Сравнительное тестирование фреймворков для обработки больших данных

За последние 10 лет в области больших данных появилось множество различных решений для обработки, хранения и передачи данных. Эти решения могут быть как взаимозаменяемы, так и дополнять друг друга на разных стадиях обработки. Объединенные под общим названием «экосистема Hadoop» [], эти решения сейчас активно используются для многих практических задач в областях, связанных с обработкой данных, и практически не имеют аналогов. В данном докладе предлагается сравнить два похожих по назначению фреймворка для потоковой обработки данных: Apache Flink [] и Apache Spark [].

Точкой отсчета и средством внешней оценки фреймворков является инфраструктура Polar.DB [], причем использовалась в основном базовая библиотека. Рассматривались разные задачи и те, которые являются выигрышными для Spark и Flink и некоторые задачи, решение которых на них не сулит выигрыша.

В докладе рассмотрены три задачи, проанализированы элементы решений и полученные результаты. Сравнение во-многом получилось качественное, поскольку важность параметров тестирования не всегда позволяет давать интегральную оценку. Что лучше – медленная работа или «быстрое» исчерпывание оперативной памяти.

# Сравнение подходов Apache Flink и Apache Spark

Фреймворк Apache Flink в целом построен на потоковой парадигме обработки. Он поддерживает обработку данных в режимах streaming и batch. Одной из главных особенностей Apache Flink являются состояния потока (state), обеспечивающее гибкость и отказоустойчивость при обработке данных.

Программа для Apache Flink состоит из источника потока (source), операций и стока (слива) потока (sink). В качестве источника и стока можно указывать файлы, базы данных, брокеры сообщений, которые взаимодействуют с Apache Flink с помощью классов коннекторов. Построенная таким образом программа формирует граф операций, который оптимизируется внутренним движком Apache Flink, после чего программа приводится в действие.

В Apache Spark данные собираются и обрабатываются порциями (batch обработка). Apache Spark предлагает для разработки структуры данных RDD (Resilient Distributive Dataset) и Dataframe. Первая позволяет распределено хранить любые объекты языка Java, вторая хранит данные в распределенных таблицах, напоминающих таблицы классических реляционных БД, и даже поддерживает язык SQL.

Программа для Apache Spark - это набор трансформаций и действий над этими структурами данных. Трансформация - это массовое преобразование, применяемое ко всей структуре данных, такое как агрегация, фильтрация, функция map. Трансформации построены в парадигме ленивых вычислений, то есть применяются они не сразу, после вызова действий, таких как вывод подсчет полей, в консоль или в файл.

Программы для обоих фреймворков можно писать на языках Java, Scala, Python и SQL. Оба проекта разработаны с открытым исходным кодом, обладают высокой отказоустойчивостью и активно развиваются. Преимуществом Apache Flink можно назвать вариант обработки в режиме streaming, что позволяет изменять данные в режиме realtime. Однако Apache Spark существует дольше и обладает более выразительным API.

# Тестовая площадка

Для тестирования данных алгоритмов была собрана тестовая площадка на основе виртуальных машин. Было создано 4 виртуальных машины с операционной системой Ubuntu 18, объединенных в кластер с помощью виртуальной сети. Каждой машине было отведено 20 ГБ оперативной памяти и 4 ядра процессора Intel Xeon CPU E5-2680 2.7 GHz. Для тестирования использовались сборки Apache Flink 1.11 и Apache Spark <ВЕРСИЯ>, взятые из открытых источников. Оба фреймворка были установлены в формате standalone, с одной машиной в качестве master и тремя в качестве slave.

# Задача WordCount

Первая задача, на которой проводилось сравнительное тестирование имеет название WordCount. На вход этой задаче подается текстовый поток из слов английского алфавита, разделенных пробелами. На выходе программа должна выдать список из пар, в которых записано слово и количество повторений этого слова в потоке. Задача считается одной из классических для потоковой обработки данных, поэтому она реализована в качестве примера в обоих сравниваемых фреймворках. В работе по оценке использовались слегка модифицированные версии этих программ.

Для тестирования был написан генератор текстов, который записывал в файл N строк по 50 слов в каждой. Слова формировались случайно, но из конечного набора.

# Задача по нахождению контигов в графе Де Брёйна

Следующая задача для тестирования была не такой тривиальной, как WordCount. Нахождение контигов в графе Де Брейна - это один из этапов задачи о сборке генома, важной в области биоинформатики.Выбор задачи был обусловлен существующими наработками.[ссылка] Далее приведем математическую постановку задачи.

Программе на вход подаются строки разной длины алфавита <A, G, C, T>.

Далее выбирается некоторое положительное целое число K, меньшее, чем самая короткая из входных строк. Каждая из входных строк разбивается на максимальное количество (длина строки - K + 1) подстрок длины K. Каждая из подстрок является вершиной для графа Де Брёйна. Между двумя вершинами проводится ребро, если в подстроки, соответствующие этим вершинам, пересекаются по K-1 символам в исходной строке. После построения графа требуется найти все последовательные цепочки вершин и свернуть их в одну вершину, заменяя соответствующие имена вершин на более длинные. Такие свернутые цепочки и называются контигами. На вывод программа должна выдать все получившиеся контиги и их длину.