DOI 10.37882/2223–2966.2023.04.28

РАЗРАБОТКА ВИРТУАЛЬНОГО ТЕСТОВОГО СТЕНДА И ИССЛЕДОВАНИЕ МЕТОДОВ ДЕДУПЛИКАЦИИ ПОТОКА ДАННЫХ НА ПРИМЕРЕ СОБЫТИЙ ОПЕРАТОРА СВЯЗИ

Потапов Илья Александрович
Магистрант, ФГАОУВО «Национальный исследовательский университет ИТМО»
ilyaros@yandex.ru

Аннотация. При проектировании и разработке сервисов часто возникает ситуация, когда входящий поток данных может содержать дубли. Это может происходить из-за различных причин, и приводит к неправильной интерпретации информации. Эта работа посвящена рассмотрению различных методов поиска и избавления потока данных от дублирующей сообщений, а также разработке виртуального тестового стенда для проверки рассматриваемых методов.

Ключевые слова: дедупликация, нагрузка, производительность, kafka, cassandra, тестирование, разработка, кластер.

DEVELOPMENT OF A VIRTUAL TEST STAND AND RESEARCH OF DATA STREAM DEDUPLICATION METHODS USING THE EXAMPLE OF TELECOM OPERATOR EVENTS

I. Potapov

Summary. When designing and developing services, there are situations when the incoming data stream may contain duplicates. This may occur due to various reasons, which is the cause of errors in the interpretation of information. This paper discusses various methods for finding duplicate messages in a data stream, and the development of a virtual test stand to test these methods.

Keywords: deduplication, load, performance, kafka, cassandra, testing, development, cluster.

Цель

Целью работы является разработка виртуального тестового стенда, на котором можно проводить нагрузочное тестирование различных методов дедупликации данных, а также исследование различных методов дедупликации данных на примере событий оператора связи

Методы

Для разработки тестового стенда использовались вычислительные мощности внешнего облака. Для тестирования использовались 3 независимые машины, на которых был развернут кластер kafka, кластер cassandra, а так же был создан сервис генерирующий сообщения с заданным профилем нагрузки.

Для всех рассматриваемых методов дедупликации проводилось тестирование на всех профилях нагрузки. При этом проводились замеры скорости обработки данных и объема потребляемых ресурсов.

Результаты

Был разработан тестовый стенд для проверки работы рассматриваемых методов дедупликации данных, а также проверены и проанализированы результаты работы различных методов дедупликации данных.

Выводы

По итогам работы был разработан виртуальный тестовый стенд, который способен выдерживать нагрузку вплоть до 60 000 событий в секунду без работы с БД и до 25 000 событий в режиме работы с БД.

Рассматриваемые методы дедупликации данных были протестированы и по итогам тестов сделаны выводы о преимуществах и недостатках методов, а также

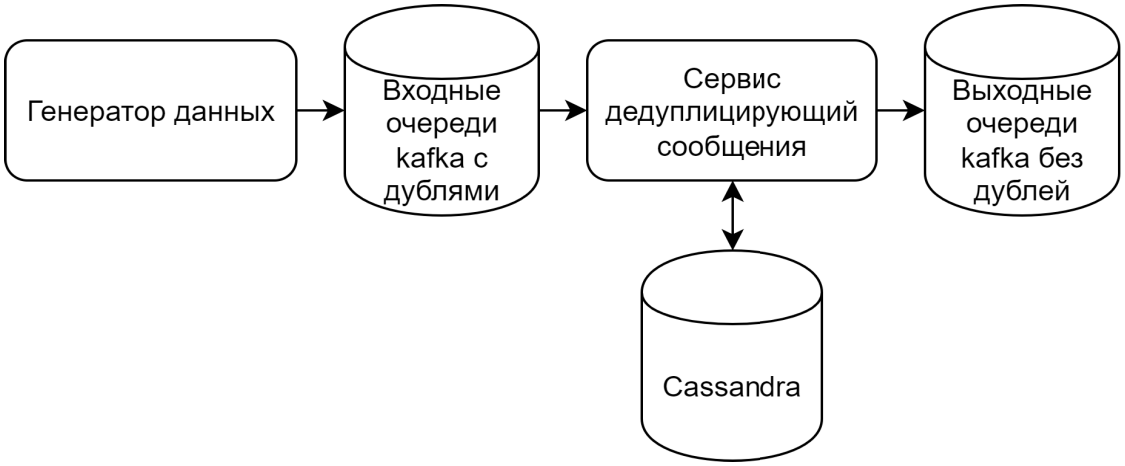

Рис 1. Принципиальная схема тестового стенда

была проанализирована применимость этих методов к различным областям использования.

Введение

При обработке потока данных достаточно часто возникает необходимость устранять дублирование данных, т.е. проводить дедупликацию.

Причины появления дублей данных могут быть разнообразны: переповторы на оборудовании, ошибки операторов, работающих с системой, рассинхронизация распределенных систем и т.д.

Проблема многократно усложняется если данных для дедупликации становится много, и поток данных не ограничен во времени.

В рамках данной работы рассматриваются различные методы дедупликации данных, а также создается виртуальный тестовый стенд, на котором возможно эти методы проверять под нагрузкой

Рассматриваемая проблема, с дублированием, часто возникает в системах где нужна “гарантированная доставка” данных. Одним из подходов гарантированной доставки с асинхронным обменом — является механизм переотправки сообщений до тех пор, пока сервис-получатель не нотифицирует отправителя о том, что сообщение получено.

В таком случае, при каких либо проблемах, переотправок может быть достаточно много. И задача сервиса-получателя усложняется тем, что он должен не только выполнить свою бизнес-логику, но еще и тем, что он должен выполнить ресурсоемкую операцию — дедупликации потоков [1]. Т.е. обеспечить “at-least-once delivery”.

В качестве сообщений, которые формируют нагрузку — будем рассматривать события следующего вида:

{
«userId»: Integer,— Уникальный Id пользователя, сформировавшего событие
«eventId»: Integer,— Уникальный, монотонно возрастающий, Id события в рамках одного пользователя
«eventDate»: DateTime,— Дата и время формирования события
«otherData»: String — Строка, содержащая случайные символы, эмулирующая другие, объемные поля
}

Сообщения такого вида используются в том числе и в системах операторов связи, для нотификаций сервисов о совершении пользователем какого-либо события.

Разработка тестового стенда

Для разрабатываемого тестового стенда, в качестве базы данных, была выбрана Apache Cassandra.

Apache Cassandra — распределённая система управления базами данных, относящаяся к классу NoSQL-систем и рассчитанная на создание высокомасштабируемых и надежных хранилищ огромных массивов данных, представленных в виде хэша. [2]

В качестве брокера сообщений был выбран Apache Kafka.

Kafka — это распределенная система, состоящая из серверов и клиентов, которые обмениваются данными по высокопроизводительному сетевому протоколу TCP. Его можно развернуть на «голом железе», вирту-

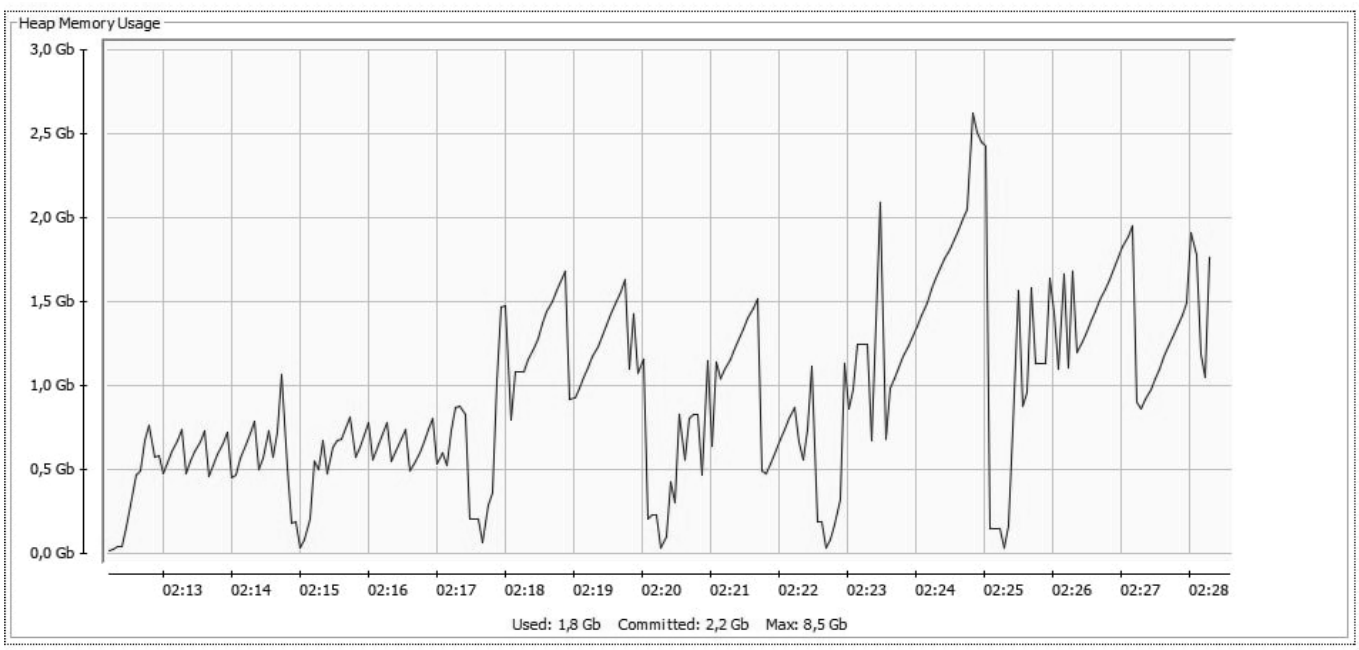

Рис 2. Пример метрики Heap Memory Usage

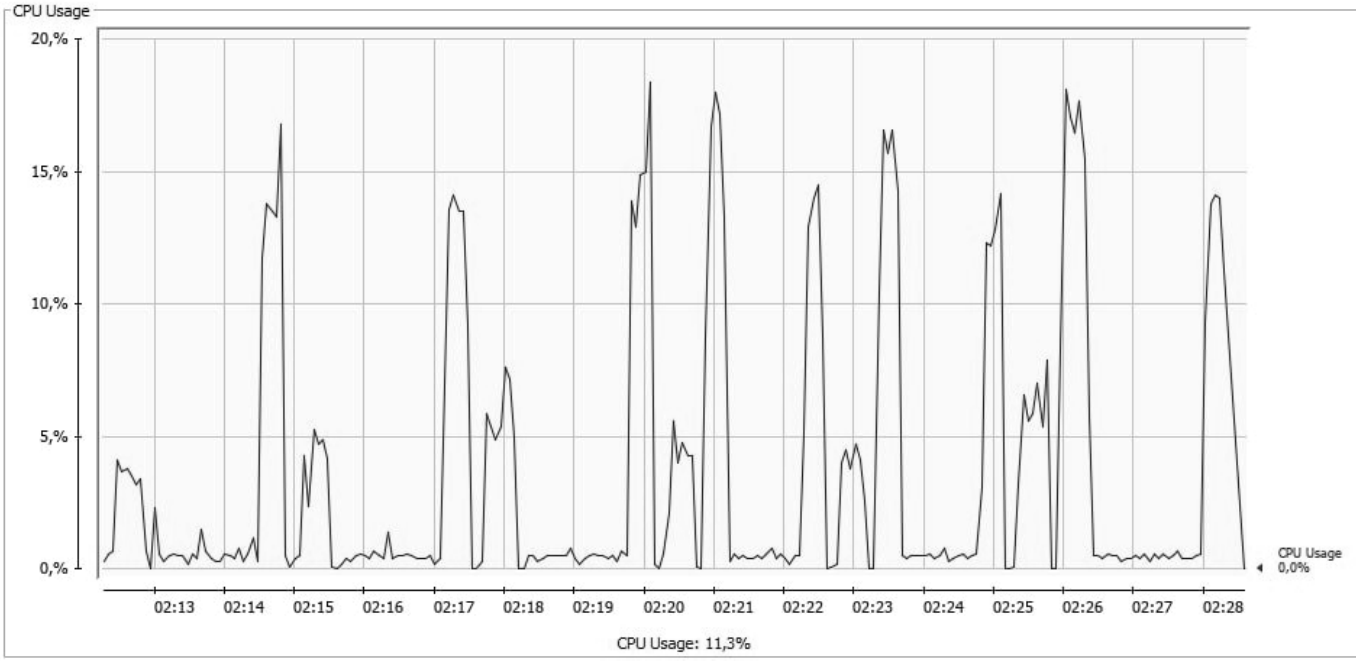

Рис. 3. Пример метрики CPU Usage

альных машинах и контейнерах как в локальной, так и в облачной среде. [3]

Архитектура тестового стенда

Тестовый стенд состоит из следующих частей:

- Генератор данных — сервис, который до начала тестирования генерирует данные по заданным условиям
- Kafka — брокер сообщений, который хранит в очередях (топиках) сгенерированные, и обработанные сообщения.
- Cassandra — NoSql база данных, которая выступает в роли персистентного хранилища данных
- Сервис дедуплицирующий сообщения — сервис, в котором реализованы методы дедупликации, он же собирает метрики о скорости, утилизации cpu и ram на своем хосте

Были реализованы следующие профили нагрузки для генерации:

1. События для каждого пользователя идут последовательно. Дубли отсутствуют. (коэффициент дублирования = 1,0)
2. События для каждого пользователя идут последовательно. Каждое 10е сообщение дублируется (коэффициент дублирования = 1,1)
3. События для каждого пользователя идут последовательно. Каждое сообщение дублируется (коэффициент дублирования = 2,0)
4. События для каждого пользователя идут в случайном порядке. Дубли отсутствуют. (коэффициент дублирования = 1,0)
5. События для каждого пользователя идут в случайном порядке. Каждое 10е сообщение дублируется (коэффициент дублирования = 1,1)
6. События для каждого пользователя идут в случайном порядке. Каждое сообщение дублируется (коэффициент дублирования = 2,0)

Данные об утилизации ресурсов собирались через утилиту Jconsole, поставляемую в дистрибутиве Java.

Для нужд тестового стенда использовалось 3 независимых машины.

Cassandra:

- CPU: 4x3.0 Ггц
- RAM 3.2 Ггц 16Гб (из которых под Cassandra выделено 8 Гб)
- SSD

kafka:

- CPU: 4x3.0 Ггц
- RAM 3.2 Ггц 16Гб (из которых под kafka выделено 8 Гб)
- SSD

Генератор сообщений + сервис для дедупликации:

- CPU: 4x3.2 Ггц
- RAM 3.2 Ггц 32Гб (из которых под каждый выделено 8 Гб

Методы дедупликации данных

Все выводы, сделанные в главе основываются на экспериментальных данных, полученных с помощью разработанного тестового стенда и реализованных методов дедупликации.

Методы для определения характеристик системы

Для того чтобы исследовать полноценные методы дедупликации данных — для начала нужно понимать какие ограничения накладываются вне этих методов. Т.е. скорость сети, CPU, памяти и т.д.

Рассмотрим “технические” методы, это те методы, которые не производят самой дедупликации, но помогают понять какие ограничения накладывают технические характеристики системы.

Для этих нужд было реализовано 3 метода:

1. Только чтение данных из kafka
2. Чтение данных из kafka и запись данных в конечный топик kafka
3. Чтение данных из кафка, запись всего набора данных в cassandra через batch-вставку, запись в конечный топик kafka

Методы для дедупликации

Дедупликация через cassandra:

Суть метода проста: при обработке события — сохраняем связку userId и eventId в таблице в БД, и при каждом новом запросе — проверяем было такое событие уже, или нет.

Преимущества метода:

- Отказоустойчивость
- Всегда можно узнать что лежит в БД и поправить это при необходимости
- Персистентность
- Любой экземпляр приложения может обрабатывать любые события
- Глубина дедупликации, то за какой период будут находиться данные ограничено только размером БД

Недостатки:

- Низкая скорость работы

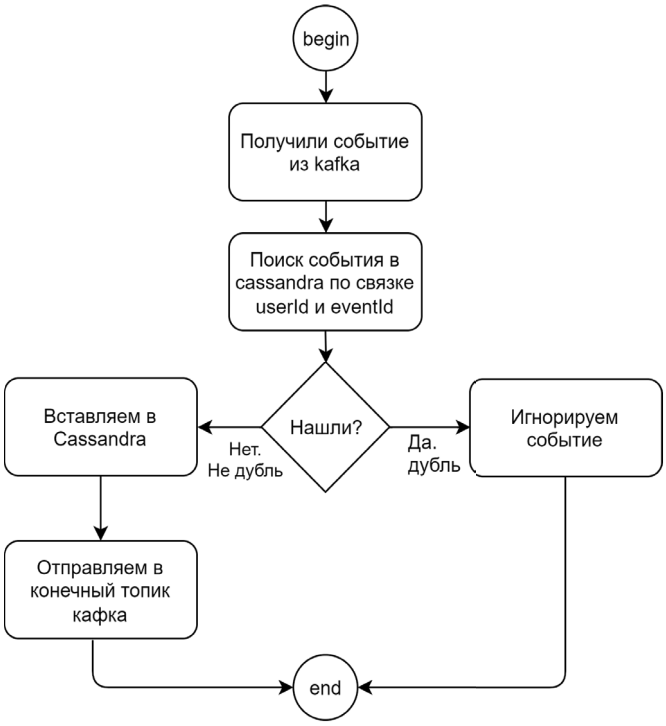

Рис 4. Алгоритм работы метода дедупликации через cassandra

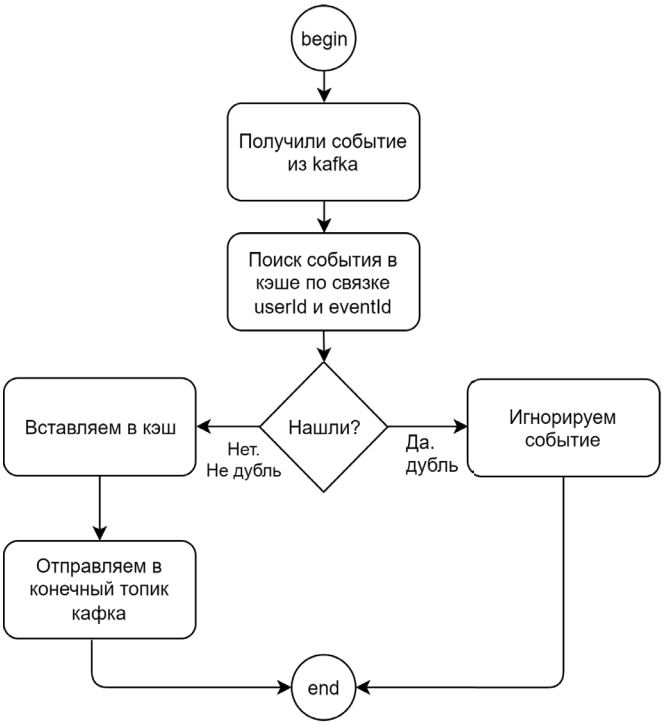

Рис 5. Алгоритм работы метода дедупликации через кэш

- Высокая нагрузка на БД (фактически на каждое событие нужно сделать один select и один insert)
- Из-за БД — сложно масштабировать горизонтально

Возможная область использования:
- Сервисы не требовательные к скорости работы
- Сервисы с необходимостью большой глубиной дедупликации (когда дубль может прийти через неделю, месяц, год)
- Сервисы с частым перезапуском (например, работающие с kubernetes)

3.2.2 Дедупликация через кэш.

Суть метода: при обработке события — сохраняем связку userId и eventId во внутренний кэш, и при каждом новом запросе — проверяем было такое событие уже, или нет.

Преимущества метода:
- Скорость работы
- Безлимитное горизонтальное масштабирование

Недостатки:
- При перезапуске кэш обнуляется.
- Нужно организовывать архитектуру таким образом, чтобы один и те же события приходили на одни и те же экземпляры приложения
- Размер кэша ограничен размером оперативной памяти сервиса

Возможная область использования:
- Сервисы требовательные к скорости
- Сервисы с небольшой глубиной дедупликации (когда дубль может прийти через минуту, час, день)
- Сервисы без частых перезапусков (например, работающие с kubernetes)

3.2.3 Дедупликация методом максимального ID события

Наиболее интересным методом является метод дедупликации через максимальный ID события.

Этот метод призван объединить достоинства предыдущих методов, и скомпенсировать их недостатки.

Суть метода: так как eventId — монотонно возрастает в рамках одного пользователя, то для того чтобы однозначно выделить “не дубли” — нам достаточно знать максимальный eventId, который мы уже обрабатывали. Если пришедшее событие имеет eventId > max_eventId, то это однозначно не дубль.

Приведем пример:

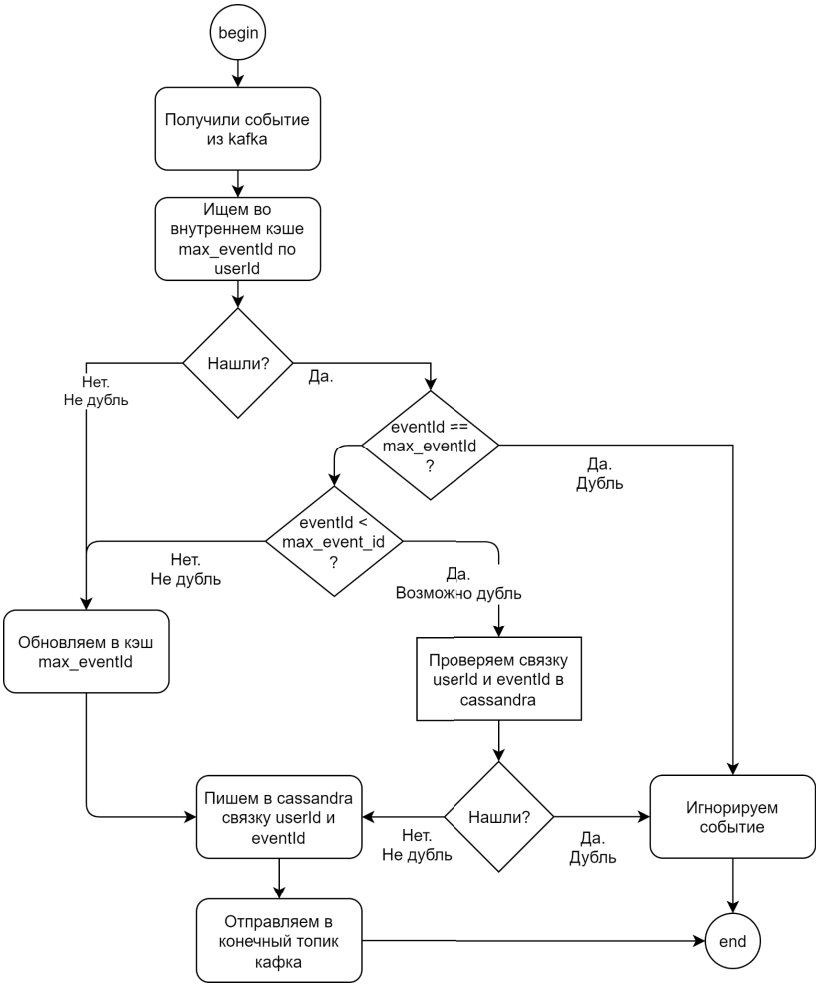

Рис. 6. Алгоритм дедупликации методом максимального eventId

В рамках одного пользователя — приходили события: с eventId =7, =10, =16

То max_eventId = 16

И если придет новое событие с eventId = 20, то мы определенно можем сказать, что это не дубль

К сожалению, в другую сторону это не работает, и мы не можем однозначно сказать что если eventId < max_eventId — дубль.

Приведу пример:

В рамках одного пользователя — приходили события: с eventId =7, =10, =16

То max_eventId = 16

И если придет новое событие с eventId = 13, то мы не можем сказать дубль это или нет, т.к. возможно что-то произошло в предыдущих сервисах, и порядок сообщений просто поменялся.

В таком случае нужно валидировать другим методом, и этот способ — cassandra.

Каждое обработанное сообщение, как и в методе с дедупликацией через cassandra — нужно сохранять в БД, но в отличие от того метода — обращаться с select`ом в БД нужно только в случае если пришедший eventId < max_eventId

Преимущества метода:

- Скорость работы
- Безлимитное горизонтальное масштабирование
- Низкое потребление RAM для кэш. Хранится только два значения Integer для каждого пользователя.
- Нагрузка на БД преимущественно только на запись
- Глубина дедупликации, то за какой период будут находиться данные ограничено только размером БД

Таблица 1. Результаты проведенных исследований методов дедупликации данных на виртуальном тестовом стенде

Метод	Профиль нагрузки, коэффициент дублирования	Событий в сек	CPU host,%	RAM host, Mb	Cassandra, cpu%	Cassandra Ram, Mb	Cassandra SSD, Mb
Только чтение из kafka	сортированный, 1.0	48 488	4	630			
	сортированный, 1.1	55 235	5	680			
	сортированный, 2.0	85 332	6	1 100			
	случайный порядок, 1.0	48 957	4	640			
	случайный порядок, 1.1	54 150	5	830			
	случайный порядок, 2.0	90 169	6	1 200			
Чтение и запись в kafka	сортированный, 1.0	33 464	15	832			
	сортированный, 1.1	36 586	13	1 634			
	сортированный, 2.0	51 058	12	1 546			
	случайный порядок, 1.0	33 332	16	1 246			
	случайный порядок, 1.1	36 011	15	1 523			
	случайный порядок, 2.0	50 186	13	1 513			
Чтение из kafka, сохранение в cassandra, запись в kafka	сортированный, 1.0	20 931	15	953	25	1 236	9,79
	сортированный, 1.1	21 831	14	1 626	27	1 257	9,79
	сортированный, 2.0	26 726	12	1 598	31	1 273	9,79
	случайный порядок, 1.0	20 581	15	1 347	26	1 241	9,79
	случайный порядок, 1.1	21 297	15	1 615	29	1 254	9,79
	случайный порядок, 2.0	25 592	13	1 702	32	1 275	9,79
Дедупликация через cassandra	сортированный, 1.0	5 238	26	6 482	35	1 928	9,79
	сортированный, 1.1	5 625	32	7 284	33	1 885	9,79
	сортированный, 2.0	5 945	74	8 496	37	1 960	9,79
	случайный порядок, 1.0	5 228	27	6 523	33	1 902	9,79
	случайный порядок, 1.1	5 615	32	7 354	34	1 945	9,79
	случайный порядок, 2.0	5 926	73	8 983	36	1 963	9,79
Дедупликация через кэш	сортированный, 1.0	33 381	16	1 815			
	сортированный, 1.1	35 362	20	1 850			
	сортированный, 2.0	56 915	16	1 794			
	случайный порядок, 1.0	32 805	14	1 950			
	случайный порядок, 1.1	34 824	15	2 053			
	случайный порядок, 2.0	55 015	18	2 463			
Дедупликация методом максимального ID события	сортированный, 1.0	18 782	16	1 024	26	1 245	9,79
	сортированный, 1.1	19 019	14	858	29	1 263	9,79
	сортированный, 2.0	24 909	18	1 613	32	1 286	9,79
	случайный порядок, 1.0	18 251	18	1 466	28	1 246	9,79
	случайный порядок, 1.1	18 539	16	2 509	30	1 265	9,79
	случайный порядок, 2.0	23 505	18	1 851	33	1 281	9,79

- Отказоустойчивость
- Персистентность

Недостатки:
- Нужно организовывать архитектуру таким образом, чтобы один и те же события приходили на одни и те же экземпляры приложения
- Требователен к качеству данных, нужно чтобы события, в большинстве своем приходили в правильном порядку

Возможная область использования:
- Сервисы требовательные к скорости
- Сервисы с любой глубиной дедупликации
- Сервисы с любым периодом перезапусков, но нужно быть готовым к тому что на этапе прогрева кэша — будет большая нагрузка на БД

Результаты

Как говорилось ранее — в рамках работы проводились измерения каждого метода дедупликации на каждом профиле нагрузки. С результатами можно ознакомится в следующей таблице:

С учетом приведенных результатов — наиболее быстрым методом является “Дедупликация через кэш”, но этот метод имеет серьезный недостаток — отсутствие персистентного хранилища, и высокая вероятность отказа. Метод дедупликации через максимальный ID — медленнее дедупликации через кэш, но при этом значительно быстрее метода дедупликации через cassandra. При этом он не имеет серьезных ограничений по применимости, и выглядит как наиболее сбалансированный из рассмотренных методов.

Заключение

В рамках данной работы был разработан виртуальный тестовый стенд, на котором проверялись различные методы дедупликации большого потока данных.

На языке java был разработан генератор тестовых данных и сервис для дедупликации данных.

Были реализованы различные методы дедупликации данных.

Созданные методы были протестированы, и по итогам тестов сделаны выводы о преимуществах и недостатках рассмотренных методов, а также была проанализирована применимость этих методов к различным областям использования.

ЛИТЕРАТУРА

1. А.Ю. Ефимов Проблемы обработки статистики сетевого трафика для обнаружения вторжений в существующих информационных системах // Программные продукты и системы. 2016. № 1 (113). URL: https://cyberleninka.ru/article/n/problemy-obrabotki-statistiki-setevogo-trafika-dlya-obnaruzheniya-vtorzheniy-v-suschestvuyuschih-informatsionnyh-sistemah (дата обращения: 26.01.2023).
2. Apache Cassandra // Википедия URL: https://ru.wikipedia.org/wiki/Apache_Cassandra (дата обращения: 26.01.2023).
3. Kafka documentation // kafka.apache.org URL: https://kafka.apache.org/documentation/#introduction (дата обращения: 26.01.2023).

Журнал «Современная наука: актуальные проблемы теории и практики»