Как писать в Apache Ignite быстро

Григорий Доможиров

Joom

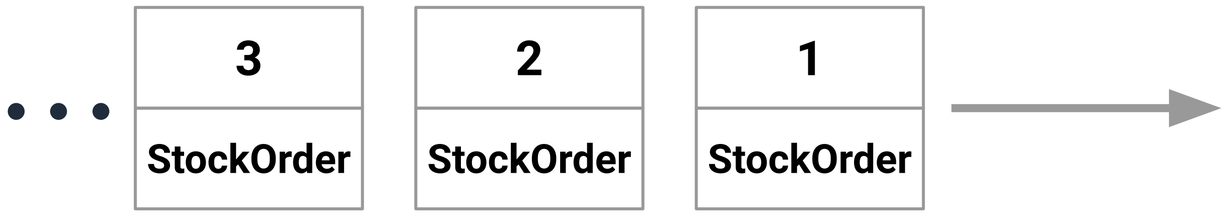
Задача

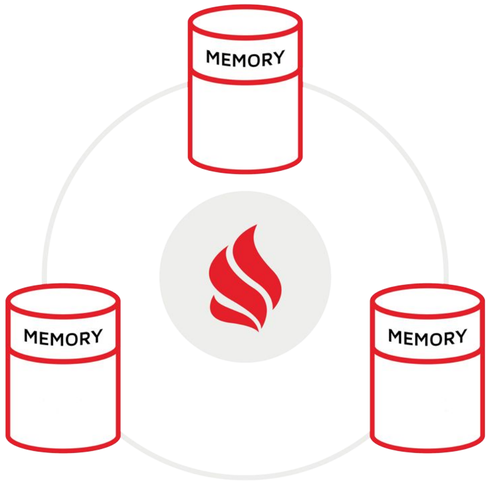
Записать в in-memory Ignite cache поток данных с максимальным writes per second

StockOrder	
secCode	SBER
price	162.93
quantity	2000
buySell	B
secBoard	TQBR
account	S01-12345678
user	MU1234567890
currency	RUB
commission	0.13
status	O
marketMaker	false

Задача

Записать в in-memory Ignite cache поток данных с максимальным writes per second





Keys	Values
1	StockOrder
2	StockOrder
3	StockOrder
4	StockOrder

План

I API и подводные камни

II Влияние различных параметров на скорость

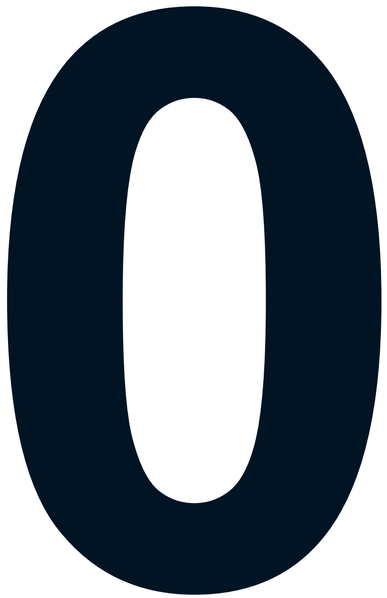
III Трюки

Формализация

```
void write(List<Entry<Long, StockOrder>> data, IgniteCache<Long, StockOrder> cache);
```

```
public class StockOrder {
  private String secCode = "SBER";
  private Double price = 162.93;
  private Integer quantity = 2000;
  private Character buySell = 'B';
  private String secBoard = "TQBR";
  private String account = "S01-12345678";
  private String user = "MU1234567890";
  private String currency = "RUB";
  private Double commission = 0.13;
  private Character status = 'O';
  private Boolean marketMaker = false;
  ...
}
```

put



0. put

```
private IgniteCache<Long, StockOrder> getDefaultCache() {
  return ignite.createCache("benchmark");
}
```

0. put

```
private IgniteCache<Long, StockOrder> getDefaultCache() {
  return ignite.createCache("benchmark");
}
```

```
@Override
public void write(List<Entry<Long, StockOrder>> data, IgniteCache<Long, StockOrder> cache) {
  data.forEach(entry -> cache.put(entry.getKey(), entry.getValue()));
}
```

AWS EC2 c6i.xlarge: 4 vCPU, up to 12.5 Gbps

5 server nodes

0. put

```
private IgniteCache<Long, StockOrder> getDefaultCache() {
  return ignite.createCache("benchmark");
}
```

```
@Override
public void write(List<Entry<Long, StockOrder>> data, IgniteCache<Long, StockOrder> cache) {
  data.forEach(entry -> cache.put(entry.getKey(), entry.getValue()));
}
```

AWS EC2 c6i.xlarge: 4 vCPU, up to 12.5 Gbps

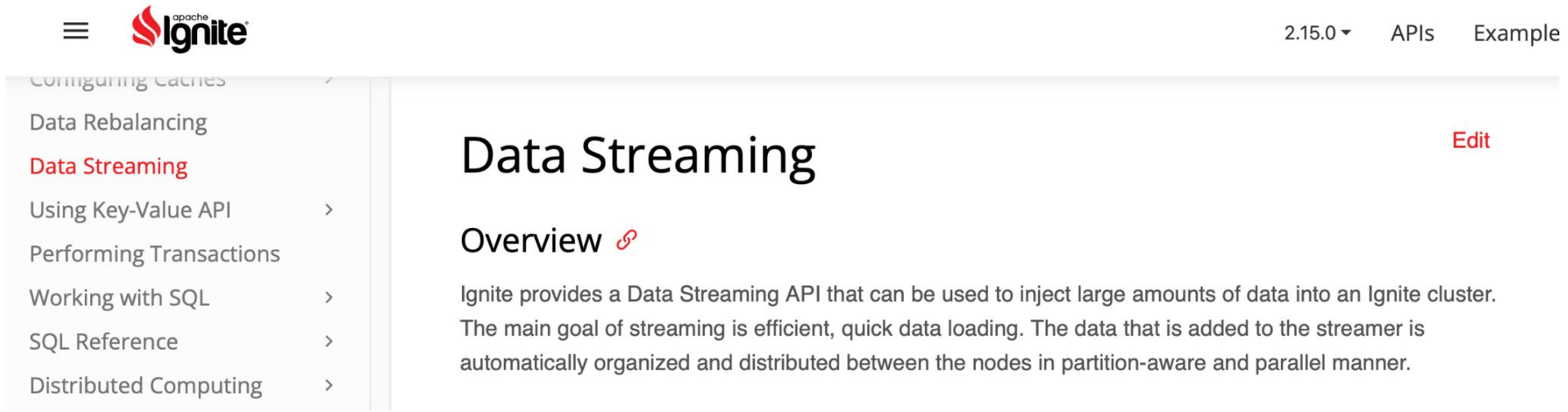
5 server nodes

4 726 writes per sec 😕

Data Streamer

1

1. Data Streamer




```
final IgniteDataStreamer<Long, StockOrder> dataStreamer = ignite.dataStreamer(cache.getName());
```

1. Data Streamer

```
data.forEach(entry -> {
    dataStreamer.addData(entry.getKey(), entry.getValue());
    if (...) { // or dataStreamer.autoFlushFrequency(...)
      dataStreamer.flush();
    }
 });
}
```

5 server nodes, flush every 100 000

1. Data Streamer

```
data.forEach(entry -> {
    dataStreamer.addData(entry.getKey(), entry.getValue());
    if (...) { // or dataStreamer.autoFlushFrequency(...)
      dataStreamer.flush();
    }
 });
}
```

5 server nodes, flush every 100 000

590 855 writes per sec 😀

1. Data Streamer

Так нельзя *

* в общем случае

1. Data Streamer

If 'allowOverwrite' property is 'false' (default), consider:

- You should not have the same keys repeating in the data being streamed;

```
dataStreamer.allowOverwrite(true);
...
```

5 server nodes, flush every 100 000

1. Data Streamer

If 'allowOverwrite' property is 'false' (default), consider:

- You should not have the same keys repeating in the data being streamed;

```
dataStreamer.allowOverwrite(true);
...
```

5 server nodes, flush every 100 000

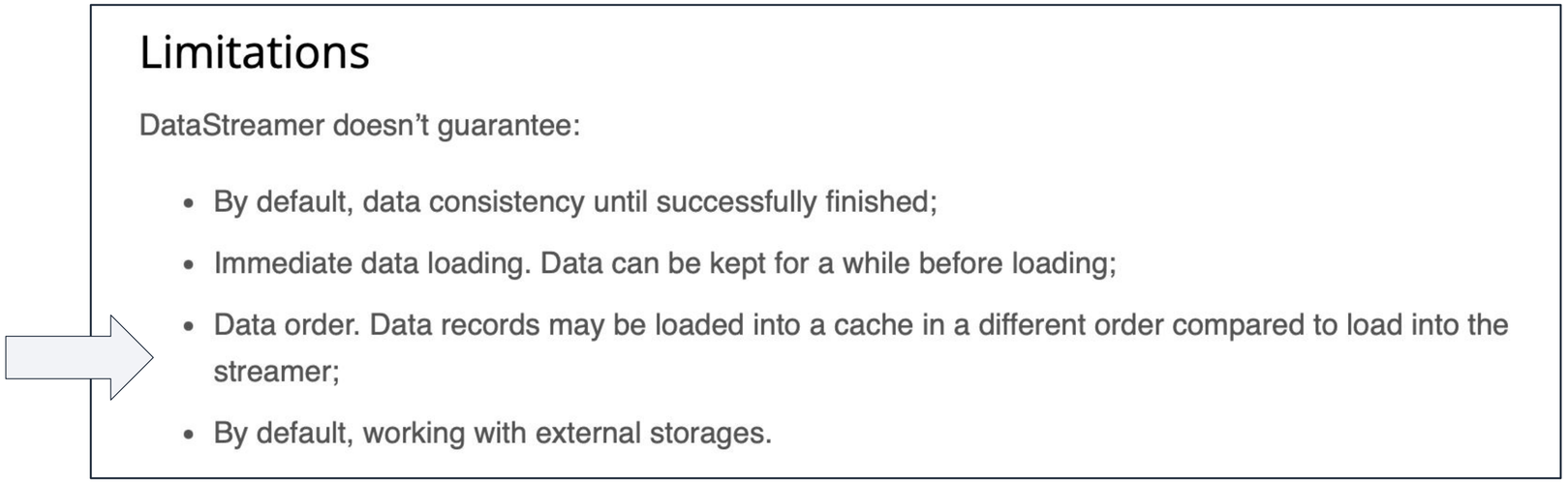
568 079 writes per sec 😀

1. Data Streamer

Так нельзя *

* так сложно

1. Data Streamer



💡 **Решения:**

1. **Гарантии интервала между повторяющимися ключами**
2. **“Умный” flush**

1. Data Streamer

Недостаточно *

* скорее всего

1. Data Streamer

```
final CacheConfiguration<Long, StockOrder> cacheConfiguration =
    new CacheConfiguration<Long, StockOrder>("benchmark")
          .setBackups(1)
          .setWriteSynchronizationMode(CacheWriteSynchronizationMode.FULL_SYNC); // default -
        PRIMARY_SYNC
          .setAtomicityMode(CacheAtomicityMode.ATOMIC)
          .setCacheMode(CacheMode.PARTITIONED)
ignite.createCache(cacheConfiguration);
```

1. Data Streamer

```
final CacheConfiguration<Long, StockOrder> cacheConfiguration =
    new CacheConfiguration<Long, StockOrder>("benchmark")
          .setBackups(1)
          .setWriteSynchronizationMode(CacheWriteSynchronizationMode.FULL_SYNC); // default -
        PRIMARY_SYNC
          .setAtomicityMode(CacheAtomicityMode.ATOMIC)
          .setCacheMode(CacheMode.PARTITIONED)
ignite.createCache(cacheConfiguration);
```

82 847 writes per sec 😨

1. Data Streamer

Streamer receivers.

Ignite DataStreamer is an orchestrator and doesn't write data itself. StreamerReceiver does. The default receiver is designed for fastest load and fewer network requests. With this receiver, streamer focuses on parallel transfer of backup and primary records.

```
public interface StreamReceiver<K, V> extends Serializable {
  /**
  * Updates cache with batch of entries.
  */
  public void receive(
    IgniteCache<K, V> cache,
    Collection<Map.Entry<K, V>> entries
  ) throws IgniteException;
}
```

1. Data Streamer

```
@Override public void allowOverwrite(boolean allow) {
    // ...
    rcvr = allow ? DataStreamerCacheUpdaters.<K, V>individual() : ISOLATED_UPDATER;
}
```

1. Data Streamer

```
@Override public void allowOverwrite(boolean allow) {
    // ...
    rcvr = allow ? DataStreamerCacheUpdaters.<K, V>individual() : ISOLATED_UPDATER;
}
```

```
/**
 * Isolated receiver which only loads entry initial value.
 */
protected static class IsolatedUpdater implements StreamReceiver<KeyCacheObject, CacheObject>
```

1. Data Streamer

```
@Override public void allowOverwrite(boolean allow) {
    // ...
    rcvr = allow ? DataStreamerCacheUpdaters.<K, V>individual() : ISOLATED_UPDATER;
}
```

```
private static class Individual<K, V> implements StreamReceiver<K, V>, InternalUpdater {
  @Override public void receive(IgniteCache<K, V> cache, Collection<Map.Entry<K, V>> entries) {
    // ...
    for (Map.Entry<K, V> entry : entries) {
      // ...
      cache.put(key, val);
    }
    // ...
  }
}
```

1. Data Streamer

```
final CacheConfiguration<Long, StockOrder> cacheConfiguration =
  new CacheConfiguration<Long, StockOrder>(“benchmark”)
        .setWriteSynchronizationMode(CacheWriteSynchronizationMode.PRIMARY_SYNC)
// ...
```

518 174 writes per sec

1. Data Streamer

```
final CacheConfiguration<Long, StockOrder> cacheConfiguration =
  new CacheConfiguration<Long, StockOrder>("benchmark")
      .setWriteSynchronizationMode(CacheWriteSynchronizationMode.PRIMARY_SYNC)
// ...
```

518 174 writes per sec

```
dataStreamer.receiver(DataStreamerCacheUpdaters.batched())

@Override public void receive(IgniteCache<K, V> cache, Collection<Map.Entry<K, V>> entries) {
  Map<K, V> putAll = null;
  for (Map.Entry<K, V> entry : entries) {
    // ...
    if (putAll == null) putAll = new HashMap<>();
    putAll.put(key, val);
  }
  updateAll(cache, rmvAll, putAll);
}
```

553 558 writes per sec

1. Data Streamer

Вывод:

Используйте при уникальных ключах, или

```
allowOverwrite(true) + DataStreamerCacheUpdaters.batched() + своевременный flush()
```

putAll(Map<K, V>)

2

2. putAll(Map<K, V>)

```
Map<Long, StockOrder> batch = new HashMap<>();
for (Entry<Long, StockOrder> entry : data) {
   batch.put(entry.getKey(), entry.getValue());
   if (...)
       cache.putAll(batch);
       batch = new HashMap<>();
   }
}

cache.putAll(batch);
```

2. putAll(Map<K, V>)

Test cases

🐣 case 1 “default”

putAll, batch 100 000, typed pojo, no sql, 0 backups, 0% key collision, 64 bytes data, 5 servers

🐌 case 2 “slow”

putAll, batch 10 000, typed pojo, sql, 2 backups, 1% key collision, 500 bytes data, 3 servers

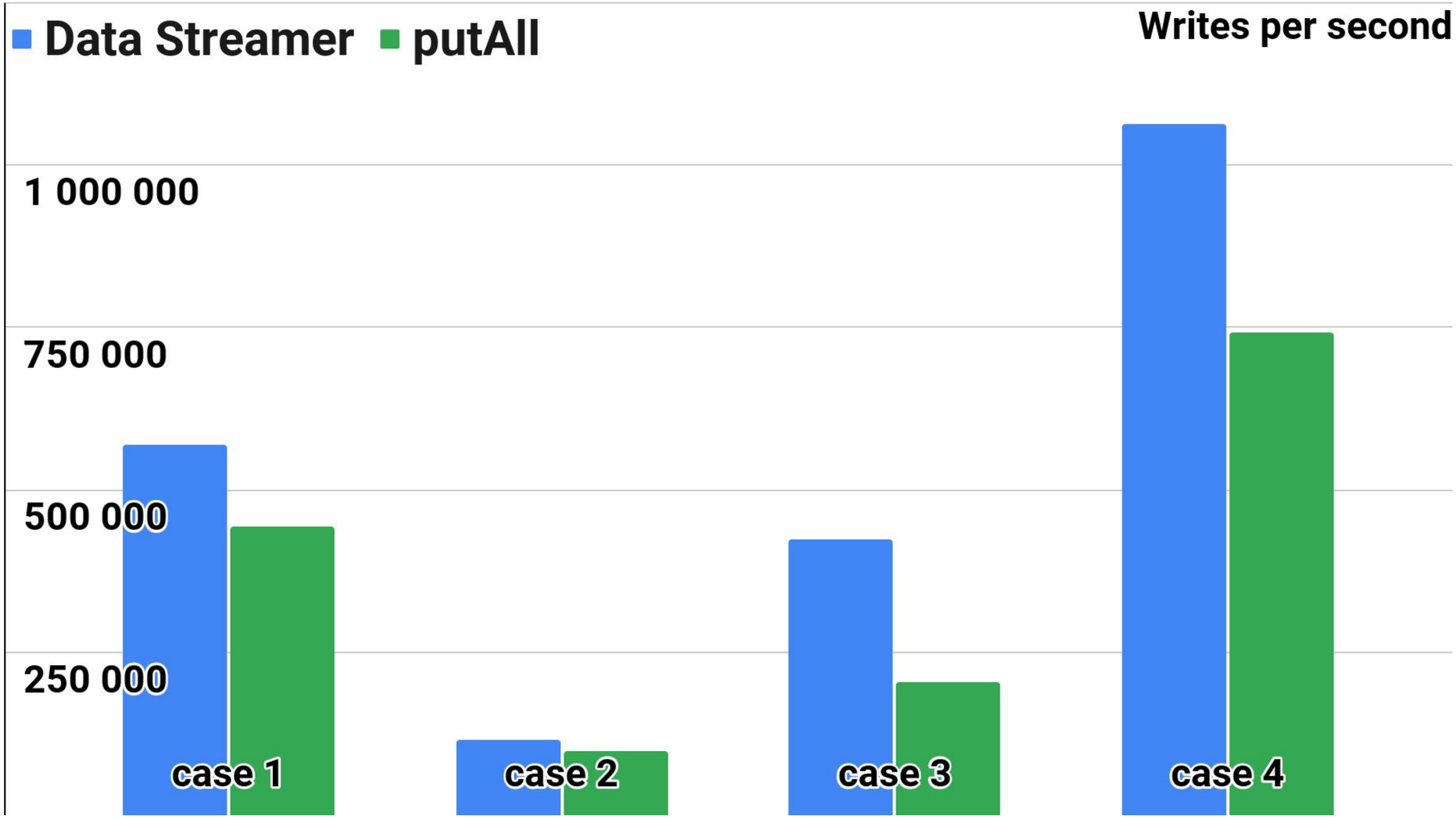
🐴 case 3 “medium”

Data Streamer, batch 100 000, binary object, sql, 1 backup, 0% key collision, 200 bytes data, 3 servers

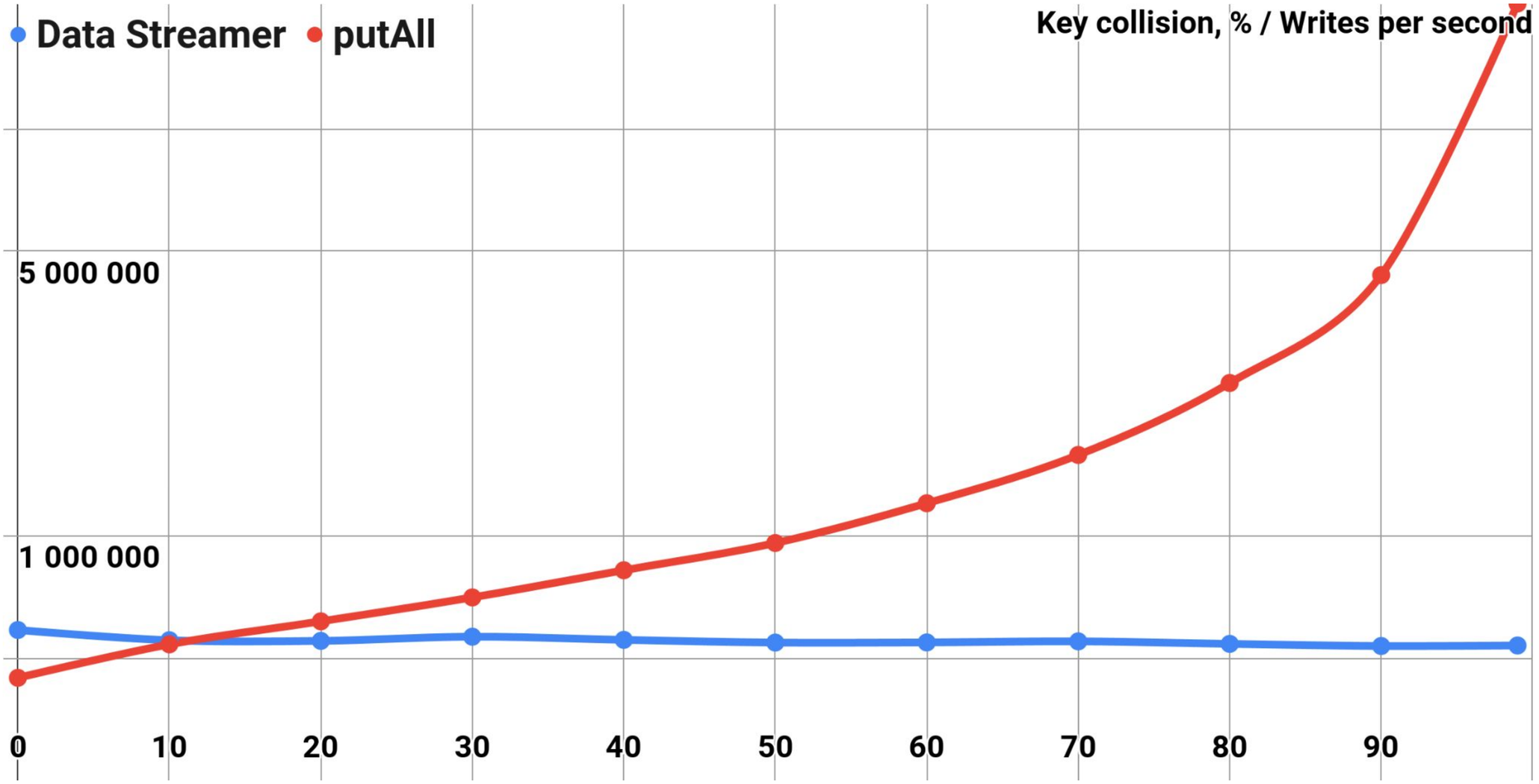
🏎️ case 4 “fast”

Data Streamer, batch 100 000, binary object, no sql, 0 backups, 0% key collision, 64 bytes data, 5 servers

2. putAll(Map<K, V>)



2. putAll(Map<K, V>)



2. putAll(Map<K, V>)

Так было нельзя *

* До версии 2.10

2. putAll(Map<K, V>)

```
LT.warn(log, "Unordered map " + m.getClass().getName() +
   " is used for " + op.title() + " operation on cache " + name() + ". " +
   "This can lead to a distributed deadlock. Switch to a sorted map like TreeMap instead.");
```

2. putAll(Map<K, V>)

```
private void updateAllAsyncInternal0(...) {
  // ...
  List<GridDhtCacheEntry> locked = lockEntries(req, req.topologyVersion());
  // ...
}
```

```
private List<GridDhtCacheEntry> lockEntries(GridNearAtomicAbstractUpdateRequest req,
AffinityTopologyVersion topVer) throws GridDhtInvalidPartitionException {
  // ...
  for (int i = 0; i < locked.size(); i++) {
    // ...
    entry.lockEntry();
    // ...
  }
  // ...
}
```

2. putAll(Map<K, V>)

💡 Решение:

1. Apache Ignite >= 2.10 / GGCE >= 8.8.23
2.
```
final TreeMap<Long, StockOrder> batch = new TreeMap<>();
```

SQL

3

3. SQL: Включение

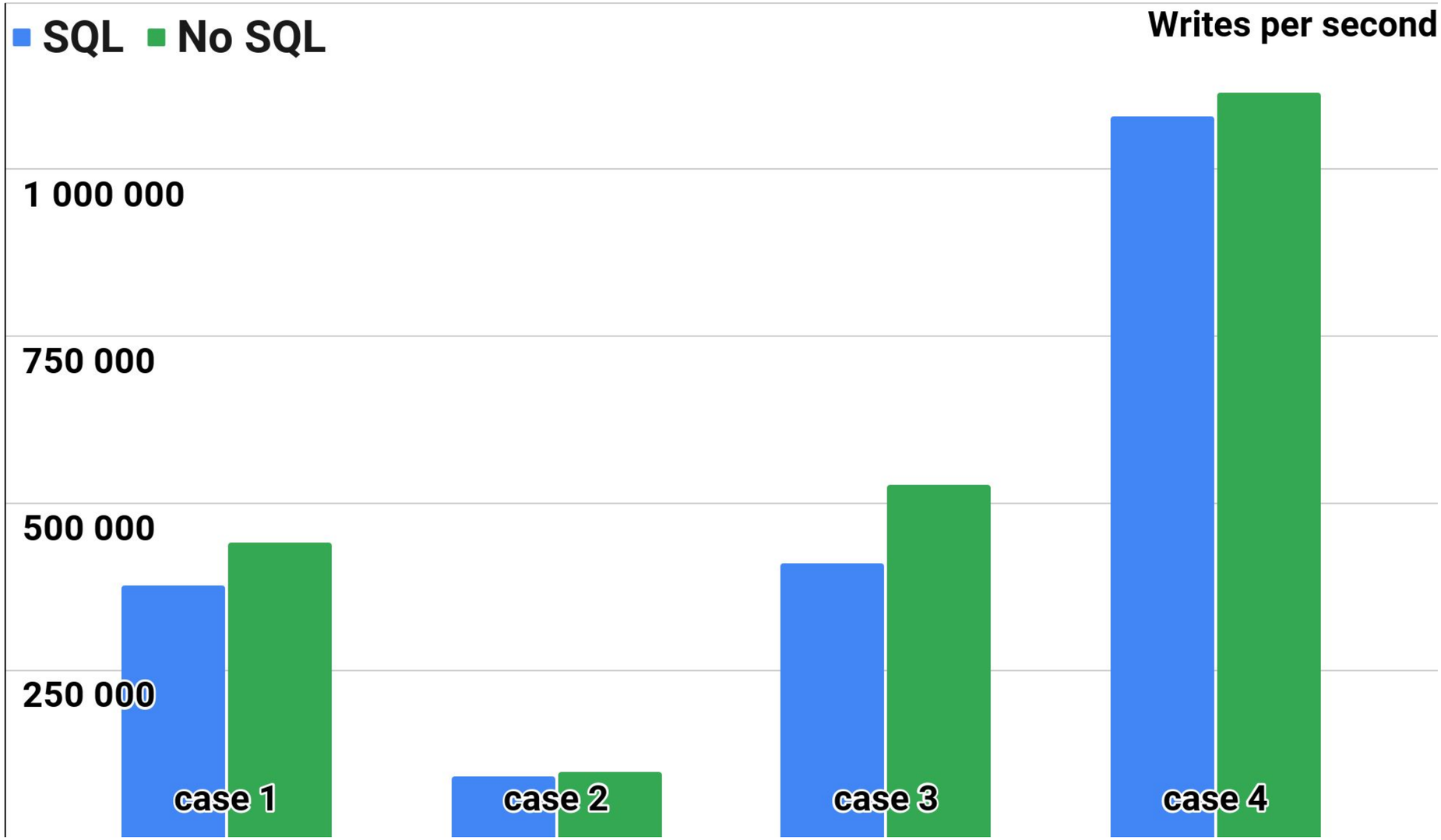
Код / XML

```
var fields = new LinkedHashMap<String, String>();
fields.put("secCode", String.class.getName());
fields.put("price", Double.class.getName());
fields.put("quantity", Integer.class.getName());
fields.put("buySell", Character.class.getName());
// ...
cacheCfg.setQueryEntities(Collections
  .singleton(new QueryEntity()
  .setKeyType(Long.class.getName())
  .setValueType(StockOrder.class.getName())
  .setTableName("StockOrder")
  .setFields(fields)));
```

Аннотации

```
public class StockOrder {
  @QuerySqlField
  private String secCode = "SBER";
  @QuerySqlField
   private Double price = 162.93;
  @QuerySqlField
   private Integer quantity = 2000;
  @QuerySqlField
   private Character buySell = 'B';
  // ...
}
cacheCfg.setIndexedTypes(
  Long.class,
  StockOrder.class
);
```

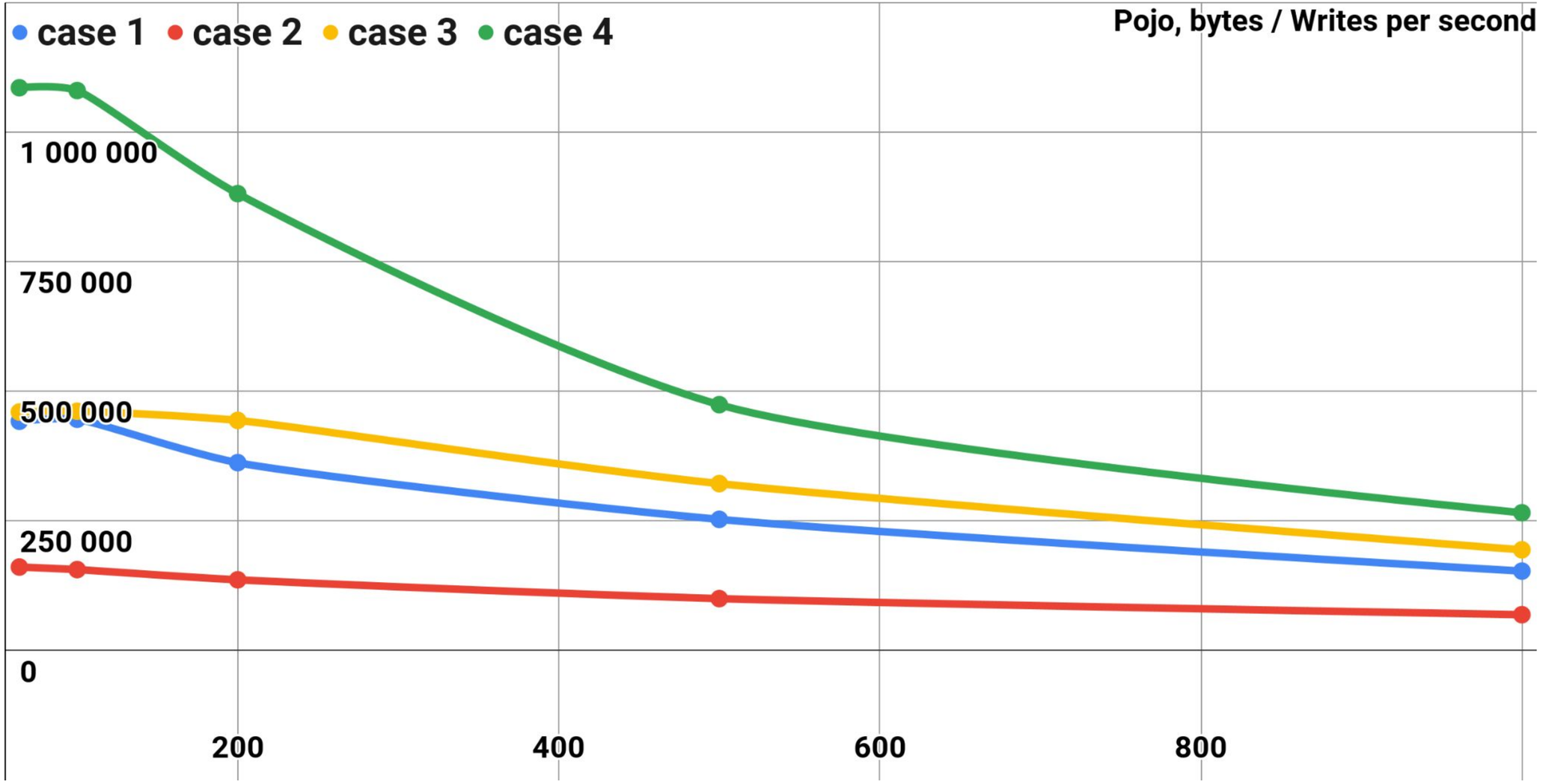
3. SQL


SQL
No SQL
Writes per second
1 000 000
750 000
500 000
250 000
case 1
case 2
case 3
case 4

Размер pojo

4

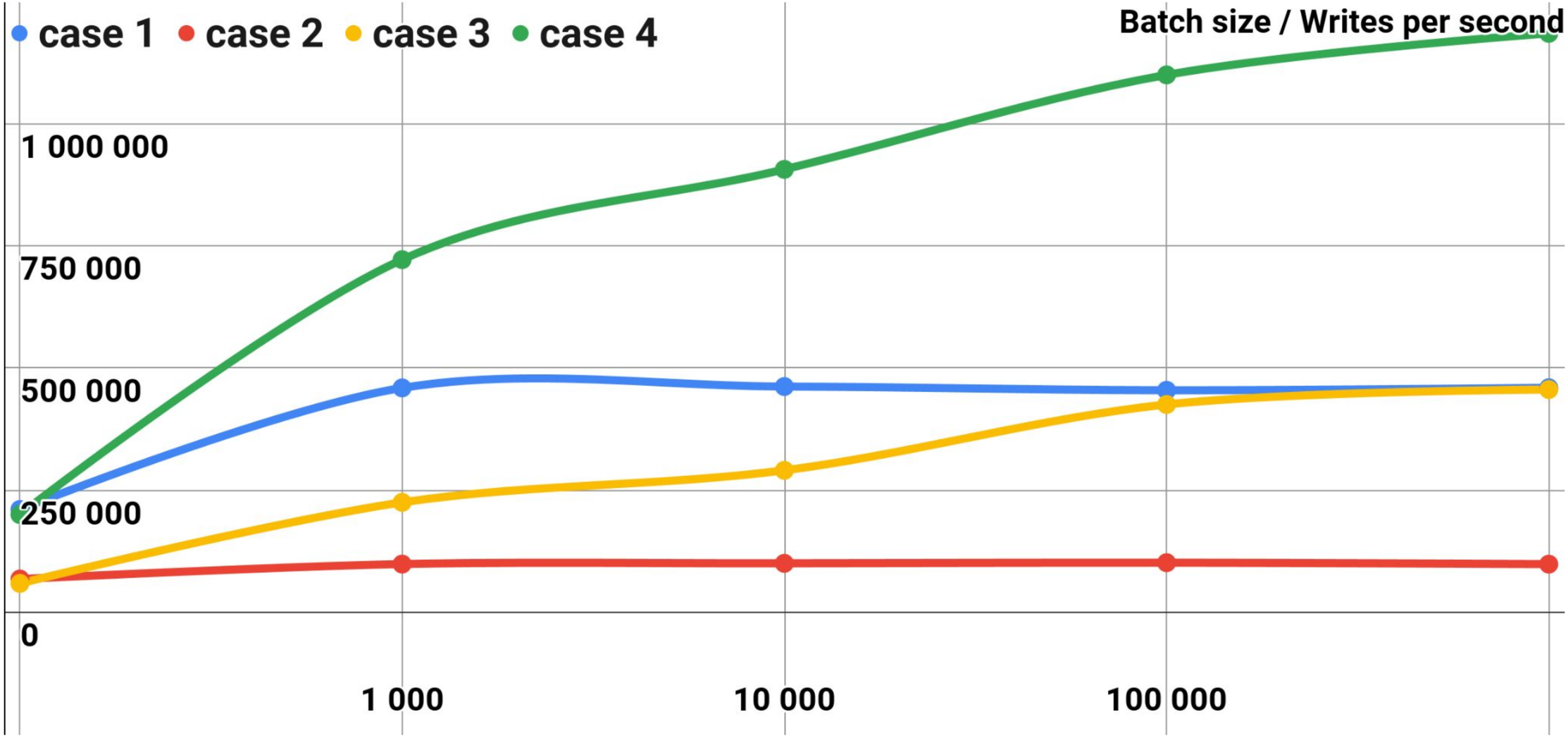
4. Размер pojo



Размер batch

5

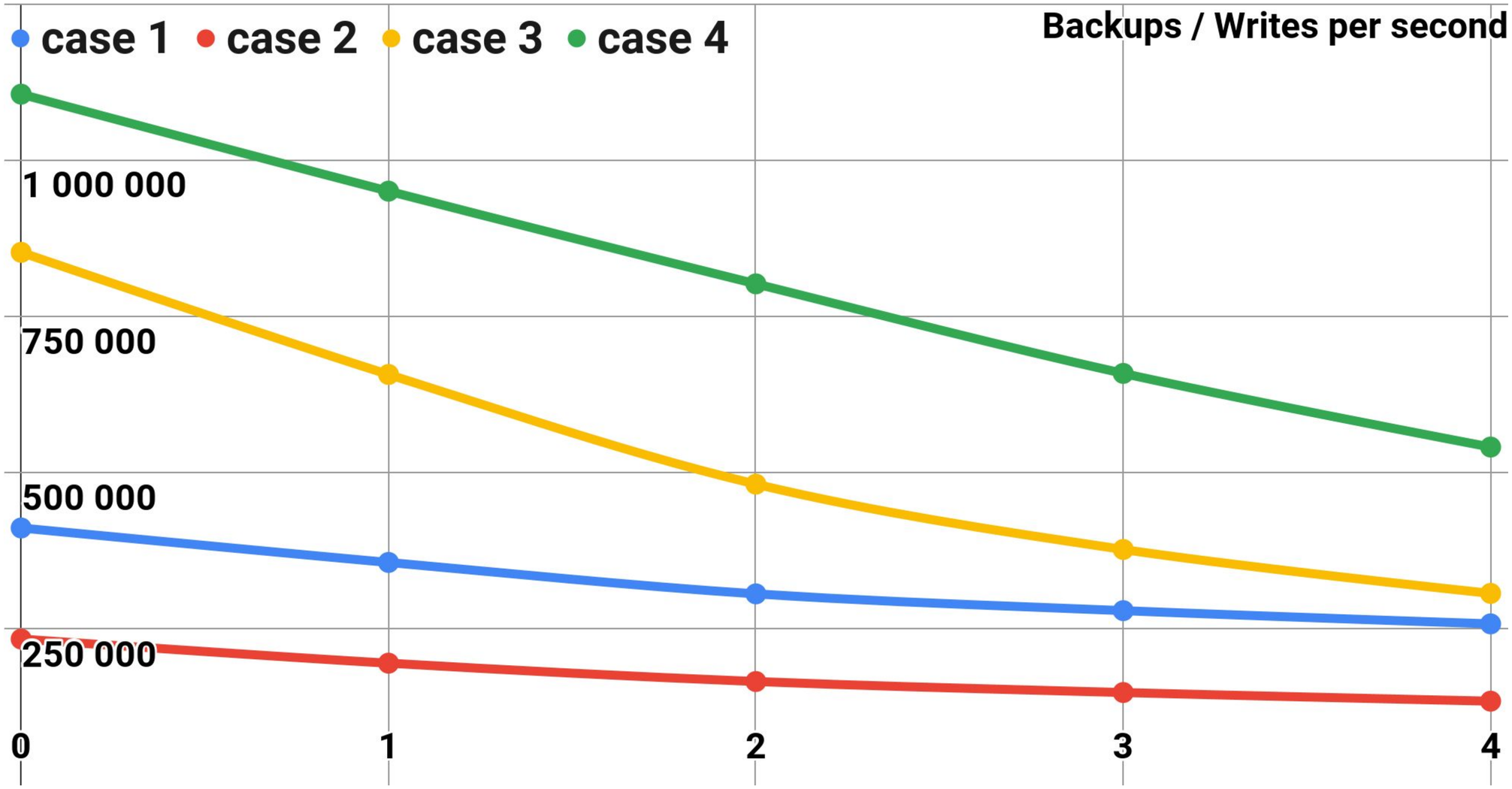
5. Размер batch



Backups

6

6. Backups

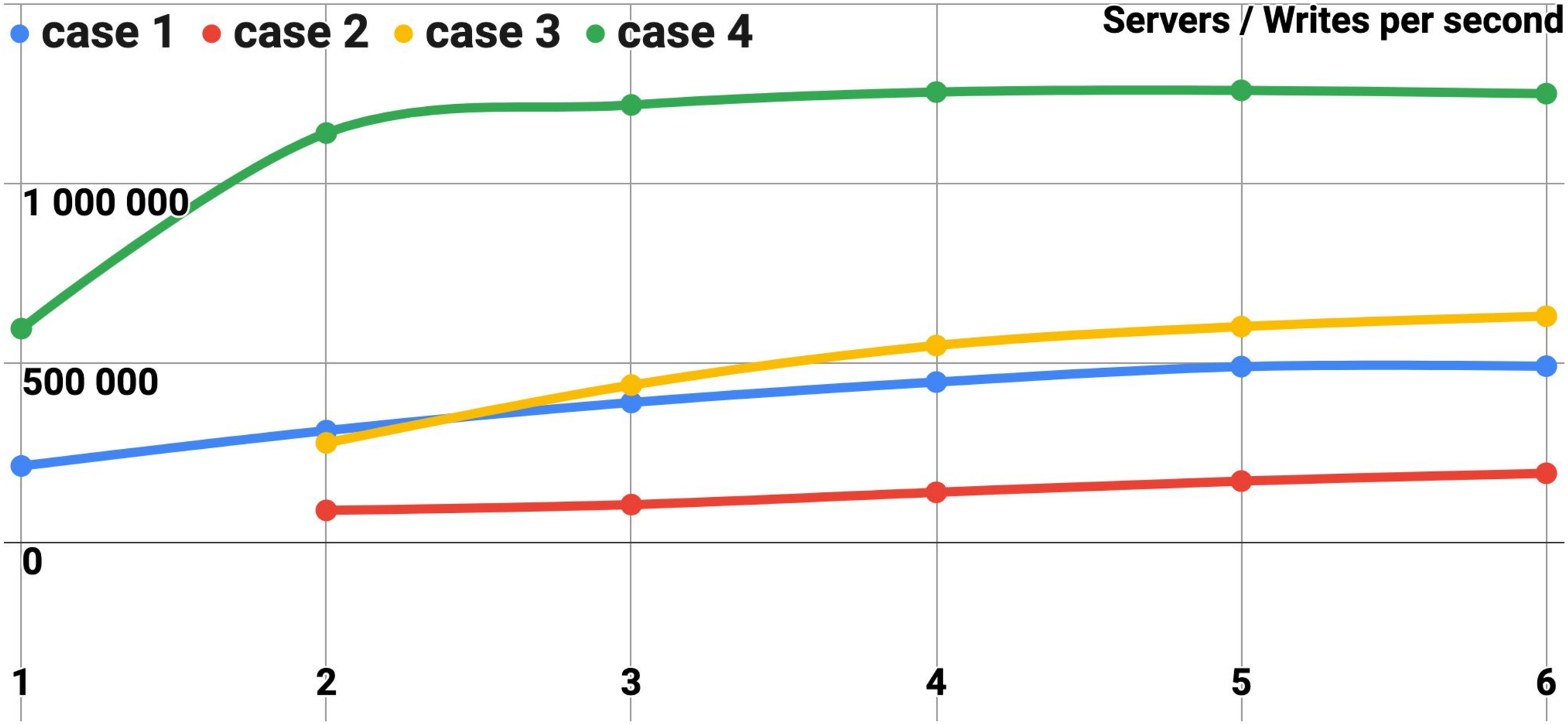

case 1
case 2
case 3
case 4
Backups / Writes per second
1 000 000
750 000
500 000
250 000
0
1
2
3
4

Размер кластера

7

7. Размер кластера

7. Размер кластера



Тип узла

8. Тонкий клиент

- Не является частью кластера
- Не содержит Data Streamer
- Делегирует putAll узлу кластера

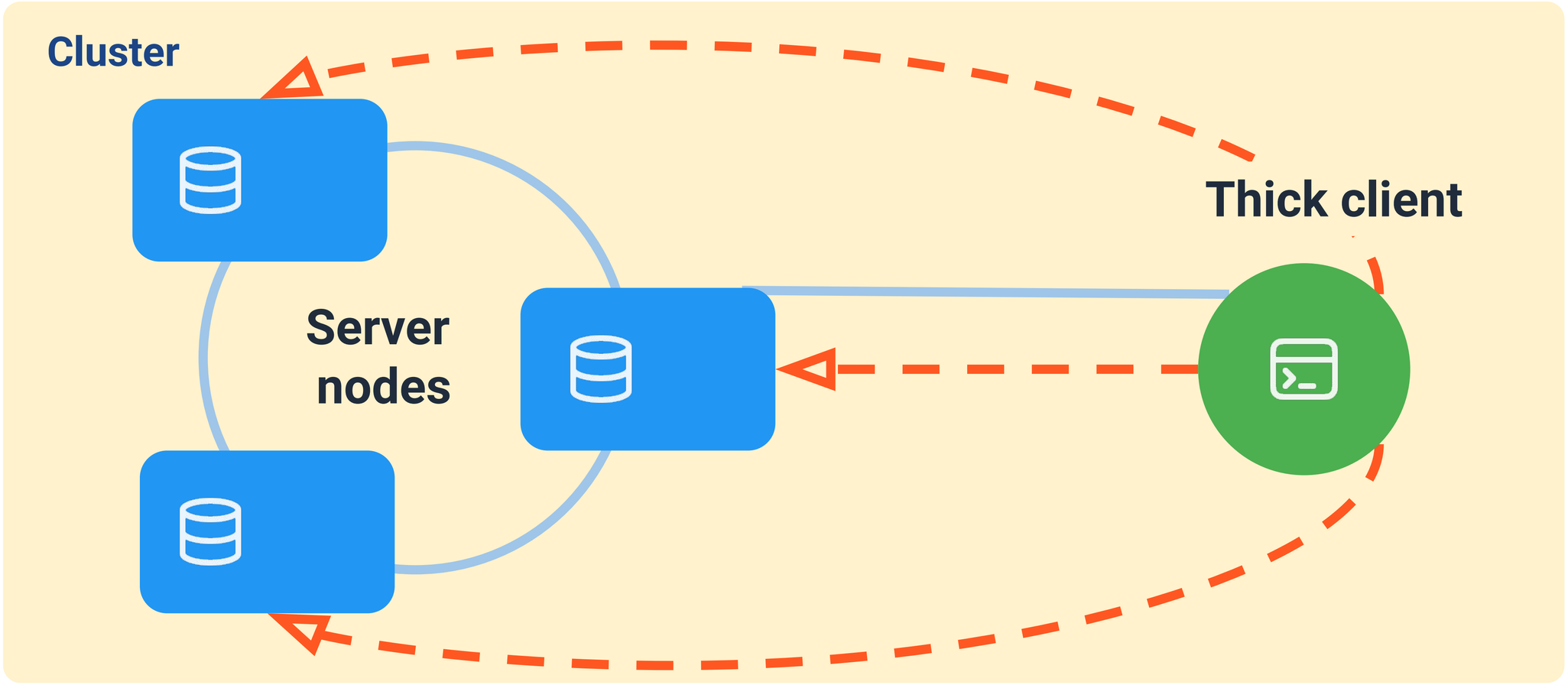
```
ClientConfiguration cfg = new ClientConfiguration().setAddresses(...);
try (IgniteClient client = Ignition.startClient(cfg)) {
  ClientCache<Long, StockOrder> cache = client.cache(...);
   ...
}
```

8. Толстый клиент

```
Ignite ignite = Ignition.start(new IgniteConfiguration().setClientMode(true) ...)
```

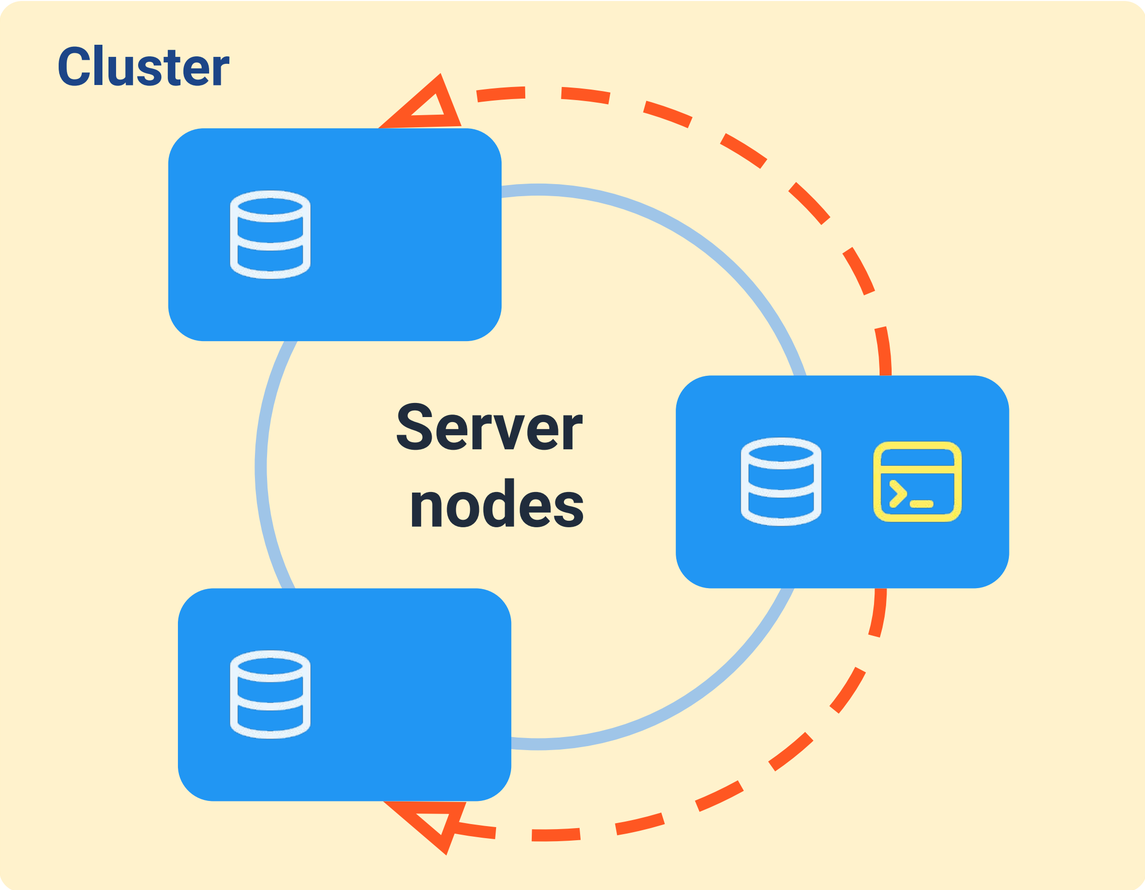
8. Толстый клиент

```
Ignite ignite = Ignition.start(new IgniteConfiguration().setClientMode(true) ...)
```



8. Серверный узел

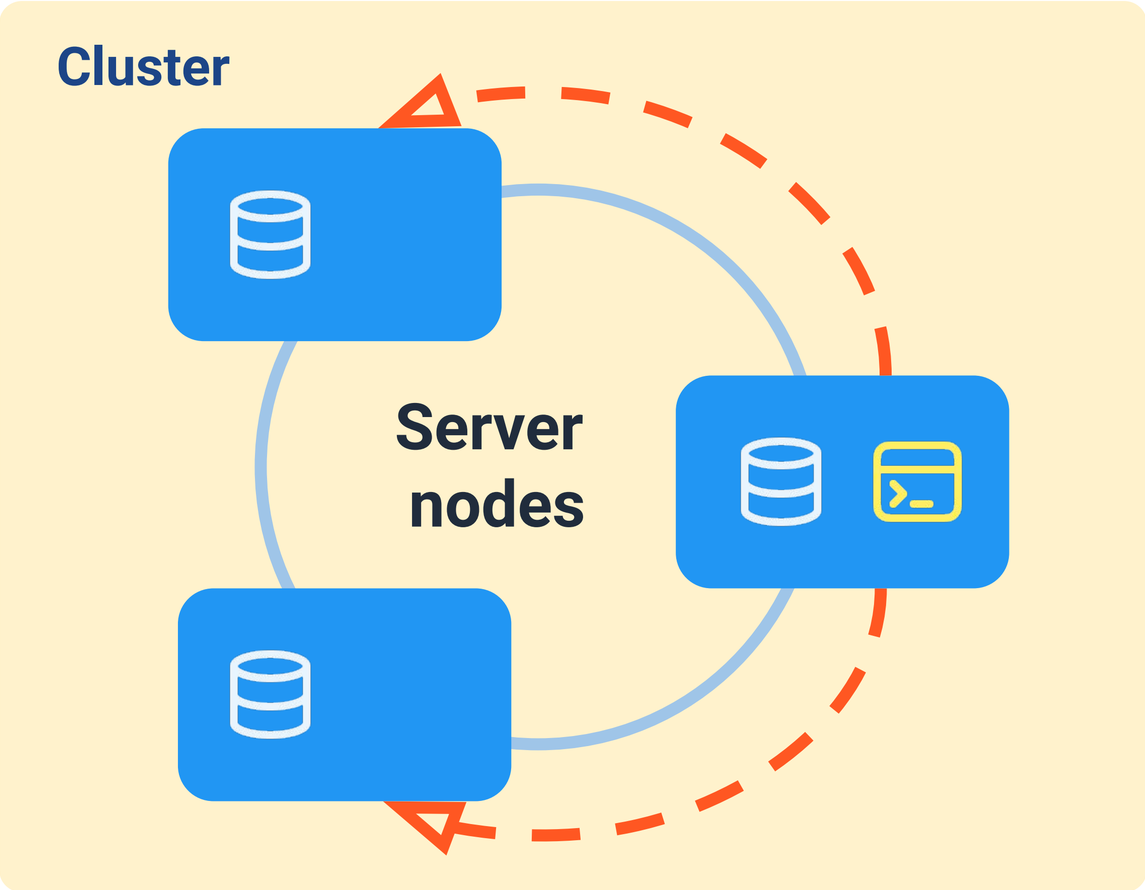
```
Ignite ignite = Ignition.start(new IgniteConfiguration().setClientMode(false) ...)
```



1️⃣ Запустить Ignite в серверном режиме

8. Серверный узел

```
Ignite ignite = Ignition.start(new IgniteConfiguration().setClientMode(false) ...)
```

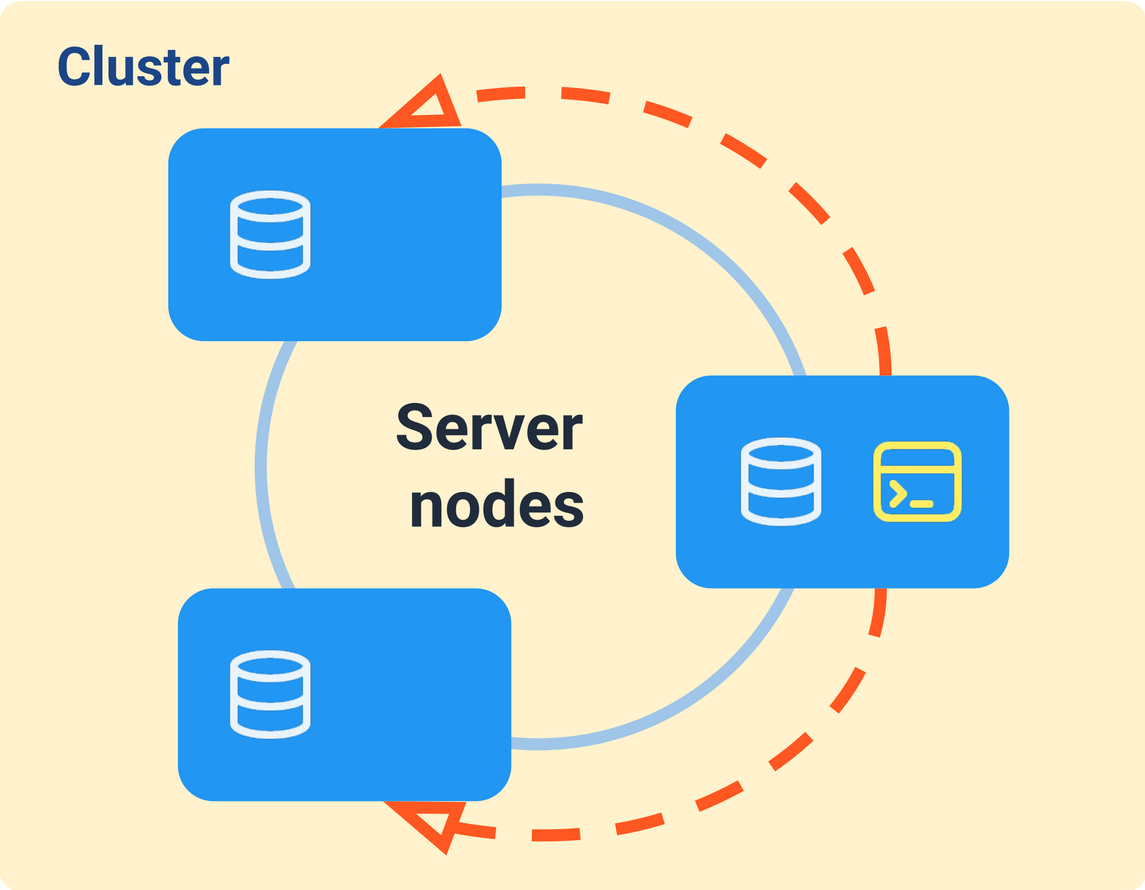


1️⃣ Запустить Ignite в серверном режиме

2️⃣ Выполнить задачу через IgniteCompute

8. Серверный узел

```
Ignite ignite = Ignition.start(new IgniteConfiguration().setClientMode(false) ...)
```

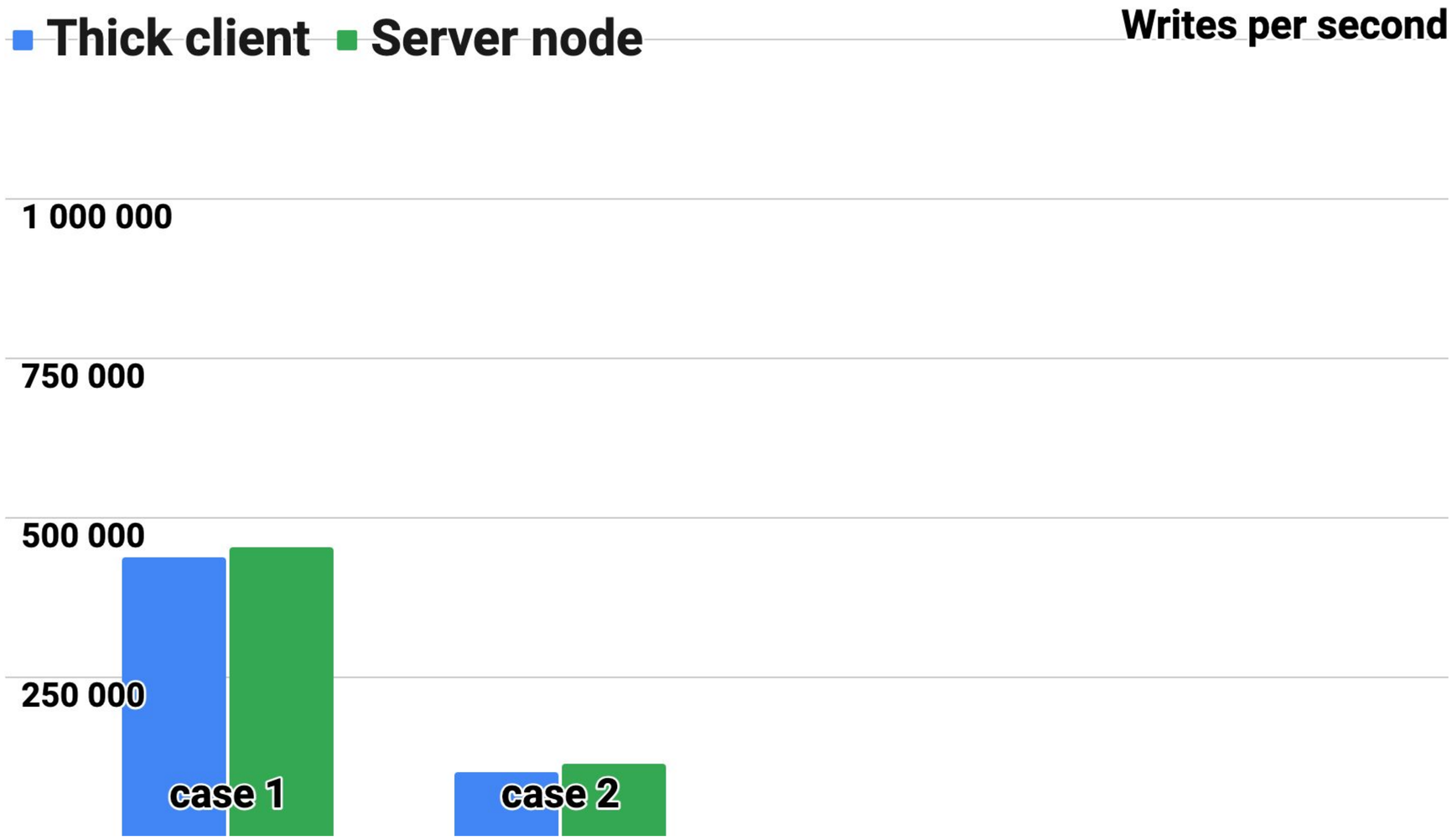


1️⃣ Запустить Ignite в серверном режиме

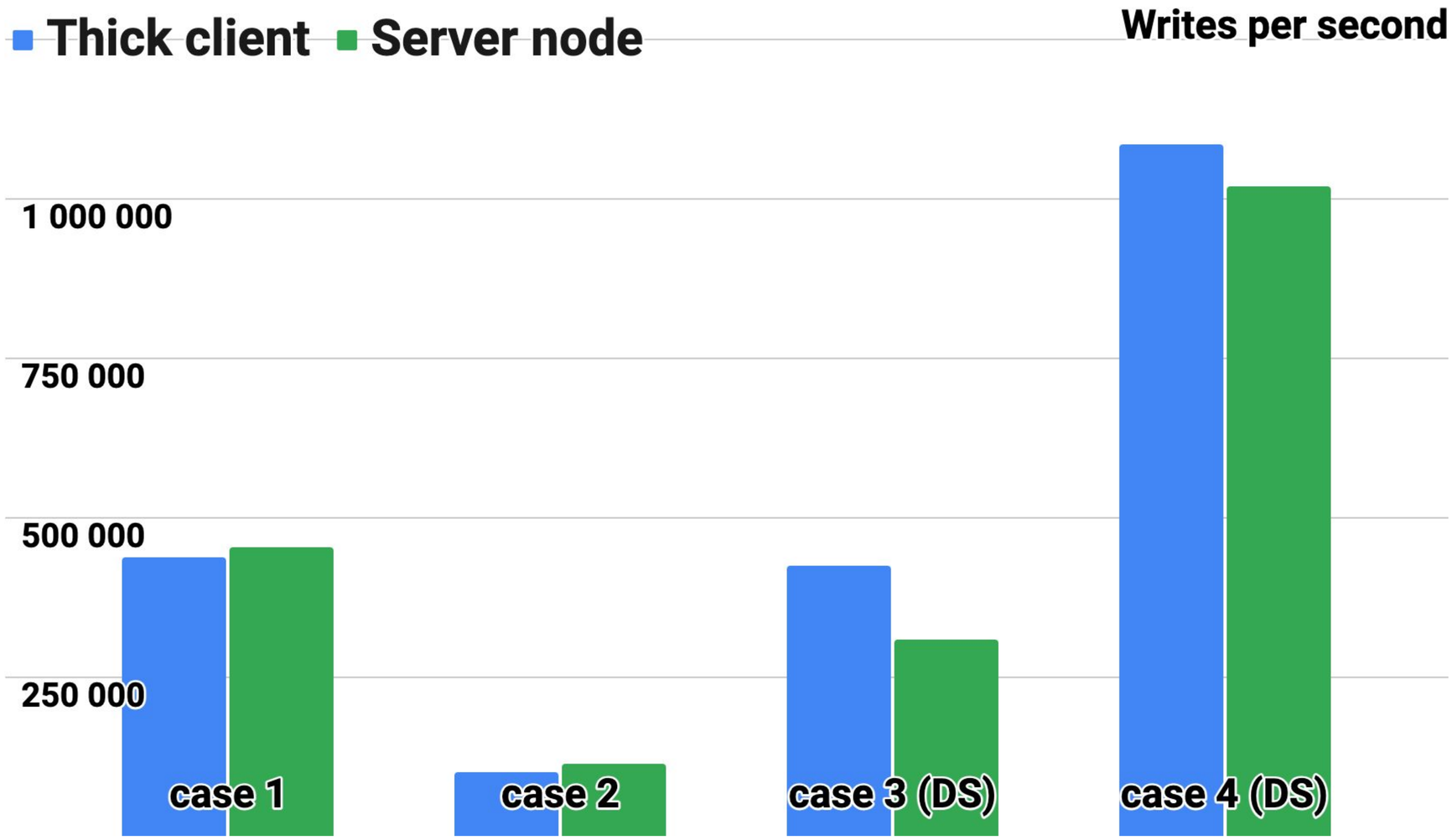
2️⃣ Выполнить задачу через IgniteCompute

3️⃣ Задеплоить сервис через IgniteServices, в том числе Singleton Service

8. Серверный узел



8. Серверный узел



Binary object

9. Binary Object

9. Binary Object


```
public Object toBinary(@Nullable Object obj, boolean failIfUnregistered) throws IgniteException {
  if (obj == null)
    return null;
  if (isBinaryObject(obj))
    return obj;
  return marshalToBinary(obj, failIfUnregistered);
}
```

9. Binary Object

```
IgniteBinary igniteBinary = Ignite.binary();
```

9. Binary Object

```
IgniteBinary igniteBinary = Ignite.binary();
```

```
BinaryObjectBuilder builder = igniteBinary.builder(StockOrder.class.getName());
builder.setField("secCode", secCode);
...
builder.setField("price", price);
BinaryObject binaryObject = builder.build();
```

9. Binary Object

```
IgniteBinary igniteBinary = Ignite.binary();
```

```
BinaryObjectBuilder builder = igniteBinary.builder(StockOrder.class.getName());
builder.setField("secCode", secCode, String.class);
...
builder.setField("price", price, Double.class);
BinaryObject binaryObject = builder.build();
```

9. Binary Object

```
IgniteBinary igniteBinary = Ignite.binary();
```

```
BinaryObjectBuilder builder = igniteBinary.builder("any.string");
builder.setField("stringField", aString, String.class);
...
builder.setField("longField", aLong, Long.class);
BinaryObject binaryObject = builder.build();
```

9. Binary Object

```
IgniteBinary igniteBinary = Ignite.binary();
```

```
BinaryObjectBuilder builder = igniteBinary.builder("any.string");
builder.setField("stringField", aString, String.class);
...
builder.setField("longField", aLong, Long.class);
BinaryObject binaryObject = builder.build();
```

```
BinaryObject binaryObject = igniteBinary.toBinary(new StockOrder());
```

9. Binary Object

```
IgniteBinary igniteBinary = Ignite.binary();
```

```
BinaryObjectBuilder builder = igniteBinary.builder("any.string");
builder.setField("stringField", aString, String.class);
...
builder.setField("longField", aLong, Long.class);
BinaryObject binaryObject = builder.build();
```

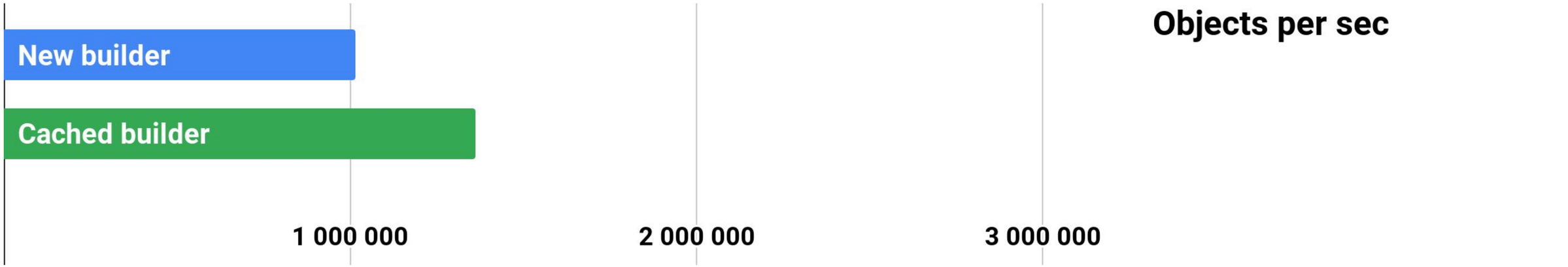
```
BinaryObject binaryObject = igniteBinary.toBinary(new StockOrder());
```

```
IgniteCache<Long, BinaryObject> binaryCache = ignite.cache("cacheName").withKeepBinary();
```

9. Binary Object

💡 Builder можно кешировать!

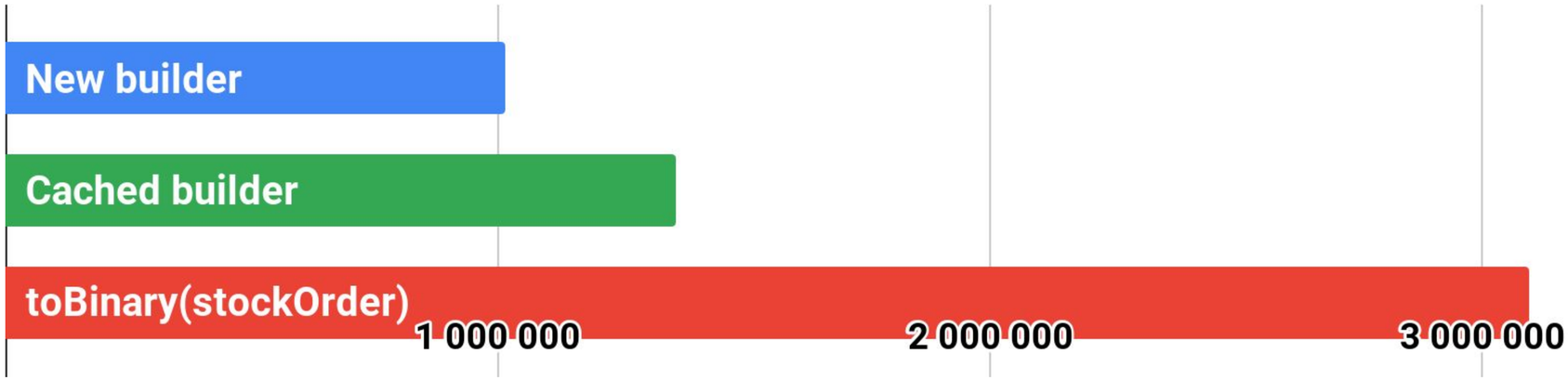
```
BinaryObjectBuilder builder = igniteBinary.builder(StockOrder.class.getName());
builder.setField("price", price, Double.class);
...
BinaryObject bo1 = builder.build();
builder.setField("price", price, Double.class);
...
BinaryObject bo2 = builder.build();
...
```



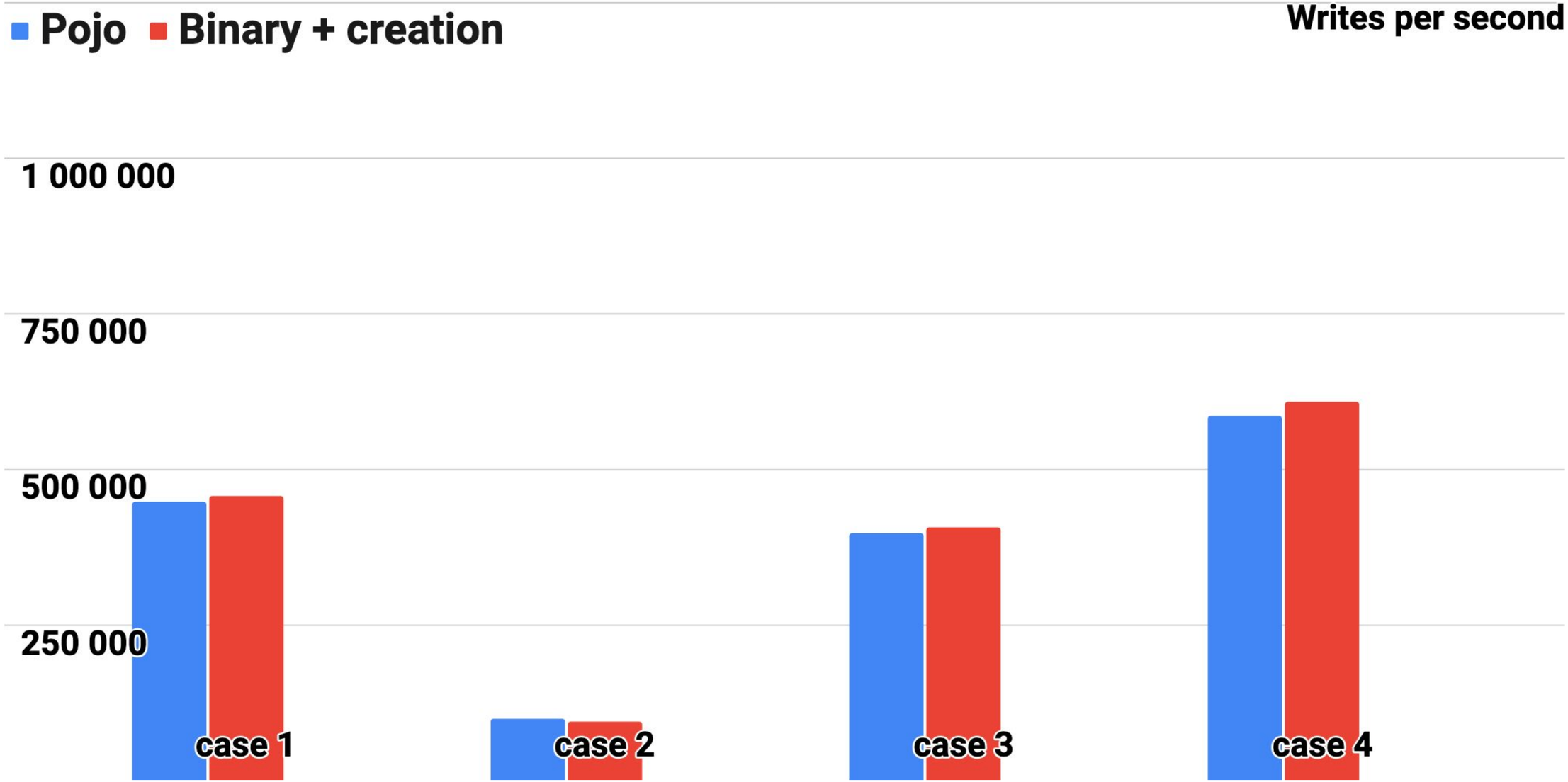
9. Binary Object

💡 Builder можно кешировать!

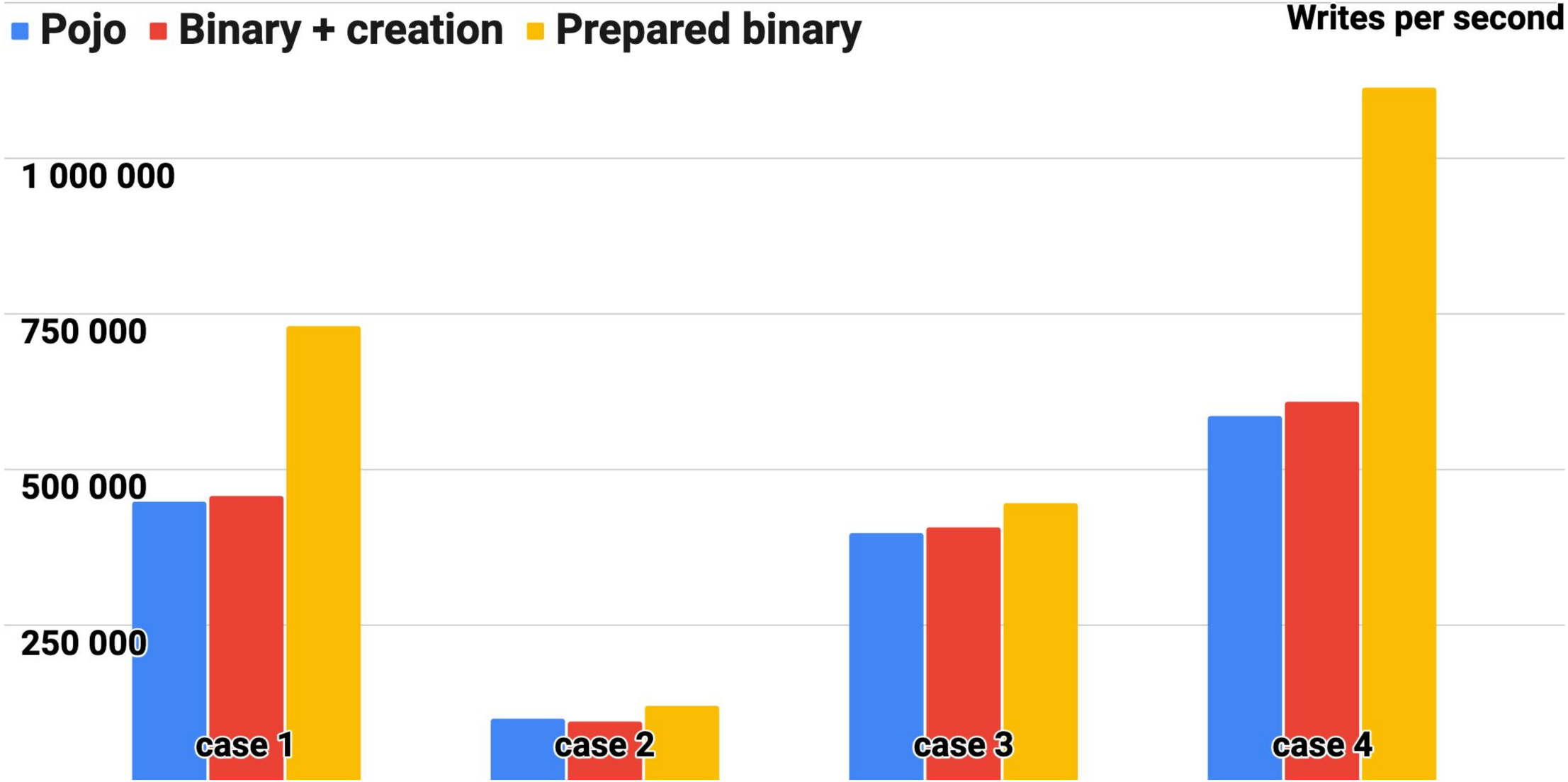
```
BinaryObjectBuilder builder = igniteBinary.builder(StockOrder.class.getName());
builder.setField("price", price, Double.class);
...
BinaryObject bo1 = builder.build();
builder.setField("price", price, Double.class);
...
BinaryObject bo2 = builder.build();
...
```



9. Binary Object



9. Binary Object



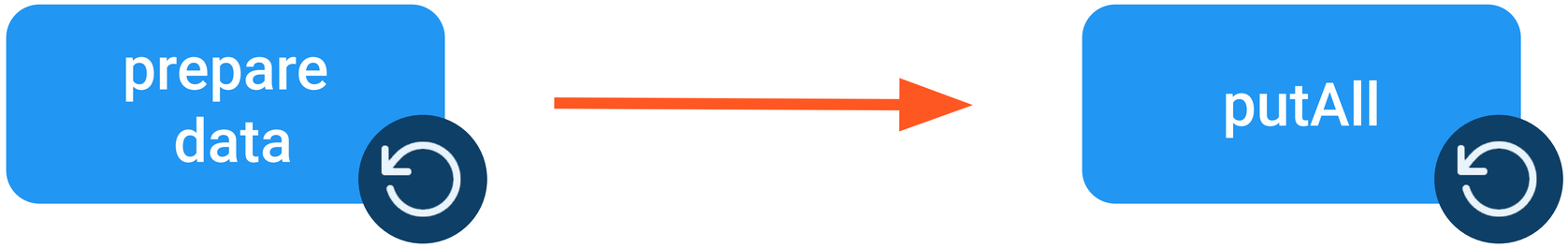
putAllAsync

10

10. putAllAsync

```
while (!Thread.currentThread().isInterrupted()) {
  final Iterable<byte[]> serializedPojos = pollSerializedData();
  final Map<Long, StockOrder> batch = deserializeAndCollect(serializedPojos); //CPU expensive
  cache.putAll(batch);
}
```

10. putAllAsync



thread 1

thread 2

10. putAllAsync

```
private IgniteFuture<Void> currentWriteFuture = ...;
public void pollAndWriteLoop() {
  while (!Thread.currentThread().isInterrupted()) {
    final Iterable<byte[]> serializedPojos = pollSerializedData();
    final Map<Long, StockOrder> batch = deserializeAndCollect(serializedPojos);
    currentWriteFuture.get();
    currentWriteFuture = cache.putAllAsync(batch);
  }
}
```

10. putAllAsync

```
private IgniteFuture<Void> currentWriteFuture = ...;
public void pollAndWriteLoop() {
  while (!Thread.currentThread().isInterrupted()) {
    final Iterable<byte[]> serializedPojos = pollSerializedData();
    final Map<Long, StockOrder> batch = deserializeAndCollect(serializedPojos);
    currentWriteFuture.get();
    currentWriteFuture = cache.putAllAsync(batch);
  }
}
```

10. putAllAsync

```
private IgniteFuture<Void> currentWriteFuture = ...;
public void pollAndWriteLoop() {
  while (!Thread.currentThread().isInterrupted()) {
    final Iterable<byte[]> serializedPojos = pollSerializedData();
    final Map<Long, StockOrder> batch = deserializeAndCollect(serializedPojos);
    currentWriteFuture.get();
    currentWriteFuture = cache.putAllAsync(batch);
  }
}
```

10. putAllAsync

```
private IgniteFuture<Void> currentWriteFuture = ...;
public void pollAndWriteLoop() {
  while (!Thread.currentThread().isInterrupted()) {
    final Iterable<byte[]> serializedPojos = pollSerializedData();
    final Map<Long, BinaryObject> batch = deserializeAndCollect(serializedPojos);
    currentWriteFuture.get();
    currentWriteFuture = cache.putAllAsync(batch);
  }
}
```

Entry Processor

11

11. Entry Processor

Update:

```
quantity = 1 000
```

Cache

StockOrder	
secCode	SBER
price	162.93
quantity	2000
buySell	B
secBoard	TQBR
account	S01-12345678
user	MU1234567890
currency	RUB
commission	0.13
status	0
marketMaker	false

11. Entry Processor

```
public class QuantityUpdateProcessor implements CacheEntryProcessor<Long, StockOrder, Void> {
  private final int newQuantity;

  public QuantityUpdateEntryProcessor(int newQuantity) { this.newQuantity = newQuantity; }

  @Override
  public Void process(MutableEntry<Long, StockOrder> mutableEntry, Object... objects) {
    StockOrder currentValue = mutableEntry.getValue();
    currentValue.setQuantity(newQuantity);
    mutableEntry.setValue(currentValue);
    return null;
  }
}
```

```
cache.invoke(1L, new QuantityUpdateEntryProcessor(1));
cache.invokeAll(Map.of(1L, new QuantityUpdateEntryProcessor(1)));
```

11. Entry Processor

```
public class FieldUpdateProcessor<K> implements CacheEntryProcessor<K, BinaryObject, Void> {
  private final String fieldName;
  private final Object fieldValue;

  public FieldUpdateEntryProcessor(String fieldName, Object fieldValue) {
    this.fieldName = fieldName;
    this.fieldValue = fieldValue;
  }

  @Override
  public Void process(MutableEntry<K, BinaryObject> mutableEntry, Object... objects) {
    BinaryObject currentValue = mutableEntry.getValue();
    BinaryObject updatedValue = currentValue.toBuilder()
        .setField(fieldName, fieldValue).build();
    mutableEntry.setValue(updatedValue);
    return null;
  }
}
```

Выводы

1. Проверьте, могут ли ключи повторяться
2. `DataStreamer` при уникальных. Иначе `putAll(Async)` / настроенный `Data Streamer`
3. Распараллельте подготовку данных и запись в кластер
4. Пишите `BinaryObject`, большими батчами
5. Используйте `EntryProcessor` для изменения части полей

Benchmarks

Григорий Доможиров

https://www.linkedin.com/in/gdomo

@hunt_id

grigorydomozhirov@gmail.com