А сколько реализаций Transactional Outbox для PostgreSQL вы знаете?

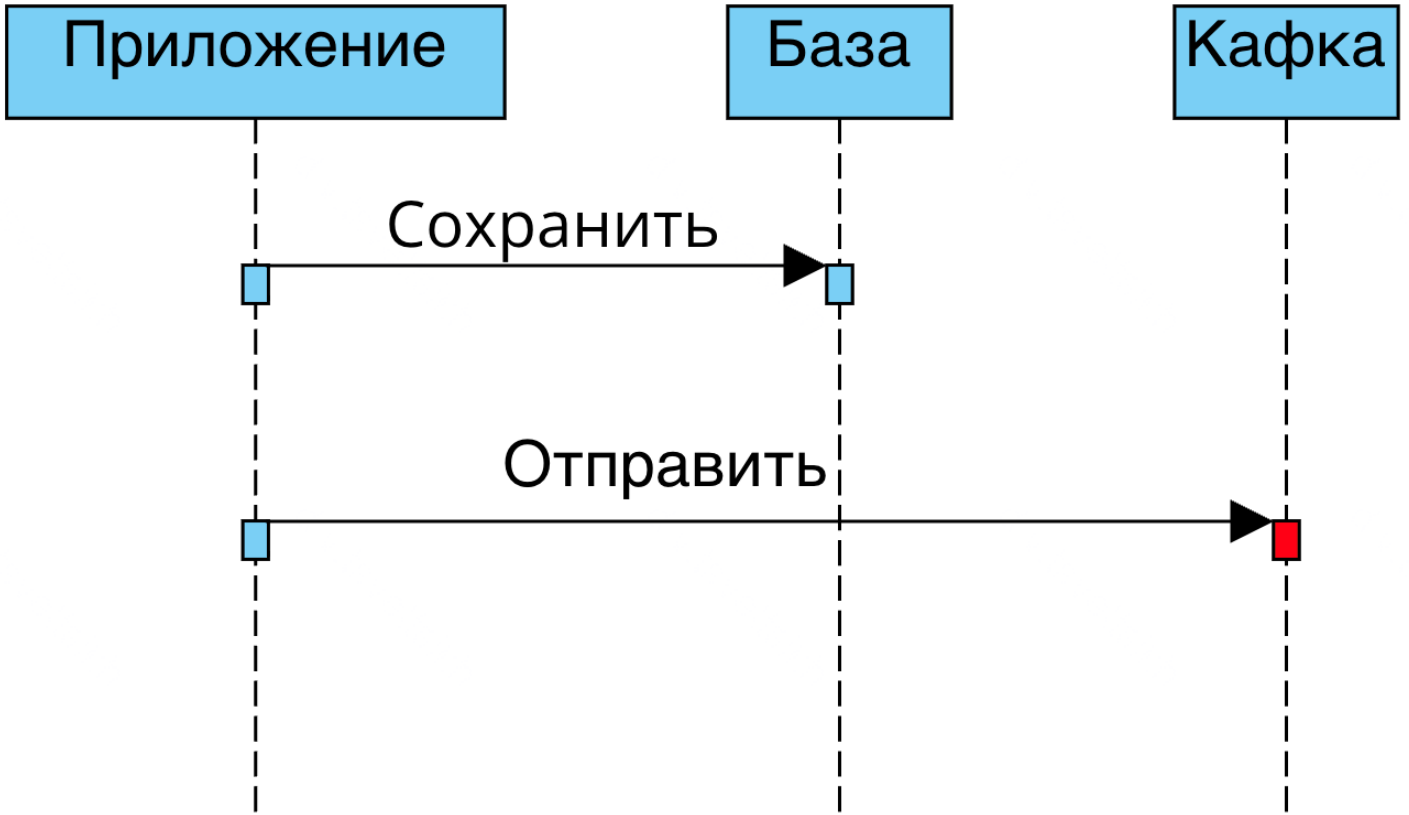
Денис Цветцих
DevBrothers

DevBrothers

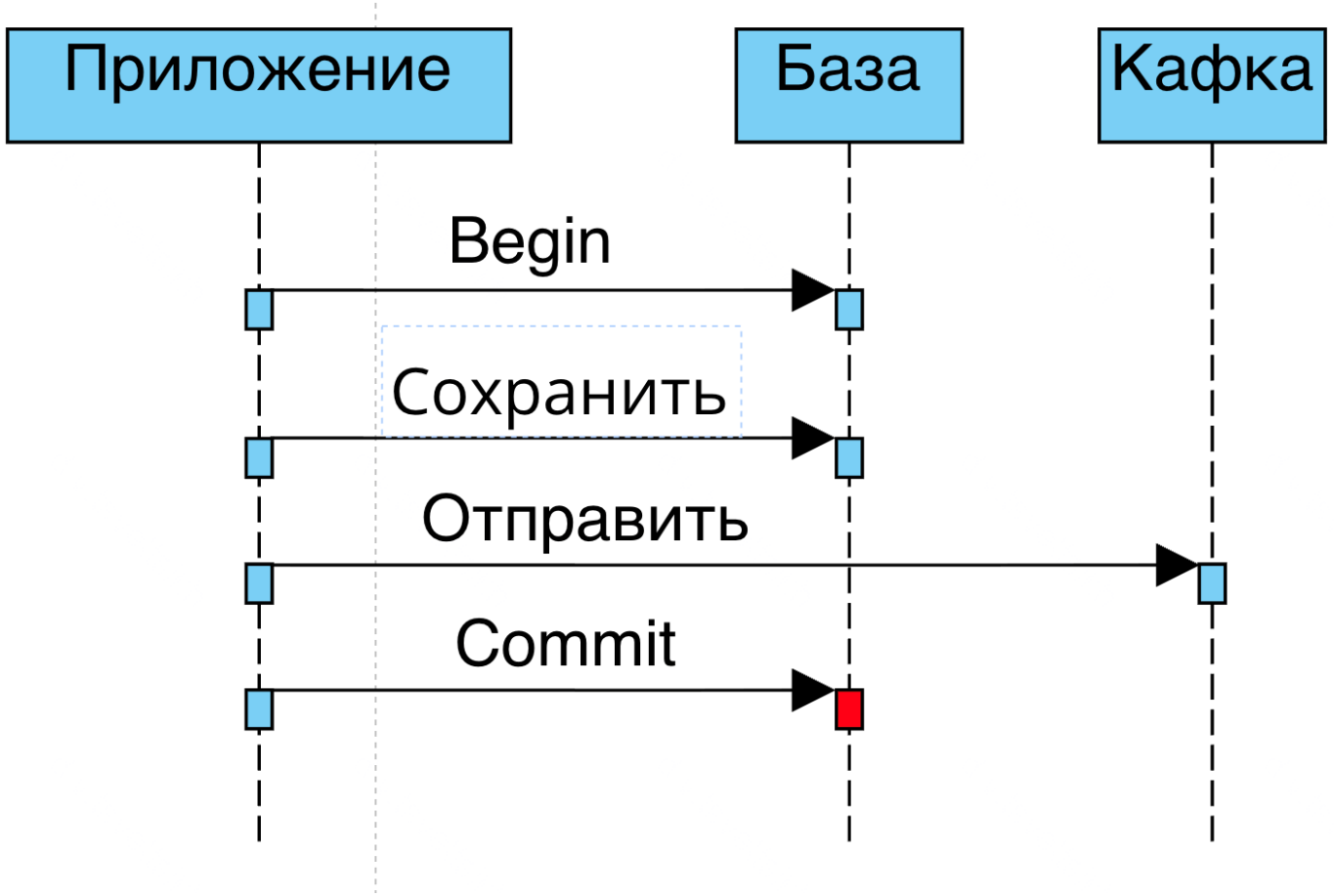
Денис Цветцих

DevBrothers

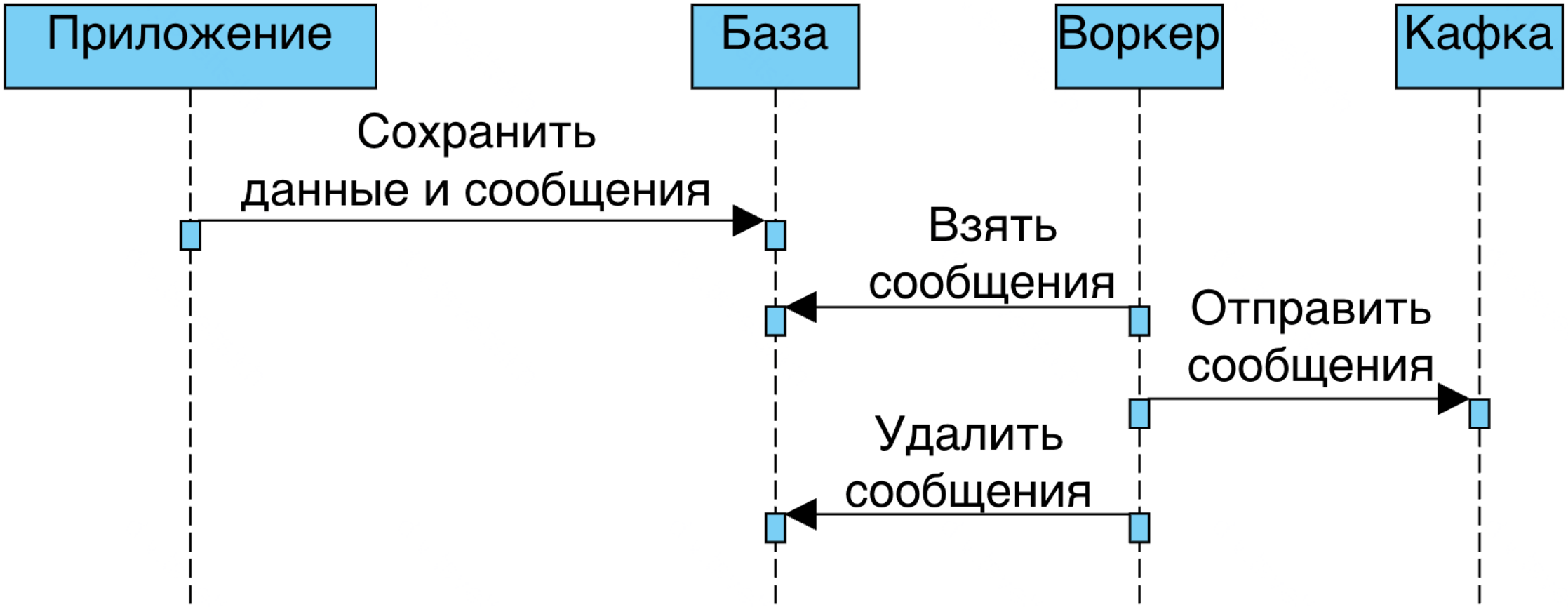
Сохранили, но не отправили



Отправили, но не сохранили



Outbox


Приложение
База
Воркер
Кафка
Сохранить
данные и сообщения
Взять
сообщения
Отправить
сообщения
Удалить
сообщения

Гарантии доставки

- at-most-once – доставили 1 раз или нет (потери)
- at-leat-once – доставили 1 и более раз (дубли)
- exactly-once – доставили строго 1 раз (основана на at-leat-once)

Стандарт X/Open XA (eXtended Architecture)

- Для 2PC (двухфазный коммит)
- Kafka и RabbitMQ не реализуют
- NoSql базы типа MongoDB и Cassandra
- А еще это долго и сложно

https://ru.wikipedia.org/wiki/XA

О чем поговорим

1. Outbox с одной длинной транзакцией
2. Outbox с двумя короткими транзакциями
3. Outbox с виртуальными партициями
4. Outbox с синхронизацией записи

Кейс – хранение в PostgreSQL и отправка в Kafka

Табличка сообщений

```
CREATE TABLE outbox_messages (
id int4 GENERATED BY DEFAULT AS IDENTITY() NOT NULL,
topic varchar(128) NOT NULL,
key varchar(128) NULL,
type varchar(128) NOT NULL,
payload jsonb NOT NULL,
headers jsonb NOT NULL,
created_at timestamptz DEFAULT now() NOT NULL,
CONSTRAINT pk_outbox_messages PRIMARY KEY (id)
);
```

Вариант 1 – длинная транзакция

Блокировка отправляемых сообщений

```
BeginTransaction()

messages = GetPendingMessagesForUpdateSkipLocked()

if (messages.Any())
    SendBatchToKafka(messages)
    DeleteMessages(messages)
    CommitTransaction()
else
    CommitTransaction()
    sleep (5 sec)
```

Блокировка отправляемых сообщений

```
--roundtrip 1
BEGIN;


--roundtrip 2
SELECT * FROM outbox_messages
ORDER BY id
LIMIT $batch_size
FOR UPDATE SKIP LOCKED;


--roundtrip 3
DELETE FROM outbox_messages
WHERE id IN ($messsage_ids);


--roundtrip 4
COMMIT;
```

Что получили

Плюсы
- Несколько воркеров

Минусы
- отправка в кафку внутри транзакции (длинная транзакция)
- блокировка строк
- мутация строк (автовакуум)
- нет порядка сообщений

Нагрузка на приложение

- CPU: Outbox – перекладывание JSON, нагрузки не будет
- Память – много короткоживущих объектов, пылесосятся сборщиком мусора
  - Оптимизация памяти - не использовать тяжелые ORM

Нагрузка на базу

- 1 000 000 сообщений
- 100 сообщений в батче
- Сообщение 256 байт
- 2 топика по 1 партиции

CPU базы 1 воркер

CPU базы 2 воркера

Скорость отправки

	1 воркер (mm:ss)		2 воркера	
1. Одна длинная	01:53	1	01:00	1

Вариант 1.1 – рекомендательная блокировка

Порядок сообщений

lock = GetAdvisoryLock()

messages = GetPendingMessages()

```
if (messages.Any())
    SendBatchToKafka(messages)
    DeleteMessages(messages)
    lock.Release()
else
    lock.Release()
    sleep (5 sec)
```

AdvisoryLock – без длинной транзакции

```
--roundtrip 1
SELECT pg_catalog.pg_advisory_lock(-1613257171, -31457281);


--roundtrip 2
SELECT * FROM outbox_messages
ORDER BY id
LIMIT $batch_size;


--roundtrip 3
DELETE FROM outbox_messages
WHERE id IN ($messsage_ids);


--roundtrip 5
SELECT pg_catalog.pg_advisory_unlock(-1613257171, -31457281);
```

AdvisoryLock – pg_bouncer session mode

- pg_bouncer в сеансовом режиме
  - иначе не разблокируем

https://habr.com/ru/companies/karuna/articles/674730/

Итоги

- Плюсы
  - нет блокировки строк
  - нет длинной транзакции
  - абсолютный порядок сообщений

- Минусы
  - мутация строк осталась
  - advisory lock держит подключение (ничего не выиграли)
  - отправка одним воркером

Вариант 2 – короткие транзакции

Колонка для пессимистической блокировки

```
CREATE TABLE outbox.outbox_messages (
id int4 GENERATED BY DEFAULT AS IDENTITY() NOT NULL,
topic varchar(128) NOT NULL,
"key" varchar(128) NULL,
"type" varchar(128) NOT NULL,
payload jsonb NOT NULL,
headers jsonb NOT NULL,
created_at timestamptz DEFAULT now() NOT NULL,
available_after timestamptz DEFAULT now() NOT NULL,
CONSTRAINT pk_outbox_messages PRIMARY KEY (id)
);

CREATE INDEX ix_outbox_messages_available_after ON outbox.outbox_messages USING
btree (available_after);
```

Две короткие транзакции

```
messages = GetPendingMessagesAndLock(now() + lockTimeout)

if (messages.Any())
    SendBatchToKafka(messages)
    DeleteMessages(messages)
else
    sleep (5 sec)
```

Блокировка сообщений

```
--roundtrip 1
BEGIN;

--roundtrip 2
SELECT id
FROM outbox_messages
WHERE available_after < now()
ORDER BY available_after
LIMIT $batch_size
FOR UPDATE SKIP LOCKED;

--roundtrip 3
UPDATE outbox_messages
SET available_after = $now_plus_lock_timeout
WHERE id IN ($messsage_ids);

--roundtrip 4
COMMIT;
```

Блокировка одним запросом

```
--roundtrip 1
UPDATE outbox_messages
SET available_after = $now_plus_lock_timeout
FROM
(
   SELECT id
   FROM outbox_messages
   WHERE available_after < now()
   ORDER BY available_after
   LIMIT $batch_size
   FOR UPDATE SKIP LOCKED
) locked_messages
WHERE
outbox_messages.id = locked_messages.id
RETURNING
outbox_messages.*;
```

Апгрейд

- Сохранять еще и какой сервис заблокировал
- И настроить сохранение имени при убийстве пода StatefullSet
- Тогда он может отправить сообщения после перезапуска, остальные не ждут таймаута

Что получили

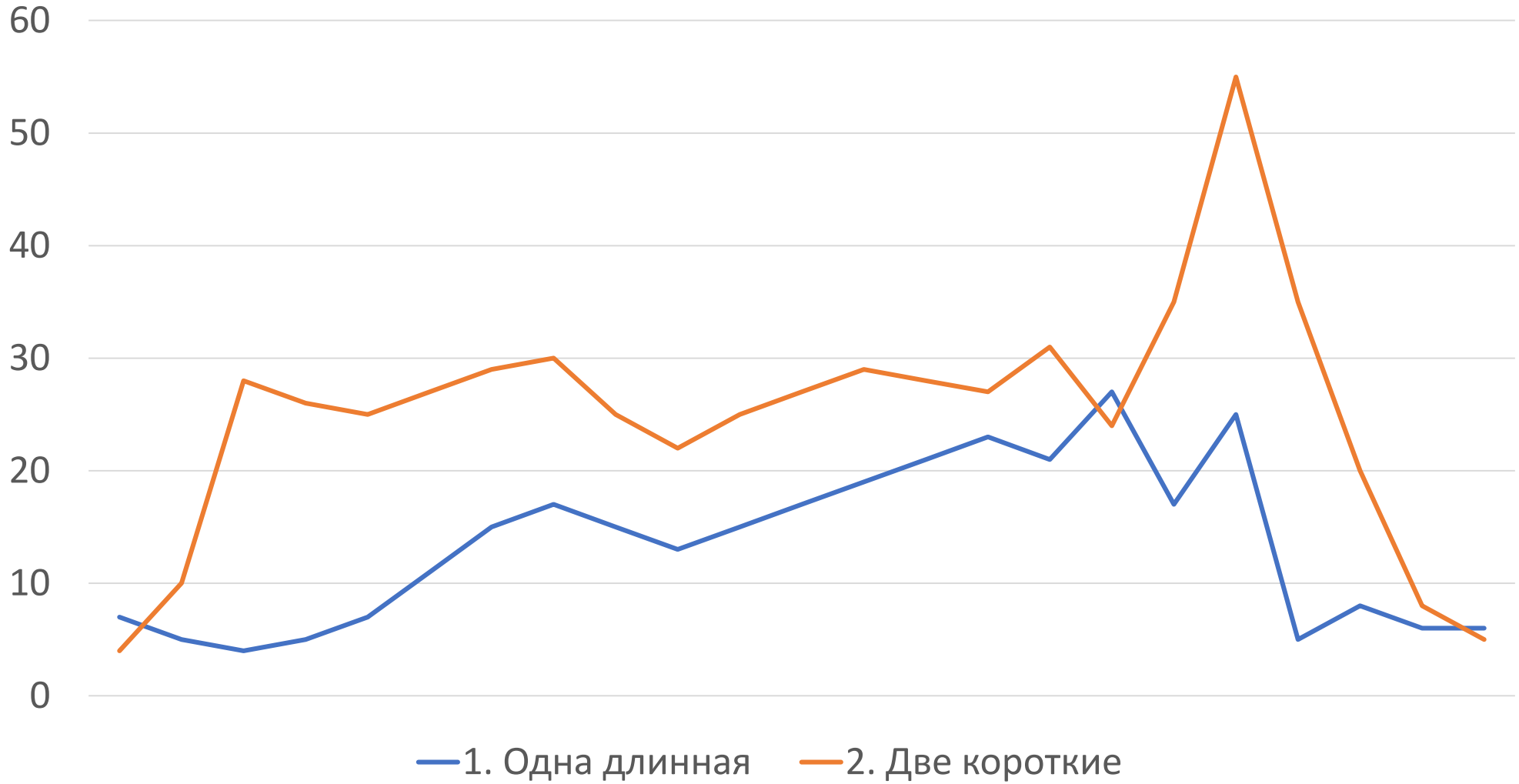
Плюсы

- короткие транзакции

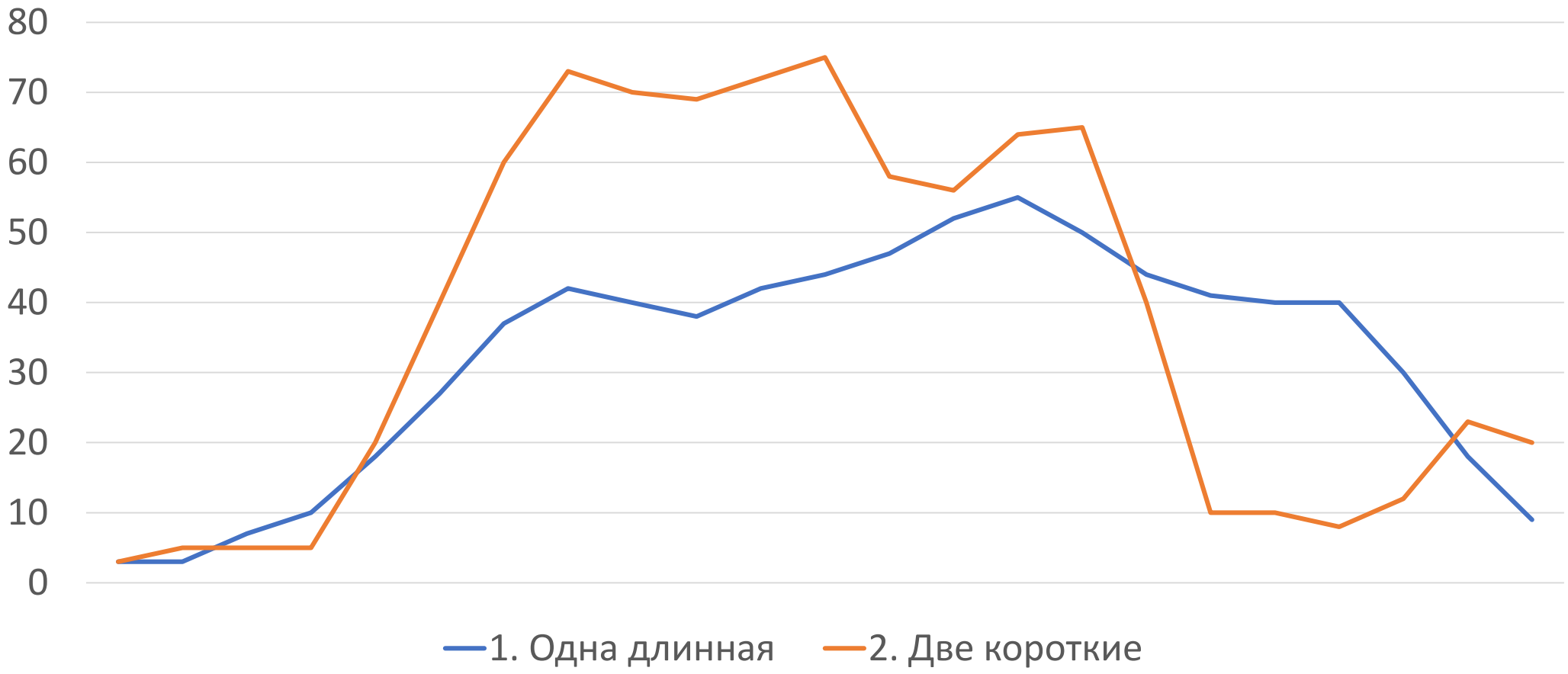
Минусы

- мутация строк x2 (автовакуум x2)
- нет порядка сообщений

CPU базы 1 воркер



CPU базы 2 воркера



Скорость отправки

	1 воркер (mm:ss)		2 воркера	
1. Одна длинная	01:53	1	01:00	1
2. Две короткие	**02:08**	**1.13**	**01:17**	**1.28**

Чего хочется

- Нужен не абсолютный порядок, в только внутри агрегата
- Несколько воркеров
- Короткие транзакции
- Не блокируем и не меняем сообщения
- Не пропускаем сообщения

Вариант 3 – виртуальные партиции

Пропуски из-за длинных транзакций

```
SELECT *
FROM outbox_messages
WHERE
id > $last_processed_id
ORDER BY id
LIMIT $batch_size;
```

Счетчик транзакций для батчинга

- 1000 завершенная
- 1001 незавершенная
- 1002 завершенная

- tx_id < min_current_tx_id – не будет пропусков

https://event-driven.io/en/ordering_in_postgres_outbox/

xmin

- xmin (xid) – транзакция, где создана строка
- xid нельзя сравнить с xid или bigint, нужно конвертировать
  - индекс не создать
- а еще xid – 32 разрядный, сбрасывается каждые 4 млрд транзакций
  - в Pro версии – 64 разрядный
- счетчик увеличивается только при записи

```
SELECT xmin FROM outbox_messages
WHERE xmin::text::bigint > txid_snapshot_xmin(txid_current_snapshot());
```

64 битный счетчик

- xid8 – новый тип с оператором сравнения
- pg_current_xact_id() – текущая транзакция
- pg_snapshot_xmin(pg_current_snapshot()) – min транзакция в процессе

```
SELECT transaction_id FROM outbox_messages
WHERE transaction_id > pg_snapshot_xmin(pg_current_snapshot());
```

Виртуальные партиции

```
CREATE TABLE outbox_offsets (
id int4 GENERATED BY DEFAULT AS IDENTITY() NOT NULL,
last_processed_id int4 NOT NULL,
available_after timestamptz DEFAULT now() NOT NULL,
last_processed_transaction_id xid8 NOT NULL,
"partition" int4 NOT NULL,
topic varchar(128) NOT NULL,
CONSTRAINT pk_outbox_offsets PRIMARY KEY (id)
);

CREATE UNIQUE INDEX ix_outbox_offsets_topic_partition
ON outbox_offsets
USING btree (topic, partition);
```

Id транзакции в сообщении

```
CREATE TABLE outbox_messages (
id int4 GENERATED BY DEFAULT AS IDENTITY() NOT NULL,
topic varchar(128) NOT NULL,
"key" varchar(128) NULL,
"type" varchar(128) NOT NULL,
payload jsonb NOT NULL,
headers jsonb NOT NULL,
created_at timestamptz DEFAULT now() NOT NULL,
"partition" int4 DEFAULT 0 NOT NULL,
transaction_id xid8 DEFAULT pg_current_xact_id() NOT NULL,
CONSTRAINT pk_outbox_messages PRIMARY KEY (id)
);
CREATE INDEX ix_outbox_messages_topic_partition_transaction_id_id ON outbox_messages USING btree
(topic, partition, transaction_id, id);
```

Виртуальные партиции

```
partition = GetAvailablePartitionAndLock(now() + lockTimeout)
if (partition == null)
        sleep (5 sec)
        return

messages = GetMessagesForPartition(partition)
if (messages.Any())
    SendBatchToKafka(messages)
    UpdatePartition(messages.Last(), now())
else
    UpdatePartition(now() + noMessagesTimeout)
```

Давно не проверенная партиция

```
UPDATE outbox_offsets
SET available_after = $now_plus_lock_timeout
FROM
(
   SELECT id
   FROM outbox_offsets
   WHERE available_after < now()
   ORDER BY available_after
   LIMIT 1
   FOR UPDATE SKIP LOCKED
) locked_offsets
WHERE
outbox_offsets.id = locked_offsets.id
RETURNING
outbox_offsets.*;
```

Сообщения для партиции

```
SELECT *
FROM outbox_messages
WHERE
topic = $topic AND
partition = $partition AND
transaction_id > $last_processed_transaction_id AND
(
   transaction_id > $last_processed_transaction_id
   OR
   transaction_id = $last_processed_transaction_id AND id > $last_processed_id
) AND
transaction_id < pg_snapshot_xmin(pg_current_snapshot())
ORDER BY transaction_id, id
LIMIT $batch_size;
```

Happy path

```
--roundtrip 1
UPDATE FROM SELECT outbox_offsets;

--roundtrip 2
SELECT * FROM outbox_messages
WHERE {condition};

--roundtrip 3
UPDATE outbox_offsets
SET
available_after = now(),
last_processed_transaction_id = $last_message_transaction_id,
last_processed_id = $last_message_id
WHERE id = $id;
```

Error flow

```
--roundtrip 1
UPDATE FROM SELECT outbox_offsets …;

--roundtrip 2
SELECT * FROM outbox_messages
WHERE {condition};

--roundtrip 3
UPDATE outbox_offsets
SET
available_after = $now_plus_timeout
WHERE id = $id;
```

Partition для сообщения

- Topic и Partition можно объединить в GroupId
- Инициализировать при создании сообщения
  - Хеш от ключа % кол-во партиций в топике
  - Свое правило

Кафка транзакции для порядка сообщений

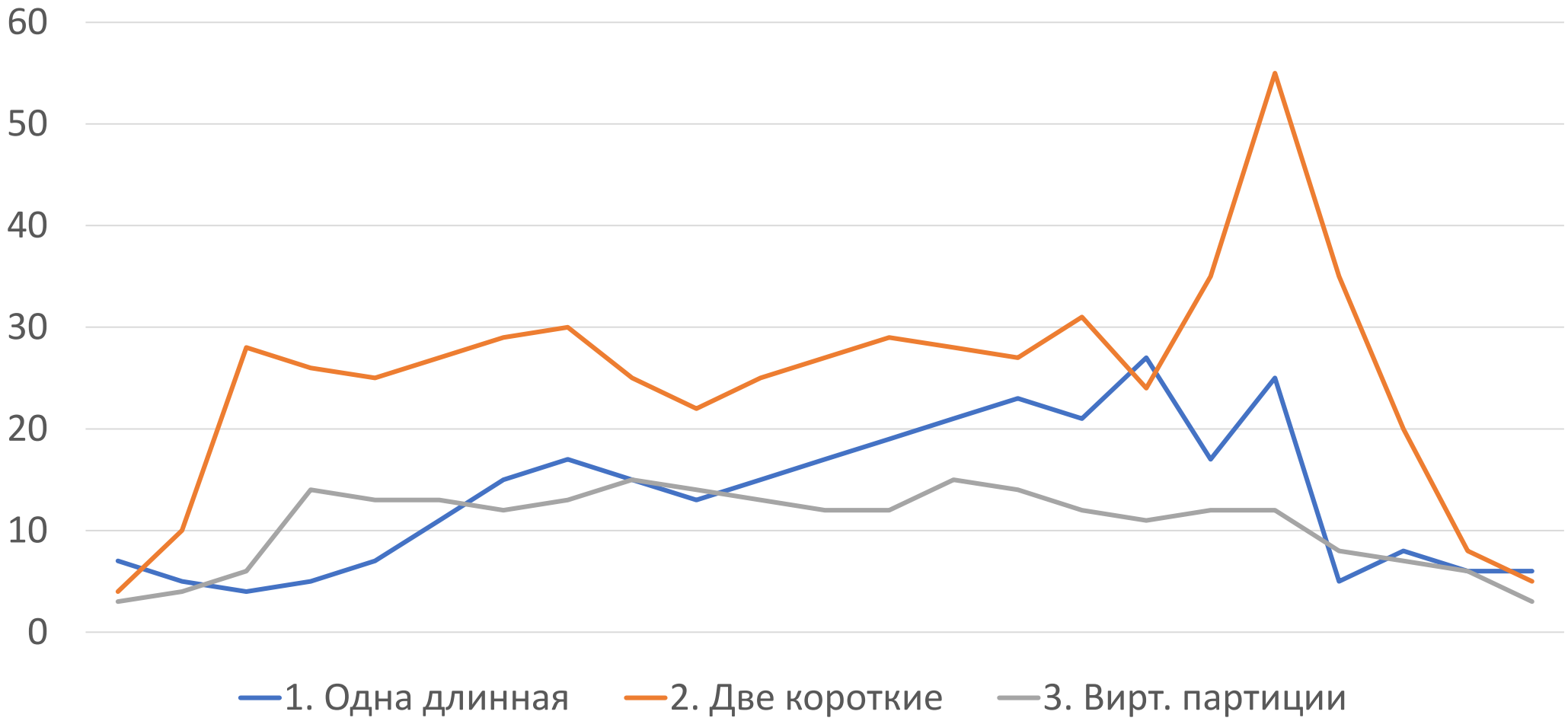
```
partition = GetAvailablePartitionAndLock(now() + lockTimeout)
if (partition == null)
        sleep (5 sec)
        return

messages = GetMessagesForPartition(partition)
if (messages.Any())
    StartKafkaTransaction()
    SendBatchToKafka(messages)
    CommitKafkaTransaction()
    UpdatePartition(messages.Last(), now())
else
    UpdatePartition(now() + noMessagesTimeout)
```

Что получили

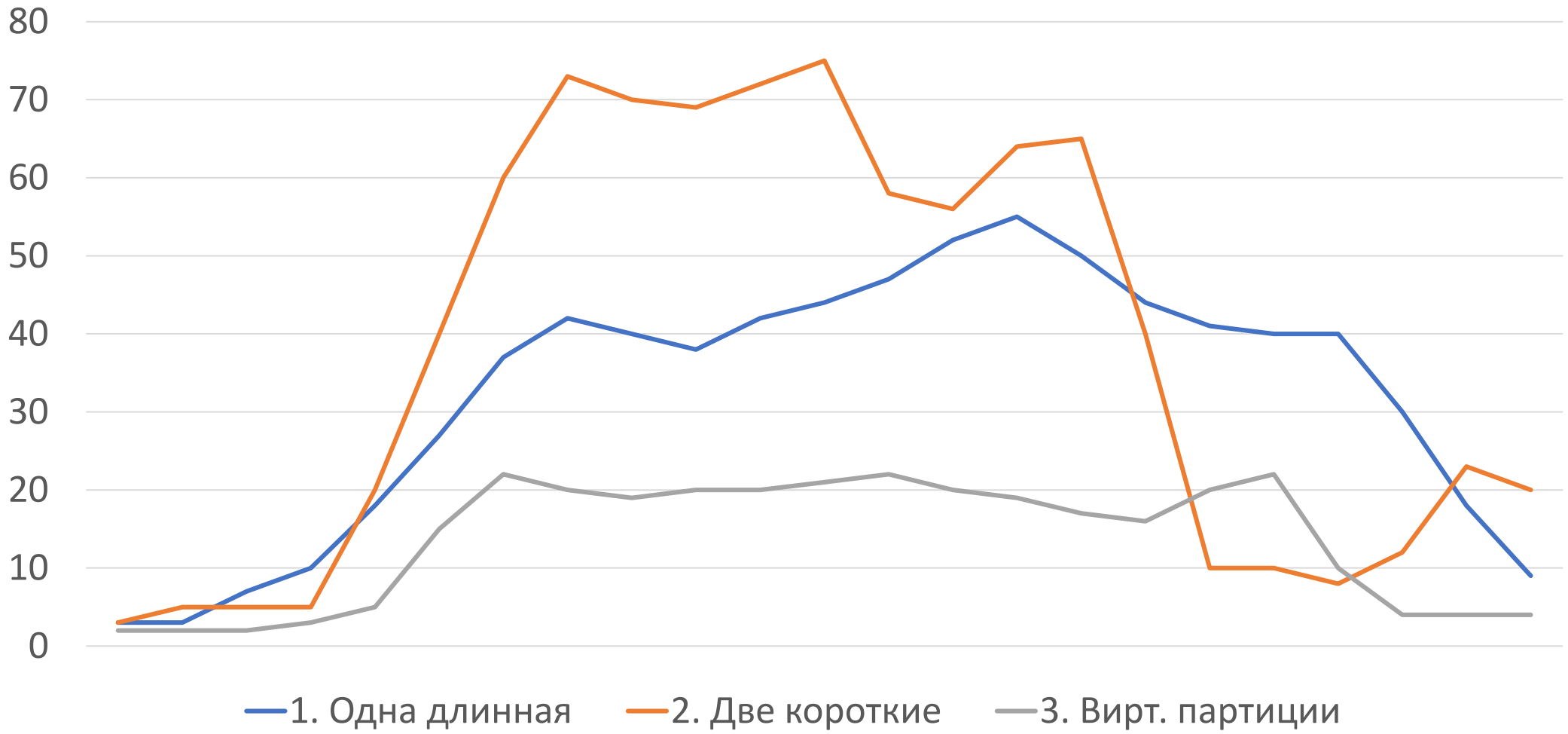
Плюсы
- Порядок внутри виртуальной партиции
- Несколько воркеров
- Короткие транзакции
- Не блокируем и не меняем сообщения
- Не пропускаем сообщения
- При ошибках может встать колом только одна партиция

Минусы
- длинная транзакция блокирует отправку из всех партиций
  - может быть критично для приоритетных сообщений
  - но их лучше отправлять сразу, а не ждать outbox
- сложное решение

CPU базы 1 воркер



CPU базы 2 воркера



Скорость отправки

	1 воркер (mm:ss)		2 воркера	
1. Одна длинная	01:53	1	01:00	1
2. Две короткие	02:08	1.13	01:17	1.28
3. Вирт. партиции	**02:36**	**1.38**	**01:07**	**1.12**

Вариант 4 –
без длинных транзакций

Избавляемся от влияния длинной транзакции

- Синхронизируем запись в таблицу outbox_messages
- Вставка в таблицу outbox_messages в конце транзакции
  - Убираем из тяжелого ORM с трекингом изменений
- Получаем id сообщения в порядке выполнения транзакций
  - Но возможно с пропусками при откате транзакций

Было: данные и сообщения

```
unitOfWork.Add(entity1);
unitOfWork.Add(message1);

unitOfWork.Add(entity2);
unitOfWork.Add(message2);

unitOfWork.SaveChanges()
```

Стало: отдельно данные и сообщения

```
unitOfWork.Add(entity1);
messageRepository.Add(message1);

unitOfWork.Add(entity2);
messageRepository.Add(message2);

beginTransaction()

unitOfWork.SaveChanges() // without messages

messageRepository.SaveChanges()

commitTransaction()
```

Было без Unit of Work

```
beginTransaction()

entityRepository.Insert(entity1);
messageRepository.Insert(message1);

entityRepository.Insert(entity2);
messageRepository.Insert(message2);

commitTransaction()
```

Стало без Unit of Work

```
beginTransaction()

entityRepository.Insert(entity1);
entityRepository.Insert(entity2);

messageRepository.Insert(message1);
messageRepository.Insert(message2);

commitTransaction()
```

Стало

```
--roundtrip 1
BEGIN;

--roundtrip 2
INSERT/UPDATE/DELETE business_data;

--roundtrip 3
SELECT pg_advisory_xact_lock(554738281823524); --transactional

INSERT INTO outbox_messages
(topic, partition, type, key, payload, headers)
VALUES
($topic, $partition, $type, $key, $payload, $headers);

--roundtrip 4
COMMIT;
```

Стало

```
--roundtrip 1
BEGIN;

--roundtrip 2
INSERT/UPDATE/DELETE business_data;

--roundtrip 3
LOCK TABLE outbox_messages IN ROW EXCLUSIVE MODE;

INSERT INTO outbox_messages
(topic, partition, type, key, payload, headers)
VALUES
($topic, $partition, $type, $key, $payload, $headers);

--roundtrip 4
COMMIT;
```

Сообщения для партиции

```
SELECT *
FROM outbox_messages
WHERE
 topic = $topic AND
 "partition" = $partition AND
 id > $last_processed_id
ORDER BY id
LIMIT $batch_size;
```

Что получили

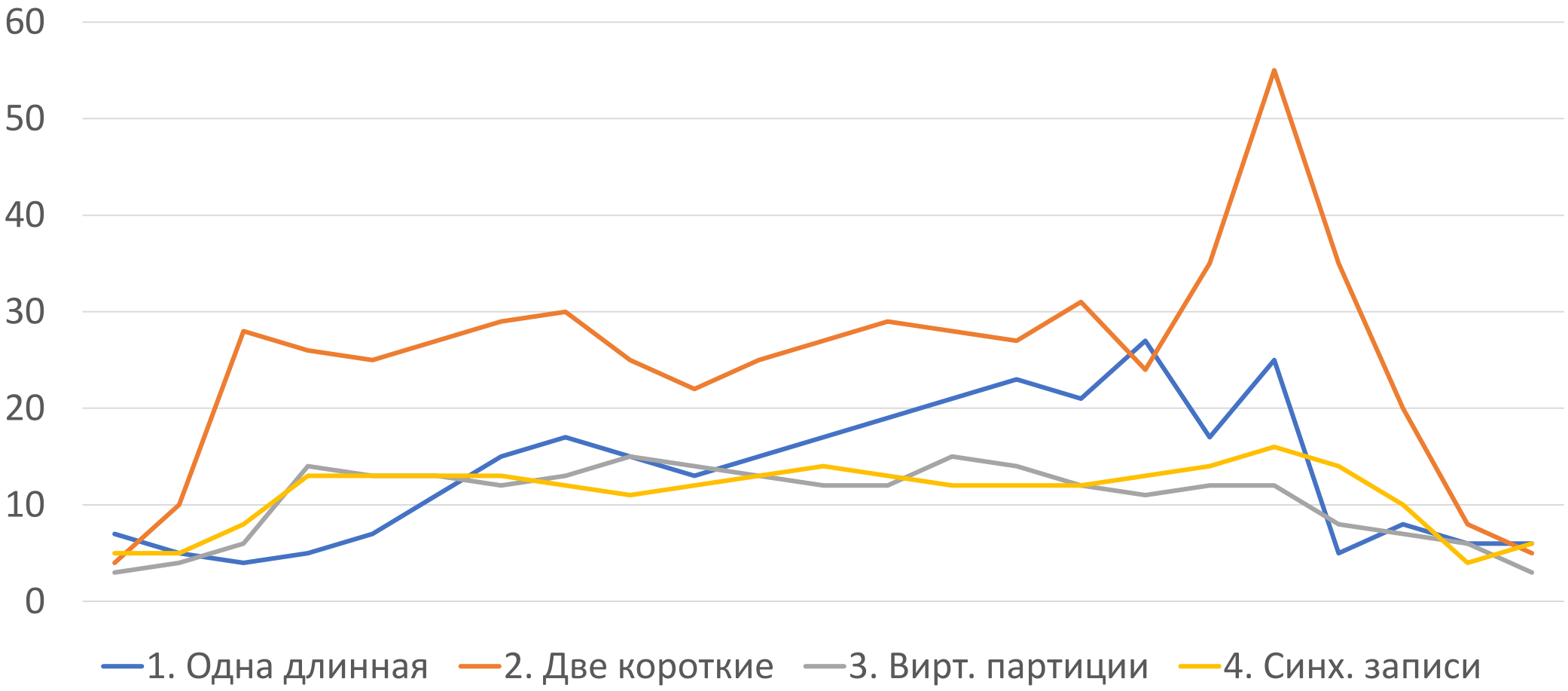
Плюсы

- Чтение не зависит от длинных транзакций

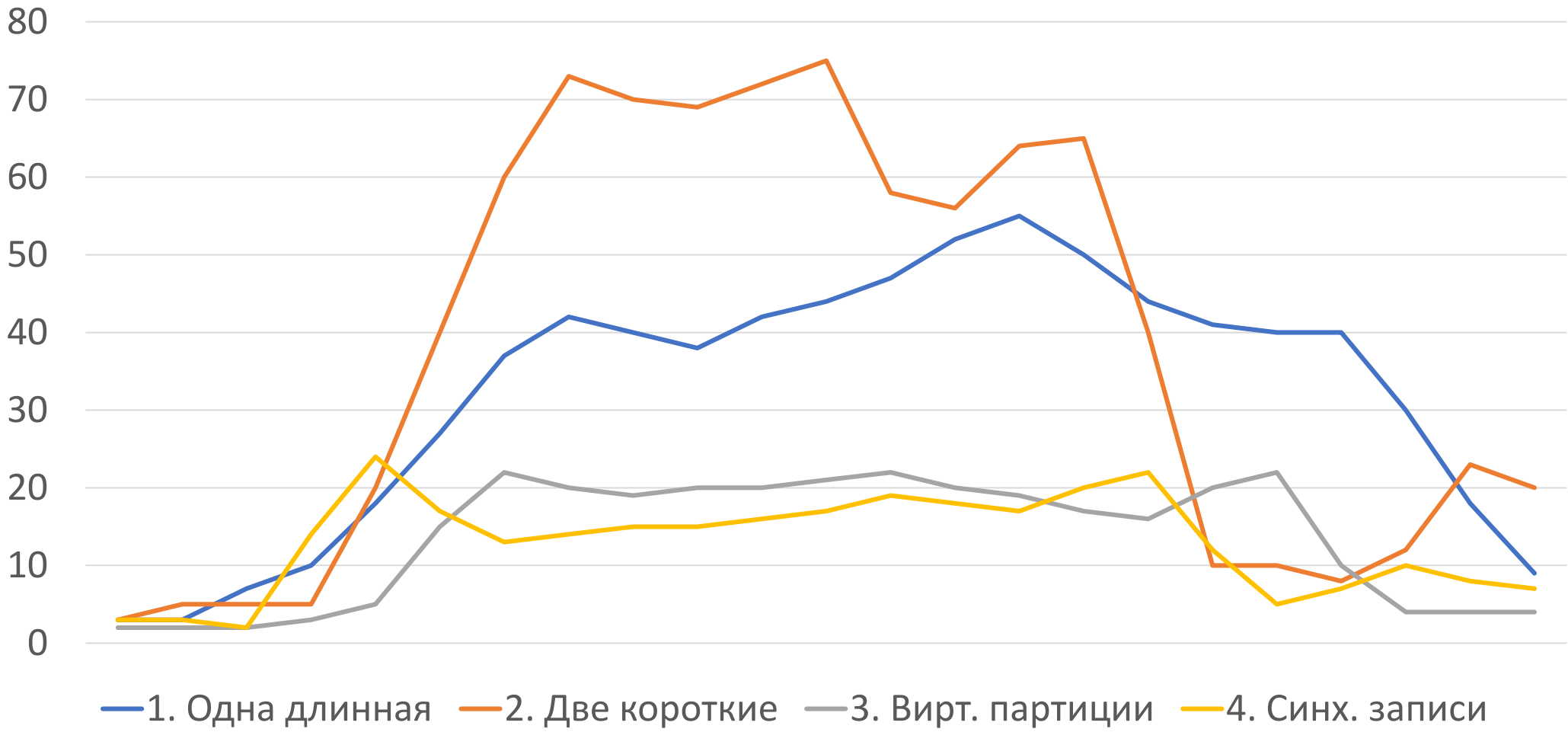
Минусы

- сложность переехала на запись во всех юскейсах
  - можно дотюнить тяжелый ORM
- Длинная транзакция блочит запись сообщений
  - снижаем влияние, записывая сообщения в конце

CPU базы 1 воркер



CPU базы 2 воркера



Скорость отправки

	1 воркер (mm:ss)		2 воркера	
1. Одна длинная	01:53	1	01:00	1
2. Две короткие	02:08	1.13	01:17	1.28
3. Вирт. партиции	02:36	1.38	01:07	1.12
4. Синх. записи	**01:58**	**1.04**	**0:57**	**0.95**

В одном процессе или в нескольких?

- В одном
  - sleepUntilReset (5 sec) – можно сразу отправить
  - 20 сервисов - 20 подключений, это для постгреса неочень
  - Quartz: 2 воркера на 20 сервисов – излишнее усложнение

- В разных
  - PosgtreSQL Notify для уведомления о новых сообщениях
  - излишнее усложнение, лучше просто ждать 5 сек
    https://postgrespro.ru/docs/postgresql/current/sql-notify

Что если есть приоритет сообщений

- Сразу делать попытку отправки, не ждать аутбокс

- State based: две таблицы, на приоритетные отдельный воркер
- State based: приоритет сообщения

- Offset based: приоритет виртуальных партиций

Если что-то пошло не так

- Если блокируем каждую строку – можно обновить пропущенные и ретраить только их
- для оффсета
  - ретраим весь батч (если он не весь отвалился)
  - или отправляем в отдельную таблицу (Dead Letter Queue)

Партиции для сообщений

- Range-партиции по дате создания сообщения, которые удаляются раз в день/неделю/…
- легко удалять

Мой опыт

- 1 вариант – универсальный
  - средне грузит базу
  - не зависит от длинных транзакций
- 3 вариант – снизить нагрузку на базу
  - в конце месяца отправка по всей базе
  - проблема с длинной транзакцией неактуальна
- 2 вариант – если нет проблем с нагрузкой на базу

- 4 вариант – слишком много возни с записью

Главные мысли

- Идеального outbox нету, либа для всех вряд ли появится
- Надо знать матчасть (хранилище)
- При оптимизации не бывает чудес
  - или длинные транзакции, или в два раза больше мутаций
  - или блокировки на чтении, или блокировки на записи

Полезные ссылки

- Как не пропускать строки
  https://event-driven.io/en/ordering_in_postgres_outbox/

Спасибо за внимание!
Есть вопросы?

Денис Цветцих
TechLead
DevBrothers


@den_tsvettsikh


den.tsvettsih@yandex.ru


https://github.com/denis-tsv