Санкт-Петербургский государственный университет

Факультет математики, механики

Дымникова Наталья Александровна

Разработка распределенной системы обработки данных

Курсовая работа

Научный руководитель:
Коновалов М. В.

Санкт-Петербург
2016

Оглавление

Введение

Объемы информации, которые генерируются современными вычислительными системами, требуют больших ресурсов для анализа и обработки. Вычислительных ресурсов одного устройства может не хватать для достаточно быстрого решения поставленных задач. При этом бизнес-задачи имеют строгие ограничения на время исполнения, а любые задержки приводят к финансовым потерям. Поэтому на сегодняшний день разработка программного обеспечения для обработки данных в пределах одного вычислительного устройства является не рациональной. Для этого создается распределенная среда, объединяющая десятки и сотни вычислительных устройств в единую сеть для параллельной обработки информации.

В компании EMC разрабатывается продукт Elastic Cloud Storage (далее ECS), основная особенность которого – это горизонтальная масштабируемость, что позволяет строить системы с большим количеством узлов. При отлаживании и тестировании ECS создается большое количество диагностической информации, распределенной на кластере, необходимой для поиска и исправления ошибок функционирования. Данная работа предназначена для решения этой проблемы, то есть для сбора и анализа данных, хранящихся на кластере. Другими словами, она описывает решение проблемы обработки данных, хранящихся на различных устройствах в одном кластере, при помощи построения запроса, состоящего из нескольких этапов: создание или чтение данных, их обработка и слияние нескольких потоков.

1. Постановка задачи

Целью данной работы является разработка и реализация распределенной системы обработки данных на кластере. В рамках работы поставлены следующие задачи:

- Разработка архитектуры распределенной системы
- Создание интерфейса построения запросов к системе
- Реализация наблюдения за состоянием выполнения запроса и оповещения об ошибке
- Разработка модуля нахождения наилучшей конфигурации запроса

2. Основная идея

Основной идеей является создание потоков данных на кластере. Например, потоком может быть последовательность строк из файла или результатов запроса к индексу. В Java существует два подхода к потокам. Первый, это java.Stream [2]. Основная его идея заключается в том, что источник выдает элемент, который по цепочке обработки проходит все этапы, прежде чем следующий элемент начнет обрабатываться.

Есть и иной подход: Reactive Extensions Observable [3]. Его основная идея заключается в том, что клиент подписывается на источник данных и запрашивает некоторое количество данных, источник просто начинает их порождать в необходимом количестве. Observable предоставляет гораздо более широкие возможности по организации обработки потоков данных нежели java.Stream, и основная ценность Observable это то, что передаваемые элементы поражаются асинхронно и не требуют блокировки текущего потока.

Система, построенная в рамках данной работы, реализует подход rx.Observable на кластере. Например, (рис. 1) может быть несколько источников данных, которые необходимо последовательно отфильтровать, как-то преобразовать и объединить в один поток, затем отдать приемнику (потребителю).

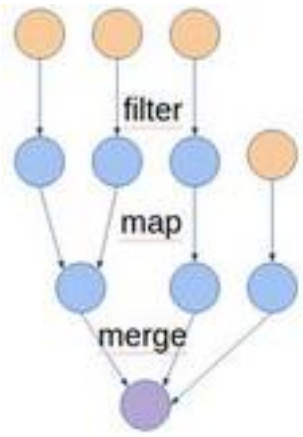

Рис. 1: Обработка данных

3. Архитектура

Сначала было необходимо решить проблему общения между узлами кластера. Для этого выбран фреймворк Akka [1], который позволяет производить распределенные вычисления на кластере, состоящем из одноранговых узлов, что позволяет избежать создания единой точки отказа системы.

Как было сказано выше, потоки данных являются основной идеей, для этого была создана компонента Планировщик (Scheduler), которая позволяет создавать последовательные этапы преобразования данных. Например, так (рис. 2). Здесь createObservable(...) представляет создание источника данных, к примеру, чтение строк из файла, subscribe() представляет создание получателя (в данном примере пустого), а каждый map(...) представляет определенный этап преобразования данных. В целом, эти этапы задают некоторый “поток” данных.

Эта компонента производит распределение подзадач на узлы кластера и после создания получателя (subscribe) дает сигнал к началу обработки данных. Каждый этап выполняется в Compute Pool-е. И вот это вот является интерфейсом для создания запросов к системе.

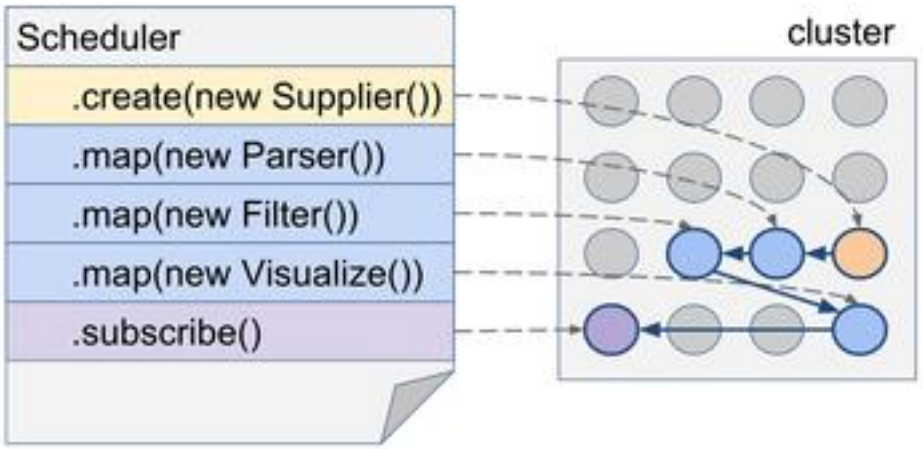

Рис. 2: Планировщик, распределение по узлам кластера

Изначально интерфейс позволял создавать только линейные запро-

сы, однако при использовании в проекте оказалось, что часто необходимо слияние нескольких потоков в один, такая возможность была реализована. Интерфейс построения запросов к системе содержит следующие методы:

- createObservable() – источник данных
- map – преобразование данных
- merge – слияние данных (так же здесь может происходить их упорядочивание)
- subscribe – получатель данных

4. Вычислительные пулы

Вычислительный пул (Compute Pool) имеется на каждом рабочем узле, он представляет собой пул вычислительных ресурсов и предназначен для выполнения вычислительных задач, которыми являются этапы запроса по обработке данных. Compute Pool получает список всех этапов обработки, берет из них назначенные ему этапы, запускает их. Также для каждого запроса он создает отдельные компоненты, которые будут следить за выполнением запроса, чтобы иметь возможность остановить весь процесс и сообщить об этом пользователю, если на каком-то этапе произошла ошибка или если один из узлов неожиданно отключился (то есть если произошла аварийная ситуация).

5. Конфигурация запроса

Конфигурация запроса заключается в том, что Планировщику необходимо выбрать узлы для обрыботки данных. Существует несколько вариантов:

- Find first (нахождение первого) – назначение этапа происходит первому подходящему. По сути это рандом, но вся система строиться на фреймворке Akka, особенностью которого является блокировка треда при обработке одного сообщения, то есть в один момент обрабатывается только одно сообщение, а остальные складываются в очередь. То есть тот узел, который ответит первым, что готов выполнять работу, будет являться наименее загруженным, иметь меньшую очередь сообщений.

- Round-robin – назначение этапа тому, кто дольше прочих не делал работу, то есть по выбираем узел по кругу.

- Less loaded member (наименее загруженный участник) – назначение этапа наименее загруженному узлу, также здесь можно учитывать скорость соединения со следующим и предыдущим этапом.

Первые два метода становятся крайне не эффективными, если кластер большой и между некоторыми его узлами сеть, по которой происходит связь, работает значительно хуже, например, из-за удаленности или способа соединения. Для этого можно собирать информацию о состоянии кластера, сети и загрузки устройств и выбирать оптимальный путь. Так как рассматривается теоретическая ситуация, то узлов может быть сколь угодно много, то будем разбивать кластер на группы и выходить за пределы группы как можно реже. Реализован вариант, когда каждый узел — отдельная группа.

Список стратегий можно расширять и добавлять более оптимальные для каждого более конкретного кластера.

6. Особенности решения

Основные особенности решения:

- Использования исключительно языка Java, что делает систему кросплатформенной, без лишних зависимостей.
- Такое решение (в отличии от Apache Spark) не требует специализированных тяжеловесных средств управления кластером, таких как Apache Mesos или Apache Hadoop YARN. Вместо этого решение содержит программный модуль который позволяет развертывать свою полную копию на любом сетевом узле доступном по SSH.
- Не требуется дублировать данные, как это делается в logStash + elasticSerach, для которых проводится еще и предварительная индексация, которая полностью копирует данные. Из-за этого запуск занимает много времени. В данном решение необходимую индексацию можно проводить во время работы программы.
- Система построена на общедоступных и бесплатных компонентах не требующих покупки коммерческих лицензий на использование, в отличии от прочих.

Заключение

В ходе выплонения данной работы были получены следующие результаты:

- Разработана система распределенной обработки данных, которая строится на работе компоненты Планировщик, которая позволяет запускать задачи на одноранговом кластере
- Создан интерфейс для построения запросов
- Разработано наблюдение за состоянием выполнения запроса и оповещения об ошибке
- Реализован модуль с различными способами нахождения узлов кластера для выполнения на них запроса
- Система была протестирована и затем использована во внутреннем проекте log-analisys в компании EMC

Исходный текст результатов моей работы можно получить по адресу https://github.com/NataliaDymnikova/akka.scheduler
Имя пользователя, под которым работал автор: NataliaDymnikova.

Список литературы

[1] Akka. Akka Documentation 2.4.6. — URL: http://doc.akka.io/docs/ (online; accessed: 23.05.2016).

[2] Java. Stream (Java Platform SE 8). — URL: https://docs.oracle.com/javase/8/docs/api/java/util/stream/Stream.html (online; accessed: 23.05.2016).

[3] ReactiveX. Observable Documentation. — URL: http://reactivex.io/documentation/observable.html (online; accessed: 23.05.2016).