс-платформенный и довольно-таки распространенный в тех сферах, где планируется использовать данную систему (Постановка задачи). Однако, у Java есть один большой

недостаток - программы на Java чуть ли не самые медленные среди всех языков программирования. Для такой системы закономерно выбирать какой-нибудь функциональный язык из семейства Lisp, поскольку обладают лаконичностью, компактностью и выразительностью. Хорошо подходят для этого Clojure, Scala или Groovy, которые реализованы на java-машине, т.е. обладают полной совместимостью с Java и обладают той же кросс-платформенностью. Но в силу ограниченности по времени написания данной работы, не было возможности выучить один из этих языков. Поэтому, жертвуя производительностью в пользу удобства написания, данная система будет полностью написана на Java.

2 Разработка прототипа

2.1 Общие сведения о схеме

Здесь, как говорится не будем изобретать велосипед заново (хотя, возможно, стоило бы и попытаться сделать все по-другому) и возьмем уже привычную схему для потоковой системы.

Основное место занимает топология (Topology), которая определяет, как именно данные перемещаются внутри программы. Элементами топологии являются воронки (spouts) и сита (bolts). Роль воронок заключается в том, чтобы извлекать данные извне, например, из БД, http, xml или из файлов. В воронках нет никаких вычислительных элементов, они всего лишь занимаются извлечением данных. Говоря терминами функционального программирования воронки - это “грязные” функции, т.е. они взаимодействуют с внешним миром, и их результат не всегда зависит от входных данных.

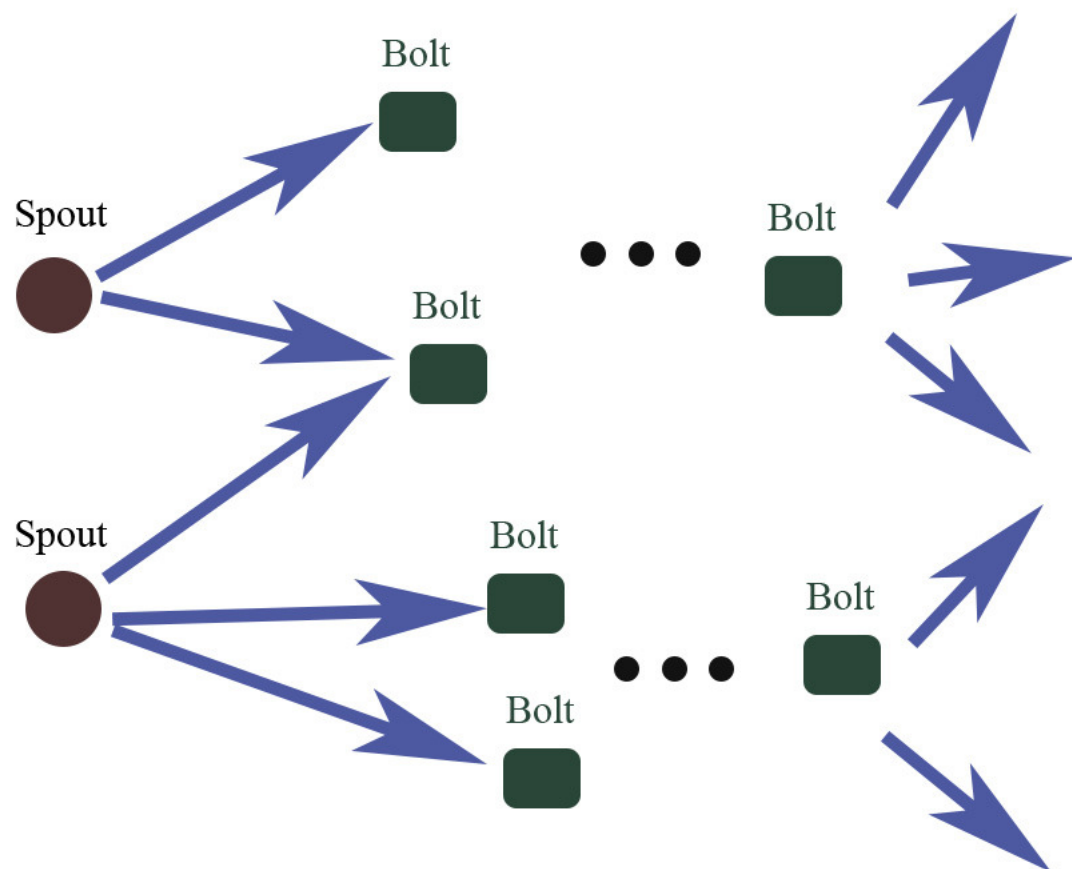
Смысл же сит (bolts) заключается в том, чтобы обрабатывать кортежи данных, которые им предназначены и передавать их дальше. Сита также могут взаимодействовать с внешним миром - передавать результат работы топологии дальше.

На рис. Схема топологии показана примерная схема топологии (в зависимости от реализации она может различаться).

Итак все в этом приложении должно строиться вокруг топологии и для топологии, чтобы иметь возможность вычислить кортежи, которые будут подаваться на вход. С точки зрения пользователя этой информации будет достаточно для понимания, как работать с данной системой в целом.

2.2 Взаимодействие с пользователем

Один из первых вопросов, которые возникают при разработке любого приложения - как данные от пользователя будут попадать в программу? Каким образом пользователь будет взаимодействовать с программой? Здесь мы воспользуемся опытом уже известной и достаточно популярной системой потоковой обработки данных - Apache Storm. Кластер системы будет предоставляться как библиотека на языке java, который программист-пользователь, импортируя в свою программу, получит набор интерфейсов



Topology

Рис. 1: Схема топологии

для коммуникации с кластером.

Основные возможности, которые доступны пользователю - это создание воронок и сит, наследующих определенный интерфейс, описанный автором. Предполагается, что пользователь будет добавлять сита и воронки в объект класса `Topology` - основной объект, которым будет манипулировать система при вычислении.

Пользователю также должна предоставляться возможно определять кортежи. Единственное требование на данном этапе, которое должно удовлетворяться кортежами - оно должно наследовать стандартный интерфейс кортежа, описанный в системе. На данном этапе этот интерфейс не имеет определенных методов, только лишь расширяет `java-интерфейс Serializable`. Это необходимо в случае распределения системы на несколько кластеров.

2.3 Логика приложения в общих чертах

За неимением опыта разработки больших и сложных систем, автор испытывает небольшую сложность в формальном описании системы и приносит свои извинения.

Конечно, выглядит не все так просто, как было написано в схемах общего вида, давайте же разберемся в этом. Первое, что нужно сказать, что топология(читать "граф") представлена стандартным образом, в виде списка воронок, сит и связей между ними. Пока здесь все ясно и не требует пояснений. Воронки и сита не должны храниться в одной "куче" потому что воронки не могут принимать данные, а значит, стоило бы их отделить, для того, чтобы программа знала, кому передавать управление. Естественно, у каждой воронки и сита должно быть уникальное имя в виде строки. Это нужно для того, чтобы не было коллизий при хешировании (предполагается, что имена будут захешированы для более быстрого доступа).

Также необходимо обговорить еще один важный вопрос: где будут располагаться кортежи данных? Как они будут перемещаться по графу вычислений? Для кортежей будет выделена конкретная память, настраиваемая пользователем, у каждого кортежа будет имя - имя сита, для которого он предназначен. Поскольку одновременно могут несколько кортежей предназначаться к одному ситу, то хранить придется кортежи не по одному, а списками, в котором содержатся все кортежи предназначенные для конкретного сита с уникальным именем. Хранение будет в специальном классе, который

будет обеспечивать поиск по имени кортежа и добавление кортежа с именем, который только что его обработал. Такой подход был реализован по причине того, что эти методы буду вызываться внутри сит, где недоступен список со связями графа, т.е. сито не знает, кому передавать дальше. Хранение будет реализовано в виде отдельного класса Collector. Ссылка на этот класс будет передаваться при инициализации сита. Класс Collector берет на себя ответственность за хранение кортежей и дальнейшего продвижения по графу.

Теперь поговорим о том, как будет происходить выполнение операций определенных в сите или воронке. Во-первых, сито и воронка будут унаследовать стандартные интерфейсы, в которых будут находиться методы, предназначенные для описания действий над кортежами. Как уже понятно из написанного выше, сита сами будут извлекать кортежи из коллектора, и это ложится на плечи программиста-пользователя. Т.е. чтобы пропустить через сито один кортеж, достаточно выполнить известный метод из сита и он сам достанет и/или вернет в коллектор кортежи. После этого необходимо выполнить метод коллектора, который будет совершать подмену имени (т.к. сито не знает, кому передавать дальше, он возвращает кортежи со своим именем) на то, которое будет в паре с данным в списке связей графа.

С воронками все гораздо проще, они будут выдавать по одному кортежу в коллектор со своим именем, а коллектор сам позаботится, какое имя ему подставить вместо этого. Очевидно, коллектору не обязательно делать подмену после каждой обработки кортежа, можно накапливать некоторое количество, а потом обработать пачкой.

Теперь нужно упомянуть о том, как будет происходить распараллеливание. Оно будет проводиться стандартными классами Java. Предполагается, что будут отдельные классы-работники, которые будут принимать на себя топологию и делиться еще на 2 потока, одни будут выполнять методы из воронок, а другие из сит. Значит, при минимальной конфигурации, будет 4 потока для работы с топологией + 1 основной, который выполняет логику кластера. При необходимости, основной поток должен уметь завершить потомков (хотя в Java это не приветствуется - предполагается, что потоки-потомки сами завершают работу, но в нашем случае потомки работают бесконечно).

На данном этапе, автор считает, что с логикой все довольно-таки понятно и пора приступать к реализации.

3 Реализация прототипа

Заключение

Список литературы

Приложение. Код программы