Модельная система

Рассмотрим простую модельную систему: у нас есть фронты (веб-сайт, приложение и т.д.) и бэк (серверное приложение). Фронтов может быть много, потому что одновременно может быть несколько версий сайта, приложения и пр. Кроме того, может быть сервис по приемке логов за балансером с несколькими инстансами. Все пользовательские действия записываются на фронте и на бэке. Для простоты будем считать, что любому событию на фронте соответствует некоторый лог на сервере.

Мы хотим считать какую-нибудь пользовательскую метрику, например, время, которое пользователь что-нибудь ждет (крутится splash окошечко, прогресс бар и т.д.). Для этого, мы сливаем логи из браузера и из сервера в потоковую систему, которая считает эту метрику. Далее, потоковая система отдает значения метрики для отображения в дашборд.

Важно, что считать метрику мы хотим по пользовательским сессиям, т.е. по промежуткам активности. Чтобы определять, что сессия закончилась, мы используем ML модель, которая по специальным фичам определяет,

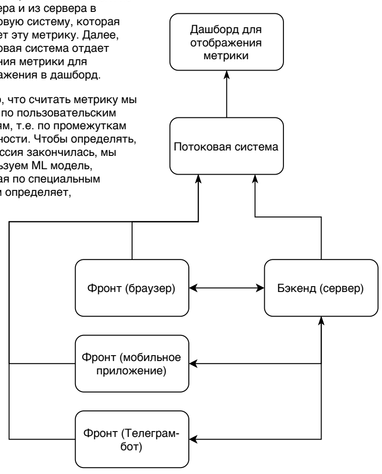
Структура логов

user id

front id

ts

payload



Naive graph

Источник браузерных событий

Это источник данных, через который идет доставка логов из браузера. Каждый элемент потока содержит структуру лога, описанную выше.

Мержим потоки

Это искусственная операция для того, чтобы из двух потоков сделать один. Ее можно представить просто как операцию, которая принимает любые элементы на вход и их отдает на выход. Вводим ее для наглядности, на самом деле, например во Flink, любая операция может иметь несколько входов.

Join логов

В этой операции мы соединяем в один элемент лог из браузера и соответствующий ему лог из сервера. На выходе получается элемент, который содержит в себе оба входных события. Чтобы правильно соединить два лога, нам нужно, чтобы логи от одного и того же пользователя шли на одну машинку физическую. Поэтому, данные шардируются по user id. Во Flink шардирование задается оператором keyBy.

Фичи для фронта

По этой ветке мы должны получить ответ от модели: закончилась сессия или нет. Сначала добавляем фичи для фронта, например, как часто идут события от этого фронта и т.д. Соответственно, также меняем и состояние этой операции, например, ту же частоту. Чтобы состояние было полным, нужно получать все события от одного фронта на одну физическую машину, поэтому данные шардируем по front id.

Фичи для юзера

Аналогичная операция для пользователей. Для согласованности состояния шардируем по user id.

Применяем модель

Применяем модель к полученным фичам для пользователя и фронта, опредяляем, закончилась ли сессия для пользователя. Т.к. модель может иметь разные параметры для каждого пользователя и изменяться во времени (online learning), то шардировать нужно по user id.

Выпускаем метрику

Здесь нам приходят два типа элементов:

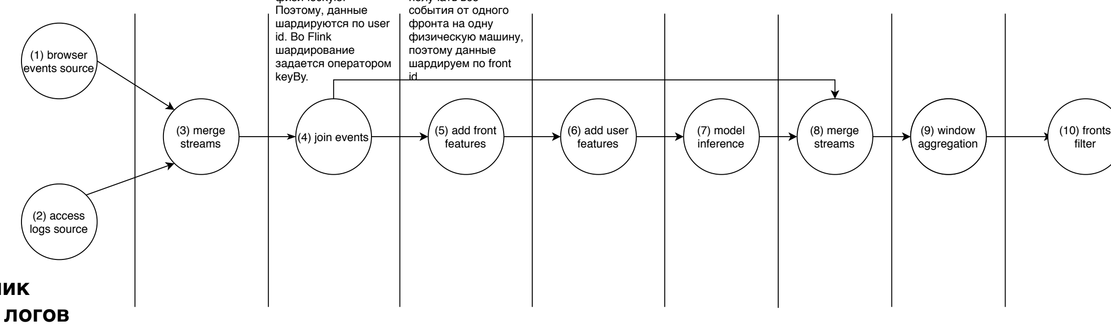
1. Соединенные логи фронт-юзер. Их мы аггрегируем в метрику.

2. Событие конца окна от модели. Выпускаем метрику наружу. Чистим состояние для последней сессии.

Шардируем по user id.

Фильтрация

Опциональная операция. Можем захотеть пофильтровать самые редкие фронты, или старые версии фронтов. Важно, что фичи для этих фронтов посчитаны и учтены в модели. Шардируем по front id, так как считаем, что здесь есть состояние.



partition by user id | partition by front id | partition by user id | partition by user id | partition by user id | partition by front id

Источник access логов

Источник данных, через который идет доставка логов с сервера. Структура логов аналогична браузерным.

Принцип построения оптимального графа

Заметим, что в исходном графе:

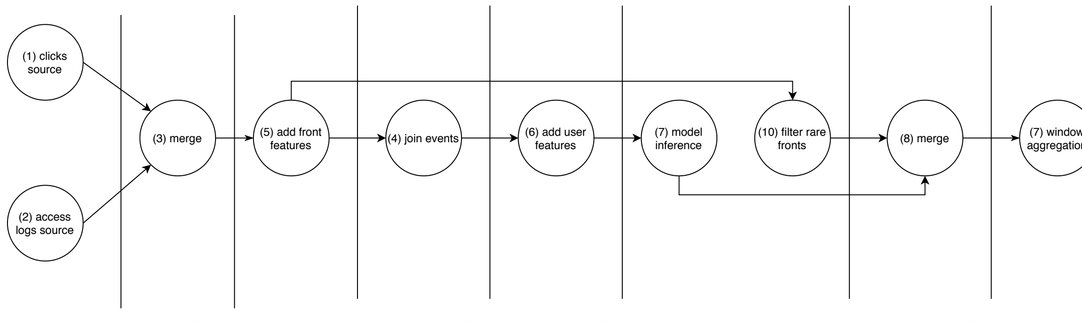
1. Много ре-шардирований

2. Добавление фичей и соединение логов можно менять местами, т.к. добавление фичей только добавляет поля в элемент

3. Фильтр можно делать до агрегации, но после обновления модели

Таким образом, мы можем переставить операции так, чтобы они "склеились" по шардированям и мы бы не гоняли данные по сети.

Optimal graph



partition by front id | partition by user id | partition by user id | partition by user id | partition by user id