BIGDATA

Лекция 3

ПЛАН

- Join в MapReduce
- HeavyHitters
- Формы и форматы хранения и обработки данных

ВСПОМНИМ ВАРИАНТЫ JOIN

- Совсем классический (INNER)
- Классический LEFT/RIGHT
- Частный случай: найти тех, кто без "сцепки"
- Классический OUTER

ВСПОМНИМ ВАРИАНТЫ JOIN

- CROSS + условие
- Другие варианты: например, "similarity join"
- Сцепление текстов, картинок по похожести
- В идеале - семантической

КРУПНЫМ ПЛАНОМ

- Зависит от размеров таблиц
- Иногда - от размеров "сцепок"
- И от типа JOIN
- Тривиально решается любой вариант для двух малых таблиц
- Как минимум - с позиций нашего курса

BIG + SMALL

- Начнем с классического INNER
- Одна таблица очень большая, другая маленькая
- Пример: лог событий с полем "id организации"
- И каталог организаций

BIG + SMALL

- На map-узлах прочитаем маленькую таблицу из HDFS
- Организуем словарь в памяти
- Ключ - соединяемое поле
- Записи большой протащим через map
- Ищем по ключу совпадения

BIG + SMALL

- map-фазы достаточно
- Можем добавить reduce
- Например, JOIN + GROUP BY
- Можно с WHERE
- И/или с HAVING

ВАРИАЦИИ

- Найти записи большой таблицы без связи (вариант LEFT/RIGHT) - проверяем отсутствие ключа в словаре
- Найти записи маленькой без связи - тут сложнее
- Нужен "единый взгляд" на большую, без reduce никак
- Ключи reduce - это ключи (primary в SQL-смысле) маленькой таблицы
- combine весьма желателен

ВАРИАЦИИ

- Такой reduce+combine - экономное решение
- Но его мало
- Узнаем тех, кто есть в первой
- Но не тех, кто во второй

ВАРИАЦИИ

- Можно сохранить уникальные ключи первой - если их мало
- Получим второй маленький файл
- И организуем локальное вычитание множеств
- А если их не мало ?

ВАРИАЦИИ

- Другой вариант - на map-фазу докинуть и записи маленькой таблицы
- Различать по map-ключу
- Данные маленькой перекинуть на reduce
- С тем же ключом, что и данные большой
- value - признак того, что это ключ из маленькой

ВАРИАЦИИ

- value - признак того, что это ключ из маленькой
- И нужные данные
- На reduce будем знать про все ключи
- OUTER - комбинация LEFT + RIGHT

ЧАСТНЫЙ СЛУЧАЙ

- Одна большая таблица
- А маленьких много
- И/или часто меняются
- И для каждой маленькой надо искать отсутствующие в большой первичные ключи
- Или присутствующие

ПРИМЕР

- Есть логи с базовых станций мобильных операторов
- Зафиксированы появлявшиеся там IMEI или IMSI
- К нам отдельно поступают пачки данных о каких-то людях
- Логи с IMEI/IMSI огромны и постоянны
- Данные о людях влезают в машину

ПРИМЕР

- Данные о людях периодически ротируются
- В данных о человеке есть его IMEI/IMSI
- Хотим быстро узнавать - кто там фигурировал в логе в конкретном месте
- Или не фигурировал
- Или в нескольких местах (несколько логов, тоже стационарных)

ЧАСТНЫЙ СЛУЧАЙ

- Не хотим каждый раз гонять через reduce
- Применим Bloom-filter
- Над множеством значений ключа

А КАК ЕГО ПОСЧИТАТЬ ?

- Тоже MapReduce
- Данные же стационарные
- Есть варианты в представлении данных
- Можно подумать и обсудить

CROSS С УСЛОВИЕМ

- Сначала пробуем свести к классическому с фильтрацией
- (Стандартный шаг для CROSS с условием)
- CROSS дает "прямоугольник" вариантов
- Проходим каждую позицию в нем

CROSS С УСЛОВИЕМ

- Один map отвечает за строку
- На худой конец перебираем
- Можно пооптимизировать
- В зависимости от природы сопутствующих сравнений

BIG + BIG

- Обе таблицы большие
- На map передаем данные обоих таблиц
- Помечаем их происхождение
- Например, в map-ключе

BIG + BIG

- JOIN-ключ становится reduce-ключом
- MAP-ключ идет в значение
- На reduce приедут все записи с общим ключом
- Потенциально - с обоих таблиц (если они там были)

ДАЛЬШЕ - ВАРИАНТЫ

- Деление по Small/Big повторяется
- На уровне групп записей с общим ключом
- Записей мало - можем просто прочитать в память
- И породить пары вложенным циклом

ДАЛЬШЕ - ВАРИАНТЫ

- Big + Small в любую сторону: хорошо бы прочитать маленькую часть в память
- А потом - вычитывать длинную
- Первая проблема - сортировка по значениям
- По умолчанию - отсутствует

ДАЛЬШЕ - ВАРИАНТЫ

- Можно включить Secondary Sort - сортировку по значениям на входе в reducer
- С одной стороны - мы и так сортируем по ключу reduce
- Как будто просто удлиняем ключ
- Но увеличиваем задержку

ДАЛЬШЕ - ВАРИАНТЫ

- Отдельный вопрос - а кто small ?
- В общем случае для каждого общего ключа - свой
- Может помочь HyperLogLog
- На этапе формирования таблиц

ДАЛЬШЕ - ВАРИАНТЫ

- Но есть нюанс
- Для каждого уникального значения должны завести HyperLogLog
- А уникальных может много с низкой частотностью
- Можно мерить диапазоны - но оценка может оказаться грубой

HEAVYHITTERS

- Алгоритмическая задача на поиск особо частых элементов
- Частный случай: 1/2
- Хотим понять, есть ли в длинном числовом массиве число, занимающее большинство позиций
- За один проход и константную память
- Ну или хотя бы за два

Находим кандидата

```
 1 def maj(data):
 2     p, c = None, 0
 3     for v in data:
 4         if v == p:
 5             c += 1
 6         elif c == 0:
 7             p = v
 8             c = 1
 9         else:
10             c -= 1
11
12     return p if c > 0 else None
```

ОБОСНУЕМ

- Пусть число реально встречается в большинстве позиций
- Покажем, что c будет больше нуля
- И в p будет это число

Модифицируем

```
 1 def maj(data):
 2     p, c = None, Counter()
 3 
 4     for v in data:
 5         if v == p:
 6             c[p] += 1
 7         elif c[p] == 0:
 8             p = v
 9             c[p] += 1
10         else:
11             c[p] -= 1
12 
13     return p if c[p] > 0 else None
```

ОБОСНУЕМ

- Соотнесем динамику счетчика и реальную частотность
- Для доминирующего элемента
- Переберем ветки и две ситуации в каждой
- p - доминирующий и не доминирующий

ОБОСНУЕМ

- 1, 3: частота растет, счетчик растет
- 2, 4: не наша частота, не наш счетчик
- 5: частота растет, счетчик (наш) - не меняется
- 6: частота (наша) не меняется, счетчик убывает

ОБОСНУЕМ

- 5/6 - не могут идти друг за другом
- И в начале не могут
- На каждое из них будет хотя бы одно другое состояние
- Значит сумма потерь в счетчике не превышает n/2

ИТОГИ

- Если есть доминирующий - то найдем
- Но если нашли кого-то - не факт, что он доминирует
- Простой контрпример: [1, 1, 1, 2, 2, 2, 3]

ОБЩИЙ СЛУЧАЙ

- Возьмем N счетчиков
- К каждому можно привязать число
- В начале ничто не привязано
- Перебираем элементы

ОБЩИЙ СЛУЧАЙ

- Если элемент привязан к счетчику - увеличиваем
- Если не привязан и еще есть свободные - занимаем свободный и ставим в 1
- Если нет свободных - уменьшаем на 1 все
- Обнулившиеся освобождаем

ПРИМЕР

- Заполняем таблицу на 100 млн элементов
- Заводим 10000 счетчиков
- Получаем кандидатов на присутствие 10000+ записях
- Кто не в этом множестве - точно не опасны для JOIN

BIG + BIG

- Вариант 1: что-то не так с постановкой задачи
- Вариант 2: что-то еще есть в запросе
- Например, GROUP BY + COUNT
- Или фильтрация, лимит
- Возможны индивидуальные решения

CROSS JOIN + УСЛОВИЕ

- Сложная история в общем случае
- Для частных случаев есть варианты
- Например, группировка скользящим окном
- Если маленький дискретный диапазон

ПРОМЕЖУТОЧНЫЙ ИТОГ

- Научились повторять реляционную модель
- (А точно и не требуется)
- И что-то сверх того
- Поговорим слегка о другом

ПРЕДСТАВЛЕНИЕ ДАННЫХ

- Начинали с текстово-строчной модели
- Плавно перешли к CSV
- Что еще бывает и зачем ?

ПЛЮСЫ-МИНУСЫ CSV

- Дает базовую табличную структуру
- Но данные объекта "плоские"
- А реальные данные объекта обычно итеративны и иерархичны
- Варианты выхода: нормализация в смысле реляционных моделей
- Или ситуативные костыли: перечень ключевых слов через запятую и т.п.

JSON

- "Локально" примененный JSON
- Каждый объект - отдельный документик
- С точки зрения идеалов реляционной модели - тоже костыль
- Но довольно качественный
- И чистая реляционность на больших данных в целом не заходит

JSON

- Структурно похожие аналоги - XML, YAML
- В контексте Hadoop используются мало
- XML более многословен, чем JSON
- Человекочитаемость YAML тоже стоит байтов
- А без нее - непонятно, зачем он

ТЕКСТОВЫЕ И БИНАРНЫЕ

- Беда текстовых - избыточные траты места
- Особенно - когда в данных много чисел
- Особенно когда они еще и в иерархии
- Есть много бинарных форматов

ЯЗЫКОВЫЕ БИНАРНЫЕ ФОРМАТЫ

- pickle в Python
- Object Serialization в Java
- Есть свои плюсы и минусы
- Основной - привязка к языку

ВНЕЯЗЫКОВЫЕ БИНАРНЫЕ ФОРМАТЫ

- Первый значимый - protobuf
- Идея - есть общее понятие структуры
- Давайте описывать структуру данных
- На своем отдельном языке

PROTOBUF

- Будем описывать поля и их имена
- У полей будут типы данных
- Целочисленные, вещественные
- Что-то типа массивов и перечислений

PROTOBUF

- Есть свой компилятор
- Он по описаниям структур порождает код на заказанном языке программирования
- Этот код умеет создавать описанные структуры
- Заполнять и читать поля