Руководство администратора платформы «РЕД КВАНТ»

СОДЕРЖАНИЕ

ВВЕДЕНИЕ **6**

1 СИСТЕМНЫЕ ТРЕБОВАНИЯ **11**

2 УСТАНОВКА И НАСТРОЙКА **12**

2.1 Установка из ZIP-архива 12
2.2 Установка из Docker 12
2.3 Базовая настройка и конфигурация 13
2.3.1 XML-конфигурация Spring 13
2.3.2 Программная конфигурация 14
2.3.3 Настройка 14
2.3.3.1 Запуск РЕД КВАНТ с Java 11 или более поздней версии 14
2.3.3.2 Использование двоичного дистрибутива 15
2.3.3.3 Использование Maven 15
2.3.3.4 Использование Docker 16
2.3.3.5 Настройка рабочего каталога 16
2.3.3.6 Включение модулей 16
2.3.3.7 Настройка параметров JVM 20
2.3.3.8 Настройка свойств системы РЕД КВАНТ 20
2.3.4 Рекомендации по настройке 21
2.3.4.1 Настройка рабочего каталога 21
2.3.4.2 Рекомендуемая конфигурация ведения журнала 21
2.4 Настройка логгирования 21
2.4.1 Ведение журнала по умолчанию 22
2.4.2 Использование Log4j 22
2.4.3 Использование Log4j2 23
2.4.4 Использование JCL 24
2.4.5 Использование SLF4J 24
2.4.6 Скрытие конфиденциальной информации 24
2.4.7 Пример конфигурации ведения журнала 25

3 ЗАПУСК И УСТАНОВКА УЗЛОВ **26**

3.1 Запуск серверных узлов 26
3.2 Запуск клиентских узлов 26
3.3 Завершение работы узлов 27
3.4 События жизненного цикла узла 27

4 НАСТРОЙКА КЛАСТЕРА **29**

4.1 TCP/IP обнаружение 30
4.1.1 Multicast поиск IP-адресов 30
4.1.2 Статический поиск IP 30
4.1.3 Multicast и статический поиск IP-адресов 31
4.1.4 Изолированные кластеры на одном наборе машин 32
4.1.5 Поиск IP-адресов на основе JDBC 34
4.1.6 Поиск IP-адресов в общей файловой системе 35
4.1.7 Поисковик IP-адресов ZooKeeper 35

4.2 Обнаружение через ZooKeeper 35
4.2.1 Сбои и обработка Split-brain 37
4.2.1.1 Кластер разбит на несколько непересекающихся компонентов 37
4.2.1.2 Отсутствуют несколько связей между узлами 39
4.2.1.3 Сегментация кластера ZooKeeper 40
4.2.2 Пользовательские события обнаружения 41
4.2.3 Рекомендации по настройке РЕД КВАНТ и ZooKeeper 41
4.3 Сетевая конфигурация 42
4.3.1 Обнаружение 42
4.3.2 Коммуникация 44
4.3.3 Тайм-ауты подключения 46
4.4 Подключение клиентских узлов 47
4.4.1 Повторное подключение клиентского узла 47
4.4.2 События отключения/повторного подключения клиента 48
4.4.3 Управление медленными клиентскими узлами 48
4.5 Топология Baseline 49
4.5.1 Базовая топология в кластерах с чистой памятью 50
4.5.2 Базовая топология в кластерах с персистентностью 50
4.5.3 Автоматическая настройка топологии Baseline 50
4.5.4 Мониторинг топологии Baseline 51

5 КОНФИГУРАЦИЯ ПАМЯТИ 52

5.1 Конфигурация регионов памяти 52
5.1.1 Настройка региона данных по умолчанию 52
5.1.2 Добавление пользовательских регионов данных 53
5.1.3 Стратегия прогрева кэша 54
5.2 Политики удаления данных 56
5.2.1 Освобождение памяти off-heap 57
5.3 Политики вытеснения данных 60

6 КОНФИГУРАЦИЯ СЛОЯ ХРАНЕНИЯ 63

6.1 Настройка хранения данных в РЕД КВАНТ 63
6.1.1 Включение постоянного хранилища 64
6.1.2 Настройка каталога постоянного хранилища 64
6.1.3 Журнал предварительной записи 65
6.1.3.1 WAL-режимы 66
6.1.3.2 Архив WAL 67
6.1.3.3 Изменение размера сегмента WAL 67
6.1.3.4 Отключение WAL 68
6.1.3.5 Сжатие архива WAL 68
6.1.3.6 Сжатие записей WAL 69
6.1.3.7 Отключение архива WAL 69
6.1.4 Checkpointing 69
6.1.5 Свойства конфигурации 70
6.2 Внешнее хранилище 71
6.2.1 Сквозное чтение и запись 72
6.2.2 Кэширование с отложенной записью 72
6.2.3 Интеграция RDBMS 73
6.2.3.1 CacheJdbcPojoStore 74
6.2.3.2 CacheJdbcBlobStore 76
6.2.4 Загрузка данных 76

6.2.5 Интеграция с базой данных NoSQL 77
6.2.5.1 Интеграция Cassandra 77
6.3 Конфигурация снимков 77
6.3.1 Настройка конфигурации снимков 77
6.4 Сжатие диска 78
6.4.1 Поддерживаемые алгоритмы 78
6.5 Change Data Capture (CDC) 79
6.5.1 Конфигурация 80
6.5.1.1 Узел РЕД КВАНТ 80
6.5.1.2 Приложение CDC 80
6.5.2 API 81
6.5.2.1 org.apache.ignite.cdc.CdcEvent 81
6.5.2.2 org.apache.ignite.cdc.CdcConsumer 81
6.5.3 Метрики 82
6.5.4 Ведение журнала 82
6.5.5 Жизненный цикл 83
6.5.6 cdc-ext 83

7 КОНФИГУРАЦИЯ СНИМКОВ 84

7.1 Конфигурация 85
7.1.1 Каталог снимков 85
7.1.2 Пул выполнения моментальных снимков 85
7.2 Создание снимка 86
7.2.1 Использование сценария управления 86
7.2.2 Использование JMX 86
7.2.3 Использование Java API 86
7.3 Проверка согласованности снимков 87
7.4 Восстановление из снимка 87
7.4.1 Процедура ручного восстановления моментального снимка 87
7.4.2 Процедура автоматического восстановления моментального снимка 88
7.4.2.1 Восстановление группы кэша из моментального снимка 88
7.4.2.2 Использование интерфейса командной строки для управления операцией восстановления 89
7.5 Гарантии согласованности 89
7.6 Текущие ограничения 89

8 НАСТРОЙКА ТАБЛИЦ КЭШЕЙ 91

8.1 Конфигурация кэшей 91
8.1.1 Пример конфигурации 91
8.1.2 Шаблоны кэша 92
8.2 Конфигурация бэкапов разделов 93
8.2.1 Настройка резервных копий 93
8.2.2 Синхронные и асинхронные резервные копии 94
8.3 Политика потери раздела 94
8.3.1 Настройка политики потери раздела 95
8.3.2 Прослушивание событий потери раздела 95
8.3.3 Обработка потери раздела 96
8.3.4 Восстановление после потери раздела 97
8.3.4.1 Чистый in-memory кластер с политикой IGNORE 97
8.3.4.2 Чистый in-memory кластер с политикой READ_WRITE_SAFE или READ_ONLY_SAFE 97

8.3.4.3 Кластеры с персистентностью 97
8.3.4.4 Кластеры с кэшем в памяти и постоянным кэшем 98
8.4 Режимы атомарности 98

9 НАСТРОЙКА ПАРАМЕТРОВ РЕБАЛАНСА 100

9.1 Настройка режима ребалансировки 100
9.2 Настройка пула потоков ребалансировки 101
9.3 Ребалансировка регулирования сообщений 102
9.4 Другие свойства 103
9.5 Мониторинг процесса ребалансировки 103

10 НАСТРОЙКА ПРОИЗВОДИТЕЛЬНОСТИ И ОСНОВНЫЕ ПРОБЛЕМЫ 104

10.1 Общие советы по повышению производительности 104
10.2 Настройка производительности RAM и JVM 105
10.2.1 Настройка параметров подкачки 105
10.2.2 Совместное использование оперативной памяти с ОС и приложениями 105
10.2.3 Настройка Java Heap и GC 106
10.2.3.1 Общие настройки GC 106
10.2.3.2 Расширенная настройка памяти 107
10.2.3.3 Расширенная настройка ввода/вывода 108
10.3 Настройка производительности слоя хранения 108
10.3.1 Настройка размера страницы 108
10.3.2 Хранение WAL отдельно 109
10.3.3 Увеличение размера сегмента WAL 109
10.3.4 Изменение режима WAL 109
10.3.5 Отключение WAL 110
10.3.6 Троттлинг записи страниц 110
10.3.7 Настройка размера буфера checkpointing 111
10.3.8 Включение прямого ввода/вывода 112
10.3.9 Приобретение твердотельных накопителей производственного уровня 112
10.3.10 Резервное копирование SSD 112
10.4 Настройка производительности SQL 113
10.4.1 Основные тезисы: РЕД КВАНТ и RDBMS 113
10.4.2 Использование оператора EXPLAIN 114
10.4.3 Оператор OR и селективность 114
10.4.4 Отказ от слишком большого числа столбцов 114
10.4.5 Ленивая загрузка 114
10.4.6 Запрос совмещенных данных 115
10.4.7 Принудительный порядок присоединения 115
10.4.8 Увеличение размера inline индекса 117
10.4.9 Параллелизм запросов 119
10.4.10 Подсказки по индексам 119
10.4.11 Обрезка раздела 120
10.4.12 Пропуск редуктора при обновлении 120
10.4.13 Кэш строк SQL в heap 121
10.4.14 Использование TIMESTAMP вместо DATE 122
10.5 Настройка производительности пулов потоков 122
10.5.1 Системный пул 122
10.5.2 Пул запросов 122
10.5.3 Public пул 122
10.5.4 Сервисный пул 122

10.5.5 Чередующийся пул ... 123
10.5.6 Пул потоков данных ... 123
10.5.7 Snapshot пул ... 123
10.5.8 Создание пользовательского пула потоков ... 123
10.6 Отладка и выявление проблем ... 124
10.6.1 Средства отладки: Команда проверки согласованности ... 124
10.6.2 Файлы сохранения исчезают при перезапуске ... 124
10.6.3 Отладка проблем GC ... 125
10.6.3.1 Дампы heap ... 125
10.6.3.2 Подробные журналы GC ... 125
10.6.3.3 Анализ производительности с помощью Flight Recorder ... 126
10.6.3.4 Паузы JVM ... 126
10.7 Обработка исключений ... 126
10.7.1 Обработка исключений РЕД КВАНТ ... 126
10.7.2 Обработка критических сбоев ... 129
10.7.2.1 Критические сбои ... 129
10.7.2.2 Обработка сбоев ... 130
10.7.2.3 Проверка работоспособности критически важных рабочих процессов ... 130
10.8 Бенчмарки ... 131
10.8.1 Локальный запуск РЕД КВАНТ Benchmarks ... 132
10.8.2 Удаленный запуск тестов РЕД КВАНТ ... 132
10.8.3 Существующие контрольные показатели ... 133
10.8.4 Свойства и аргументы командной строки ... 134
10.8.5 Создание из источников ... 135
10.8.6 Пользовательские тесты РЕД КВАНТ ... 135
ЛИСТ РЕГИСТРАЦИИ ИЗМЕНЕНИЙ ... 136

ВВЕДЕНИЕ

РЕД КВАНТ — это ориентированная на память распределенная база данных, кэширующая и вычислительная платформа для разработки приложений, интенсивно использующих данные, масштабируемые до сотен миллионов транзакций в секунду и петабайт данных в памяти. Эта ориентированная на память платформа следующего поколения может функционировать как data grid в памяти или может быть развернута как полнофункциональное постоянное хранилище данных с поддержкой транзакций SQL и ACID.

Основной принцип платформы РЕД КВАНТ заключается в том, чтобы хранить весь набор данных в памяти и на диске, повышая производительность приложений за счет возможностей кэширования и параллельной обработки данных. Такой принцип был определен как решение проблем производительности традиционных баз данных на основе дисков, поскольку платформа позволяет загружать и выполнять все необходимые наборы данных в памяти. Этот принцип устраняет значительное количество дисковых операций ввода-вывода, которые препятствуют транзакциям и создают узкое место в производительности традиционных систем управления баз данных (СУБД).

Помимо определения, упомянутого выше, РЕД КВАНТ также может использоваться как:

1. Полная in-memory база данных: РЕД КВАНТ может быть развернута как распределенная база данных с поддержкой SQL поверх нереляционной модели данных, всякий раз, когда отключено сохранение на собственном диске.

2. Мультимодельная база данных: под капотом РЕД КВАНТ хранит данные в хранилище «ключ-значение». РЕД КВАНТ хранит данные на специальных страницах памяти, где вся память разбивается на страницы, внутри которых хранятся записи типа «ключ-значение». РЕД КВАНТ предоставляет SQL для моделирования и доступа к данным, что позволяет платформе действовать как мультимодельная база данных.

3. Транзакционная база данных: РЕД КВАНТ поддерживает транзакции на уровне API «ключ-значение», которые могут охватывать разные разделы на разных серверах.

4. Платформа микросервисов: благодаря возможности хранить данные и предоставлять сервисы для обработки данных с одного и того же вычислительного ресурса, РЕД КВАНТ является первым в области разработки отказоустойчивых, масштабируемых микросервисов.

5. Фабрики in-memory данных: РЕД КВАНТ также предоставляет такие функции, как управление большими данными, кластеризация веб-сеансов, Spring кэш, и может использоваться в качестве реализации для диспетчера кластеров.

Учитывая эти серьезные обещания, лучшими вариантами использования РЕД КВАНТ являются следующие:

1. Обработка больших объемов ACID-транзакций.
2. Кэш как сервис (CaaS).
3. Кэширование базы данных.
4. Обнаружение мошенничества в режиме онлайн.

5. Complex event processing (CEP) - Обработка сложных событий для IoT-проектов.
6. Аналитика в реальном времени.
7. Бизнес-приложения HTAP.
8. Быстрая обработка данных или реализация лямбда-архитектуры.

Таким образом, РЕД КВАНТ может хранить и обрабатывать большие объемы данных в памяти и на диске благодаря своей надежной архитектуре памяти. Он может перезагружать свое состояние всякий раз, когда перезапускается узел РЕД КВАНТ, либо с жесткого диска, либо по сети из постоянного хранилища. Это все еще база данных в памяти, несмотря на запись на диск, потому что диск используется только как журнал, для надежности доступный только для добавления. РЕД КВАНТ имеет следующие ключевые особенности:

1. Compute Grid
2. Service Grid.
3. SQL Grid.
4. Ускоритель для работы с большими данными.
5. Streaming grid.
6. Машинное обучение.
7. Долговременная память.
8. Стороннее постоянное хранилище.
9. Поддержка ORM (Hibernate OGM и Spring Data).

На рисунке 1 показаны основные функции платформы РЕД КВАНТ.

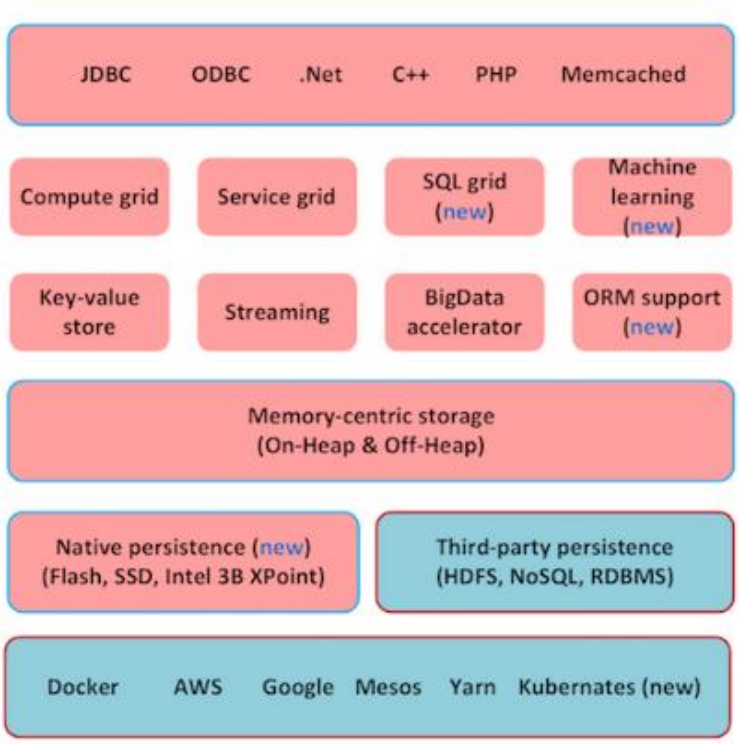

Рисунок 1 - Функции платформы РЕД КВАНТ

Основные технологии РЕД КВАНТ:

1. Написан на Java и C++.
2. Интегрируется со Spring.
3. База данных на основе H2 и Apache Calcite SQL движков.
4. Apache Lucene (полнотекстовый поиск).
5. Поддерживает JDBC, ODBC, .Net, C++, PHP и т. д.

Основные возможности, которые предоставляет РЕД КВАНТ:

- **Эластичность**. Кластер КВАНТ может увеличиваться горизонтально за счет добавления новых узлов. Иногда это называется масштабированием или горизонтальным масштабированием. РЕД КВАНТ масштабируется до тысяч узлов на обычном оборудовании.
- **Надежность**. РЕД КВАНТ предоставляет надежные механизмы для предотвращения потери данных за счет устойчивости на основе дисков. У диска есть два существенных преимущества: они постоянны (при отключении питания данные не потеряются) и имеют меньшую стоимость за гигабайт, чем оперативная память. РЕД КВАНТ постоянно записывает транзакции в файл, доступный только для добавления (WAL) и периодически сбрасывает изменения на диск.
- **Поддержка SQL**. РЕД КВАНТ специально разработан для выполнения SQL на большом объеме данных. РЕД КВАНТ поставляется с совместимой с ANSI-99, горизонтально масштабируемой и отказоустойчивой распределенной базой данных SQL.

- **Децентрализованность**. Нет единой точки отказа или SPOF; каждый узел в кластере идентичен и хранит несколько копий данных.
- **Отказоустойчивость**. Данные автоматически реплицируются на несколько узлов с помощью конфигурации резервной копии. Вышедшие из строя узлы могут быть заменены без простоев.
- **Кэш как сервис (CaaS)**. РЕД КВАНТ поддерживает кэширование как сервис во всей организации, что позволяет нескольким приложениям из разных отделов получать доступ к управляемому кэшу в памяти вместо медленных баз данных на дисках.
- **Кэш 2-го уровня**. РЕД КВАНТ — идеальный уровень кэширования для использования в качестве распределенного кэша 2-го уровня в Hibernate или MyBatis. Этот кэш не ограничивается одним сеансом, а распределяется между сеансами, поэтому данные доступны для всего приложения, а не только для текущего пользователя. Кэш 2-го уровня может значительно повысить производительность приложений, поскольку часто используемые данные могут храниться в памяти на уровне приложений.
- **Совместное использование состояния в памяти между приложениями Spark**. Ориентированная на память архитектура РЕД КВАНТ обеспечивает эффективное совместное использование RDD с IgniteContext и IgniteRDD между приложениями Spark. Ignite RDD позволяет легко обмениваться состояниями в памяти между различными заданиями или приложениями Spark. Любое приложение Spark может поместить данные в кэш КВАНТ с помощью общих Ignite RDD в памяти, которые позже будут доступны другому приложению Spark. РЕД КВАНТ также поддерживает современный API DataFrame Apache Spark. Можно обмениваться данными и состоянием между заданиями Spark с поддержкой Spark DataFrame, записывая и считывая фреймы данных в РЕД КВАНТ или из него.
- **Распределенные вычисления**. РЕД КВАНТ предоставляет набор простых API-интерфейсов, которые позволяют пользователю распределять вычисления и обработку данных между несколькими узлами кластера для повышения производительности. Распределенные сервисы РЕД КВАНТ очень полезны для разработки и реализации микросервисной архитектуры.
- **Потоковая передача**. РЕД КВАНТ позволяет обрабатывать непрерывные бесконечные потоки данных масштабируемым и отказоустойчивым способом в памяти, а не анализировать данные после их сохранения в базе данных.
- **Стороннее постоянное хранилище**. РЕД КВАНТ может сохранять записи кэша в СУБД, даже в NoSQL, например MongoDB или Cassandra.
- **Поддержка плагинов**. Встроенная система подключаемых модулей РЕД КВАНТ позволила третьим сторонам расширить основные функциональные возможности РЕД КВАНТ. Пользователь может реализовать такие функции, как аутентификация и авторизация в РЕД КВАНТ, используя систему плагинов.

1 СИСТЕМНЫЕ ТРЕБОВАНИЯ

Системные требования для запуска РЕД КВАНТ:

- JDK: Oracle JDK 8 или 11, Open JDK 8 или 11, IBM JDK 8 или 11;
- ОС: Linux (любая версия), Mac OSX (10.6 и выше), Windows (XP и выше), Windows Server (2008 и выше), Oracle Solaris;
- ISA: x86, x64, SPARC, PowerPC;
- Сеть: Без ограничений (рекомендуется 10G).

2 УСТАНОВКА И НАСТРОЙКА

2.1 Установка из ZIP-архива

1. Собрать проект «РЕД КВАНТ».

Для сборки необходимо выполнить следующие команды (на компьютере должен быть установлен maven, а также java 1.8):

```
mvn clean install -Pall-java,licenses -DskipTests
mvn initialize -Prelease
```

В каталоге *./target/bin* появится архив *apache-ignite-<version>-bin.zip*.

Проект программного продукта «РЕД КВАНТ» собран.

2. Разархивировать zip-архив в каталог установки.
3. (Необязательно) Включить необходимые модули.

2.2 Установка из Docker

1. Собрать проект «РЕД КВАНТ».

Для сборки необходимо выполнить следующие команды (на компьютере должен быть установлен maven, а также java 1.8):

```
mvn clean install -Pall-java,licenses -DskipTests
mvn initialize -Prelease
```

В каталоге *./target/bin* появится архив *apache-ignite-<version>-bin.zip*.

Проект программного продукта «РЕД КВАНТ» собран.

2. Перейти в каталог *docker/apache-ignite* и скопировать в него ранее созданный архив *../target/bin/apache-ignite-*.zip*. Распаковать архив.
3. Собрать *docker*-образ. Для этого на компьютере должен быть установлен *Docker*.

Выполнить команду:

```
docker build . -t apacheignite/ignite[:<version>]
```

4. Проверить, что *docker*-образ собрался, с помощью команды:

```
docker images
```

В результате должен быть получен следующий вывод (рисунок 2).

```
REPOSITORY            TAG       IMAGE ID       CREATED          SIZE
apacheignite/ignite   2.8.1.9   a889ad884b03   23 seconds ago   170MB
```

Рисунок 2 - Результат выполнения команды docker images

5. Запустить контейнер.

Docker-образ предоставляет возможность использовать порты 11211, 47100, 47500, 49112, 10800, 8080, которые также могут быть открыты при запуске контейнера.

Например, чтобы подключить тонкий клиент к узлу, работающему внутри контейнера, нужно открыть порт 10800:

```
docker run –name kvant -d -p 10800:10800 apacheignite/ignite[:<version>]
```

И проверить, что контейнер запущен, с помощью команды:

```
docker ps
```

В результате должен быть получен следующий вывод (рисунок 3).

```
CONTAINER ID   IMAGE                          COMMAND                  CREATED              STATUS
PORTS                                                                                  NAMES
dcc7ab8993af   apacheignite/ignite:2.8.1.9    "/bin/sh -c $IGNITE_…"   About a minute ago   Up About a minute
8080/tcp, 11211/tcp, 47100/tcp, 47500/tcp, 49112/tcp, 0.0.0.0:10800->10800/tcp   kvant
```

Рисунок 3 - Результат выполнения команды docker ps

2.3 Базовая настройка и конфигурация

Можно указать пользовательские параметры конфигурации, используя экземпляр класса IgniteConfiguration для РЕД КВАНТ при запуске узла. Можно установить параметры либо программно, либо через файл конфигурации XML. Эти 2 способа полностью взаимозаменяемы.

Файл конфигурации XML — это файл определения компонента Spring, который должен содержать компонент IgniteConfiguration. При запуске узла из командной строки нужно передать файл конфигурации в качестве параметра сценарию ignite.sh|bat следующим образом:

```
ignite.sh ignite-config.xml
```

Если не указать файл конфигурации, будет использоваться файл по умолчанию {IGNITE_HOME}/config/default-config.xml.

2.3.1 XML-конфигурация Spring

Чтобы создать конфигурацию в формате Spring XML, необходимо определить компонент IgniteConfiguration и установить параметры, которые должны отличаться от параметров по умолчанию.

В приведенном ниже примере создается компонент IgniteConfiguration, устанавливается свойство workDirectory и настраивается секционированный кэш.

```
<?xml version="1.0" encoding="UTF-8"?>


<beans xmlns="http://www.springframework.org/schema/beans"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="
       http://www.springframework.org/schema/beans
       http://www.springframework.org/schema/beans/spring-beans.xsd">

    <bean class="org.apache.ignite.configuration.IgniteConfiguration">
       <property name="workDirectory" value="/path/to/work/directory"/>
```

```
        <property name="cacheConfiguration">
          <bean class="org.apache.ignite.configuration.CacheConfiguration">
            <!-- Set the cache name. -->
            <property name="name" value="myCache"/>
            <!-- Set the cache mode. -->
            <property name="cacheMode" value="PARTITIONED"/>
            <!-- Other cache parameters. -->
          </bean>
        </property>
      </bean>
    </beans>
```

2.3.2 Программная конфигурация

Необходимо создать экземпляр класса IgniteConfiguration и задать необходимые параметры, как показано в примере ниже.

```
IgniteConfiguration igniteCfg = new IgniteConfiguration();

//setting a work directory
igniteCfg.setWorkDirectory("/path/to/work/directory");

//defining a partitioned cache
CacheConfiguration cacheCfg = new CacheConfiguration("myCache");
cacheCfg.setCacheMode(CacheMode.PARTITIONED);

igniteCfg.setCacheConfiguration(cacheCfg);
```

2.3.3 Настройка

2.3.3.1 Запуск РЕД КВАНТ с Java 11 или более поздней версии

Чтобы запустить РЕД КВАНТ с Java 11, необходимо выполнить следующие действия:

1. Установите переменную среды JAVA_HOME, чтобы она указывала на каталог установки Java.

2. РЕД КВАНТ использует собственные API-интерфейсы SDK, недоступные по умолчанию. Нужно передать определенные флаги в JVM, чтобы сделать эти API доступными. Если используется сценарий запуска ignite.sh (или ignite.bat для Windows), ничего делать не нужно, поскольку эти флаги уже установлены в сценарии. В противном случае нужно указать следующие параметры для JVM приложения:

```
--add-exports=java.base/jdk.internal.misc=ALL-UNNAMED

--add-exports=java.base/sun.nio.ch=ALL-UNNAMED
--add-exports=java.management/com.sun.jmx.mbeanserver=ALL-UNNAMED
--add-exports=jdk.internal.jvmstat/sun.jvmstat.monitor=ALL-UNNAMED
--add-exports=java.base/sun.reflect.generics.reflectiveObjects=ALL-UNNAMED
--add-opens=jdk.management/com.sun.management.internal=ALL-UNNAMED
--illegal-access=permit
```

2.3.3.2 Использование двоичного дистрибутива

1. Необходимо собрать проект «РЕД КВАНТ».

2. Распаковать архив в каталог.

3. (Необязательно) Установить переменную среды IGNITE_HOME так, чтобы она указывала на папку установки, и убедиться, что в пути нет конечного /.

2.3.3.3 Использование Maven

Самый простой способ использовать РЕД КВАНТ — добавить его в файл pom , а также добавить репозитарий nexus.

```
<properties>

  <ignite.version>2.13.0</ignite.version>
</properties>

<dependencies>
  <dependency>
    <groupId>org.apache.ignite</groupId>
    <artifactId>ignite-core</artifactId>
    <version>${ignite.version}</version>
  </dependency>
</dependencies>
```

Библиотека ignite-core содержит основные функции РЕД КВАНТ. Дополнительную функциональность обеспечивают различные модули РЕД КВАНТ.

Ниже приведены два наиболее часто используемых модуля:

- ignite-spring (поддержка конфигурации на основе XML);
- ignite-indexing (поддержка индексации SQL).

```
<dependency>
  <groupId>org.apache.ignite</groupId>
  <artifactId>ignite-spring</artifactId>
  <version>${ignite.version}</version>
</dependency>

<dependency>
  <groupId>org.apache.ignite</groupId>
  <artifactId>ignite-indexing</artifactId>
  <version>${ignite.version}</version>
</dependency>
```

2.3.3.4 Использование Docker

Если необходимо запустить РЕД КВАНТ в Docker, см. пункт Установка из Docker.

2.3.3.5 Настройка рабочего каталога

РЕД КВАНТ использует рабочий каталог для хранения данных приложения (если используется функция встроенного сохранения), индексных файлов, информации о метаданных, журналов и других файлов. Рабочий каталог по умолчанию выглядит следующим образом:

- $IGNITE_HOME/work, если определено системное свойство IGNITE_HOME. Это тот случай, когда РЕД КВАНТ запускается с помощью скрипта bin/ignite.sh из дистрибутива.
- ./ignite/work, этот путь относится к каталогу, в котором запускается приложение.

Есть несколько способов изменить рабочий каталог по умолчанию:

1. в качестве переменной среды:

```
export IGNITE_WORK_DIR=/path/to/work/directory
```

2. в конфигурации узла:

```
<bean class="org.apache.ignite.configuration.IgniteConfiguration">

  <property name="workDirectory" value="/path/to/work/directory"/>
  <!-- other properties -->
</bean>
```

2.3.3.6 Включение модулей

РЕД КВАНТ поставляется с рядом модулей и имеет множество расширений, обеспечивающих различные функции. Можно включать модули или расширения по одному по мере необходимости.

Все модули включены в бинарный дистрибутив, но по умолчанию они отключены (кроме модулей ignite-core, ignite-spring и ignite-indexing). Модули находятся в каталоге lib/optional дистрибутива (каждый модуль находится в отдельном подкаталоге).

Кроме того, можно загрузить любое необходимое расширение РЕД КВАНТ.

В зависимости от того, как используется РЕД КВАНТ, можно включить модули или расширения одним из следующих способов:

- Если используется бинарный дистрибутив, перед запуском узла необходимо переместить lib/optional/{module-dir} в каталог libs.
- Добавить библиотеки из lib/Optional/{module-dir} в путь к классам приложения.
- Добавить модуль в качестве зависимости Maven в проект.

```
<dependency>
  <groupId>org.apache.ignite</groupId>
  <artifactId>ignite-log4j2</artifactId>
  <version>${ignite.version}</version>
```

```
    </dependency>
```

Следующие модули имеют зависимости LGPL и поэтому не могут быть развернуты в репозитории Maven Central:

- ignite-hibernate;
- ignite-geospatial;
- ignite-schedule.

Чтобы использовать эти модули, нужно собрать их из исходников и добавить в проект. Например, чтобы установить ignite-hibernate в локальный репозиторий, нужно запустить эту команду в исходном пакете РЕД КВАНТ:

```
./mvnw clean install -DskipTests -Plgpl -pl modules/hibernate -am
```

Перечень доступных модулей представлен в таблице 1.

Таблица 1 - Доступные модули

Идентификатор модуля (artifactId)	**Описание**
ignite-aop	Модуль Ignite AOP предоставляет возможность превратить любой метод Java в распределенное замыкание, добавив к нему аннотацию @Gridify.
ignite-cassandra-serializers	Модуль Ignite Cassandra Serializers предоставляет дополнительные сериализаторы для хранения объектов в виде больших двоичных объектов в Cassandra. Модуль можно использовать как в связке с модулем Ignite Cassandra Store.
ignite-cassandra-store	Ignite Cassandra Store предоставляет реализацию CacheStore, поддерживаемую базой данных Cassandra.
ignite-cloud	Ignite Cloud предоставляет реализации Apache jclouds средства поиска IP-адресов для TCP Discovery.
ignite-direct-io	Ignite Direct IO — это плагин, который предоставляет хранилищу страниц возможность записи и чтения разделов кэша в режиме O_DIRECT.
ignite-indexing	SQL-запросы и индексация.
ignite-jcl	Поддержка платформы Jakarta Common Logging (JCL).
ignite-jta	Интеграция транзакций РЕД КВАНТ с JTA.
ignite-kafka	Ignite Kafka Streamer обеспечивает возможность потоковой передачи данных из Kafka в кэши РЕД КВАНТ.
ignite-kubernetes	Модуль Ignite Kubernetes предоставляет средство поиска IP-адресов TCP Discovery, которое использует выделенную службу

Идентификатор модуля (artifactId)	Описание
	Kubernetes для поиска IP-адресов модулей РЕД КВАНТ, контейнеризованных Kubernetes.
ignite-log4j	Поддержка Log4j
ignite-log4j2	Поддержка Log4j2
ignite-ml	Ignite ML Grid предоставляет функции машинного обучения и соответствующие структуры данных и методы линейной алгебры, в том числе в куче и вне кучи, плотные и разреженные, локальные и распределенные реализации.
ignite-osgi	Этот модуль предоставляет связующие компоненты для беспрепятственной работы РЕД КВАНТ внутри контейнера OSGi, такого как Apache Karaf.
ignite-osgi-karaf	Этот модуль содержит репозиторий функций, облегчающий установку РЕД КВАНТ в контейнер Apache Karaf.
ignite-osgi-paxlogging	Этот модуль представляет собой фрагмент OSGi, который предоставляет следующие пакеты из пакета Pax Logging API: org.apache.log4j.varia org.apache.log4j.xml Эти пакеты необходимы при установке пакета ignite-log4j и по умолчанию не предоставляются Pax Logging API — платформой ведения журналов, используемой Apache Karaf.
ignite-rest-http	Ignite REST-HTTP запускает сервер на основе Jetty в узле, который можно использовать для выполнения задач и/или кэширования команд в сетке с использованием API-интерфейсов RESTful на основе HTTP.
ignite-scalar	Модуль Ignite Scalar предоставляет DSL на основе Scala с расширениями и ярлыками для Ignite API.
ignite-scalar_2.10	Модуль Ignite Scalar, поддерживающий Scala 2.10.
ignite-schedule	Этот модуль предоставляет функциональные возможности для локального планирования заданий с использованием синтаксиса UNIX cron.
ignite-slf4j	Поддержка платформы ведения журнала SLF4J.
ignite-spark	Этот модуль предоставляет реализацию абстракции Spark RDD, которая обеспечивает легкий доступ к кэшам РЕД КВАНТ.
ignite-ssh	Модуль Ignite SSH предоставляет возможности для запуска узлов РЕД КВАНТ на удаленных машинах через SSH.
ignite-tensorflow	Модуль интеграции Ignite TensorFlow позволяет использовать TensorFlow с РЕД КВАНТ. В этом сценарии РЕД КВАНТ будет источником данных для обучения любой модели TensorFlow.

Идентификатор модуля (artifactId)	Описание
ignite-urideploy	Модуль Ignite URI Deploy предоставляет возможности для развертывания задач из разных источников, таких как файловая система, HTTP или даже электронная почта.
ignite-visor-console	Инструмент управления и мониторинга командной строки с открытым исходным кодом
ignite-web	Ignite Web позволяет запускать узлы внутри любого веб-контейнера на основе сервлета и слушателя контекста сервлета. Кроме того, этот модуль предоставляет возможность кэшировать веб-сессии в кэше РЕД КВАНТ.
ignite-zookeeper	Реализация ZooKeeper Discovery.

Перечень доступных расширений представлен в таблице 2.

Таблица 2 - Доступные расширения

Идентификатор артефакта расширения	Описание
ignite-aws-ext	Обнаружение кластера на AWS S3.
ignite-azure-ext	Ignite Azure предоставляет реализацию средства поиска IP-адресов для обнаружения TCP на основе хранилища BLOB-объектов Azure.
ignite-gce-ext	Ignite GCE предоставляет реализацию средства поиска IP-адресов на основе Google Cloud Storage для обнаружения TCP.
ignite-spring-data-ext	Ignite Spring Data обеспечивает интеграцию со средой Spring Data.
ignite-spring-data_2.0-ext	Ignite Spring Data 2.0 обеспечивает интеграцию со средой Spring Data 2.0.
ignite-spring-data_2.2-ext	Ignite Spring Data 2.2 обеспечивает интеграцию со средой Spring Data 2.2.
ignite-zookeeper-ip-finder-ext	TCP Discovery IP Finder, который использует каталог ZooKeeper для поиска других узлов РЕД КВАНТ для подключения.

2.3.3.7 Настройка параметров JVM

Существует несколько способов, которыми можно установить параметры JVM при запуске узла с помощью скрипта ignite.sh:

- с помощью системной переменной JVM_OPTS. Можно установить переменную окружения JVM_OPTS:

```
export JVM_OPTS="$JVM_OPTS -Xmx6G -DIGNITE_TO_STRING_INCLUDE_SENSITIVE=false"; $IGNITE_HOME/bin/ignite.sh
```

- через аргументы командной строки. Можно передавать параметры JVM, используя префикс -J:

```
./ignite.sh -J-Xmx6G -J-DIGNITE_TO_STRING_INCLUDE_SENSITIVE=false
```

2.3.3.8 Настройка свойств системы РЕД КВАНТ

В дополнение к общедоступным параметрам конфигурации можно настроить конкретное, обычно низкоуровневое поведение РЕД КВАНТ с помощью внутренних свойств системы. Найти все свойства с их описаниями и значениями по умолчанию можно, используя следующую команду:

```
./ignite.sh -systemProps
```

2.3.4 РЕКОМЕНДАЦИИ ПО НАСТРОЙКЕ

Ниже приведены некоторые советы по настройке, направленные на то, чтобы упростить работу с кластером РЕД КВАНТ или разработку приложений с помощью РЕД КВАНТ.

2.3.4.1 Настройка рабочего каталога

Если используется либо бинарный дистрибутив, либо Maven, рекомендуется настроить рабочий каталог для РЕД КВАНТ. Рабочий каталог используется для хранения метаданных, индексных файлов, данных приложения (если используется функция встроенного сохранения), журналов и других файлов. Рекомендуется всегда настраивать рабочий каталог.

2.3.4.2 Рекомендуемая конфигурация ведения журнала

Журналы играют важную роль, когда речь идет об устранении неполадок и обнаружении того, что пошло не так. Вот несколько общих советов о том, как управлять файлами журналов:

- Необходимо запустить РЕД КВАНТ в подробном режиме:
  - Если используется ignite.sh, нужно указать параметр -v.
  - Если РЕД КВАНТ запускается из кода Java, нужно установить следующую системную переменную: IGNITE_QUIET=false.
- Не нужно хранить файлы журналов в папке /tmp. Эта папка очищается каждый раз при перезапуске сервера.
- Необходимо убедиться, что в хранилище, где хранятся файлы журналов, достаточно свободного места.
- Необходимо периодически архивировать старые файлы журналов, чтобы сэкономить место для хранения.

2.4 НАСТРОЙКА ЛОГГИРОВАНИЯ

РЕД КВАНТ поддерживает ряд библиотек и фреймворков ведения журналов:

- JUL (default),
- Log4j,
- Log4j2,
- JCL,
- SLF4J.

Ниже представлено, как настроить регистратор.

Когда узел запускается, он выводит на консоль информацию о запуске, включая информацию о настроенной библиотеке ведения журнала. Каждая библиотека ведения журнала имеет свои собственные параметры конфигурации и должна быть настроена в соответствии с ее официальной документацией. Помимо конфигурации, зависящей от конкретной библиотеки, существует ряд системных свойств, позволяющих настраивать ведение журнала. Эти свойства представлены в таблице 3.

Таблица 3 - Системные свойства для настройки ведения журнала

Системное свойство	Описание	Значение по умолчанию
IGNITE_LOG_INSTANCE_NAME	Если свойство задано, РЕД КВАНТ включает имя своего экземпляра в сообщения журнала.	Не задано
IGNITE_QUIET	Необходимо установить значение false, чтобы отключить тихий режим и включить подробный режим. В подробном режиме узел регистрирует намного больше информации.	true
IGNITE_LOG_DIR	Каталог, в который РЕД КВАНТ записывает файлы журнала.	$IGNITE_HOME/work/log
IGNITE_DUMP_THREADS_ON_FAILURE	Необходимо установить значение true, чтобы выводить дампы потоков в журнал при обнаружении критической ошибки.	true

2.4.1 ВЕДЕНИЕ ЖУРНАЛА ПО УМОЛЧАНИЮ

По умолчанию РЕД КВАНТ использует фреймворк java.util.logging (JUL). Если РЕД КВАНТ запускается с помощью скрипта ignite.sh|bat из дистрибутива, РЕД КВАНТ использует $IGNITE_HOME/config/java.util.logging.properties в качестве файла конфигурации ведения журнала по умолчанию и выводит все сообщения в файлы журнала в папке $IGNITE_HOME/work/log. Можно переопределить каталог журналов по умолчанию, указав системное свойство IGNITE_LOG_DIR.

Если РЕД КВАНТ используется в приложении в качестве библиотеки, конфигурация ведения журнала по умолчанию включает только обработчик консоли на уровне INFO. Пользовательский файл конфигурации можно указать с помощью системного свойства java.util.logging.config.file.

2.4.2 Использование Log4j

Внимание! Модуль ignite-log4j устарел и будет удален в следующих выпусках. Срок службы Log4j 1.x истек, и содержит критические уязвимости. Вместо этого необходимо использовать модуль ignite-log4j2.

Чтобы включить регистратор Log4j, необходимо установить свойство gridLogger IgniteConfiguration, как показано в следующем примере:

```
<bean class="org.apache.ignite.configuration.IgniteConfiguration" id="ignite.cfg">

  <property name="gridLogger">
    <bean class="org.apache.ignite.logger.log4j.Log4JLogger">
      <!-- log4j configuration file -->
      <constructor-arg type="java.lang.String" value="log4j-config.xml"/>
    </bean>
  </property>

  <!-- other properties -->

</bean>
```

В приведенном выше примере путь к log4j-config.xml может быть либо абсолютным путем, либо локальным путем относительно META-INF в пути к классам, либо путем к IGNITE_HOME. Пример файла конфигурации log4j можно найти в дистрибутиве ($IGNITE_HOME/config/ignite-log4j.xml).

2.4.3 Использование Log4j2

Примечание: Перед использованием Log4j необходимо включить модуль ignite-log4j.

Чтобы включить регистратор Log4j2, необходимо установить свойство gridLogger IgniteConfiguration, как показано ниже:

```
<bean class="org.apache.ignite.configuration.IgniteConfiguration" id="ignite.cfg">

  <property name="gridLogger">
    <bean class="org.apache.ignite.logger.log4j2.Log4J2Logger">
      <!-- log4j2 configuration file -->
      <constructor-arg type="java.lang.String" value="log4j2-config.xml"/>
    </bean>
  </property>

  <!-- other properties -->

</bean>
```

В приведенном выше примере путь к log4j2-config.xml может быть либо абсолютным путем, либо локальным путем относительно META-INF в пути к классам, либо путем к IGNITE_HOME. Пример файла конфигурации log4j2 можно найти в дистрибутиве ($IGNITE_HOME/config/ignite-log4j2.xml).

Примечание: Log4j2 поддерживает реконфигурацию во время выполнения, т.е. изменения в файле конфигурации применяются без необходимости перезапуска приложения.

2.4.4 Использование JCL

Примечание: Перед использованием JCL необходимо включить модуль ignite-jcl.

Примечание: Следует отметить, что JCL просто пересылает сообщения журнала в базовую систему ведения журнала, которую необходимо правильно настроить. Например, если нужно использовать Log4j, следует убедиться, что необходимые библиотеки добавлены в путь к классам.

Чтобы включить регистратор Log4j2, нужно установить свойство gridLogger IgniteConfiguration, как показано ниже:

```
<bean class="org.apache.ignite.configuration.IgniteConfiguration" id="ignite.cfg">

  <property name="gridLogger">
    <bean class="org.apache.ignite.logger.jcl.JclLogger">
    </bean>
  </property>

  <!-- other properties -->

</bean>
```

2.4.5 Использование SLF4J

Примечание: Перед использованием SLF4J необходимо включить модуль ignite-slf4j.

Чтобы включить регистратор SLF4J, нужно установить свойство gridLogger IgniteConfiguration, как показано ниже:

```
<bean class="org.apache.ignite.configuration.IgniteConfiguration" id="ignite.cfg">

  <property name="gridLogger">
    <bean class="org.apache.ignite.logger.slf4j.Slf4jLogger">
    </bean>
  </property>

  <!-- other properties -->

</bean>
```

2.4.6 Скрытие конфиденциальной информации

Журналы могут включать содержимое записей кэша, системных свойств, параметров запуска и т. д. В некоторых случаях они могут содержать конфиденциальную информацию. Вы

можете предотвратить запись такой информации в журнал, установив для системного свойства IGNITE_TO_STRING_INCLUDE_SENSITIVE значение false.

```
./ignite.sh -J-DIGNITE_TO_STRING_INCLUDE_SENSITIVE=false
```

См. Запуск и установка узлов, чтобы узнать о различных способах установки свойств системы.

2.4.7 Пример конфигурации ведения журнала

Следующие шаги помогут настроить ведение журнала. Это должно подойти для большинства случаев.

1. Необходимо использовать Log4j или Log4j2 в качестве основы ведения журнала. Чтобы включить его, нужно следовать инструкциям, приведенным в соответствующем разделе выше.
2. Если используется файл конфигурации по умолчанию (либо ignite-log4j.xml, либо ignite-log4j2.xml), нужно раскомментировать добавление CONSOLE.
3. В файле конфигурации log4j нужно указать путь к файлу журнала. Расположение по умолчанию: ${IGNITE_HOME}/work/log/ignite.log.
4. Запустить узлы в подробном режиме:
   - Если используется ignite.sh для запуска узлов, нужно указать параметр -v.
   - Если узлы запускаются из кода Java, нужно использовать системную переменную IGNITE_QUIET=false.

3 ЗАПУСК И УСТАНОВКА УЗЛОВ

Существует два типа узлов: серверные узлы и клиентские узлы. Клиентские узлы также называются толстыми клиентами, чтобы отличать их от тонких клиентов. Серверные узлы участвуют в кэшировании, выполнении вычислений, потоковой обработке и т. д. Толстые клиенты предоставляют возможность удаленного подключения к серверам. Клиентские узлы предоставляют весь набор API-интерфейсов РЕД КВАНТ, включая кэширование, транзакции, вычисления, потоковую передачу, сервисы и т. д. со стороны клиента.

По умолчанию все узлы РЕД КВАНТ запускаются как серверные узлы, и необходимо явно включить клиентский режим.

3.1 ЗАПУСК СЕРВЕРНЫХ УЗЛОВ

Чтобы запустить серверный узел, необходимо воспользоваться следующей командой или фрагментом кода:

```
ignite.sh path/to/configuration.xml
```

3.2 ЗАПУСК КЛИЕНТСКИХ УЗЛОВ

Чтобы запустить клиентский узел, необходимо включить режим клиента в конфигурации узла:

```
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:util="http://www.springframework.org/schema/util" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xsi:schemaLocation=" http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
http://www.springframework.org/schema/util http://www.springframework.org/schema/util/spring-util.xsd">

  <bean class="org.apache.ignite.configuration.IgniteConfiguration">
    <property name="clientMode" value="true"/>

  </bean>
</beans>
```

Для удобства также можно включить или отключить режим клиента с помощью класса Ignition, чтобы клиенты и серверы могли повторно использовать одну и ту же конфигурацию.

```
Ignition.setClientMode(true);


// Start the node in client mode.
Ignite ignite = Ignition.start();
```

3.3 ЗАВЕРШЕНИЕ РАБОТЫ УЗЛОВ

Когда выполняется жесткое (принудительное) отключение узла, это может привести к потере данных или несогласованности данных и даже может помешать перезапуску узла.

Жесткое завершение работы следует использовать в качестве крайней меры, когда узел не отвечает и его нельзя завершить корректно.

Грамотное завершение работы позволяет узлу завершить критически важные операции и корректно завершить свой жизненный цикл. Правильная процедура корректного завершения работы выглядит следующим образом:

1. Необходимо остановить узел одним из следующих способов:
   - программно вызвать Ignite.close();
   - программно вызвать System.exit();
   - отправить сигнал пользовательского прерывания. РЕД КВАНТ использует ловушку завершения работы JVM для выполнения пользовательской логики до того, как JVM остановится. Если запустить узел, запустив ignite.sh и не отсоединяя его от терминала, можно остановить узел, нажав Ctrl+C.
2. Необходимо удалить узел из базовой топологии. Этот шаг может не понадобиться, если включена автоматическая настройка базового уровня.

Удаление узла из базовой топологии запускает процесс ребаланса на оставшихся узлах. Если планируется перезапуск узла вскоре после выключения, выполнять ребаланс не нужно. В этом случае не нужно удалять узлы из базовой топологии.

3.4 СОБЫТИЯ ЖИЗНЕННОГО ЦИКЛА УЗЛА

События жизненного цикла дают возможность выполнять пользовательский код на разных этапах жизненного цикла узла.

Существует 4 события жизненного цикла:

- BEFORE_NODE_START — вызывается перед запуском процедуры запуска узла;
- AFTER_NODE_START — вызывается сразу после запуска узла;
- BEFORE_NODE_STOP — вызывается перед запуском процедуры остановки узла;
- AFTER_NODE_STOP — вызывается сразу после остановки узла.

Следующие шаги описывают, как добавить пользовательский слушатель событий жизненного цикла.

1. Необходимо создать пользовательский компонент жизненного цикла, реализовав интерфейс LifecycleBean. В интерфейсе есть метод onLifecycleEvent(), который вызывается для любого события жизненного цикла.

```
public class MyLifecycleBean implements LifecycleBean {

  @IgniteInstanceResource
  public Ignite ignite;

  @Override
  public void onLifecycleEvent(LifecycleEventType evt) {
    if (evt == LifecycleEventType.AFTER_NODE_START) {
```

```
                System.out.format("After the node (consistentId = %s) starts.\n",
ignite.cluster().node().consistentId());

            }
        }
    }
```

2. Необходимо зарегистрировать реализацию в конфигурации узла.

```
    <bean class="org.apache.ignite.configuration.IgniteConfiguration">

        <property name="lifecycleBeans">
            <list>
                <bean class="org.apache.ignite.snippets.MyLifecycleBean"/>
            </list>
        </property>
    </bean>
```

4 НАСТРОЙКА КЛАСТЕРА

При запуске узлу назначается одна из двух ролей: серверный узел или клиентский узел. Серверные узлы — это рабочие единицы кластера; они кэшируют данные, выполняют вычислительные задачи и т. д. Клиентские узлы присоединяются к топологии как обычные узлы, но не хранят данные. Клиентские узлы используются для потоковой передачи данных в кластер и выполнения пользовательских запросов.

Чтобы сформировать кластер, каждый узел должен иметь возможность подключаться ко всем другим узлам. Для этого необходимо настроить соответствующий механизм обнаружения.

В дополнение к клиентским узлам можно использовать тонкие клиенты для определения и управления данными в кластере.

Узлы могут автоматически обнаруживать друг друга и формировать кластер (рисунок 4). Это позволяет при необходимости выполнять масштабирование без перезапуска всего кластера. Разработчики также могут использовать поддержку гибридного облака РЕД КВАНТ, которая позволяет устанавливать соединение между частными и общедоступными облаками, такими как Amazon Web Services, предоставляя им лучшее из обоих систем.

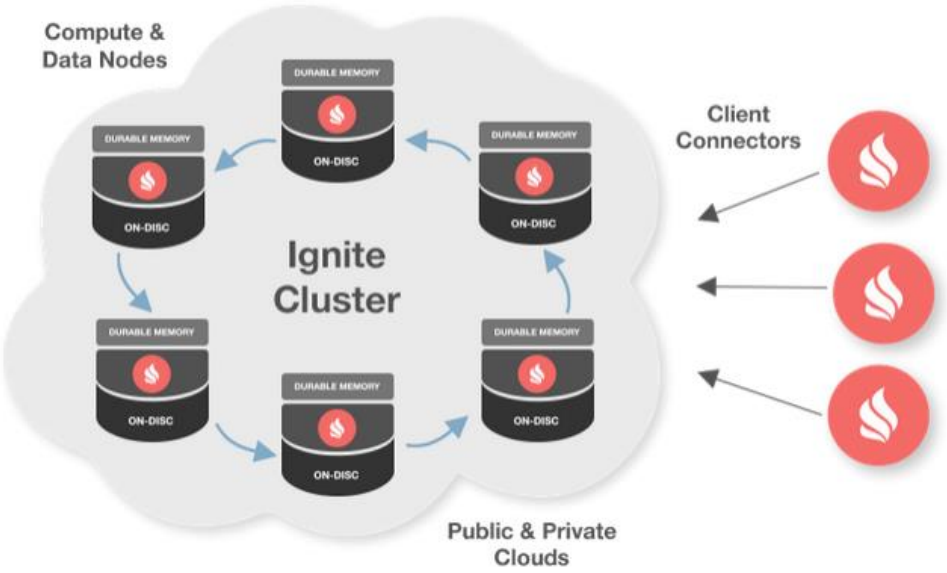

Рисунок 4 - Кластер

РЕД КВАНТ предоставляет две реализации механизма обнаружения, предназначенные для разных сценариев использования:

- Обнаружение TCP/IP разработано и оптимизировано для сотен узлов.

• ZooKeeper Discovery, который позволяет масштабировать кластеры РЕД КВАНТ до 100 и 1000 узлов, сохраняя линейную масштабируемость и производительность.

4.1 TCP/IP ОБНАРУЖЕНИЕ

В кластере РЕД КВАНТ узлы могут обнаруживать друг друга с помощью DiscoverySpi. РЕД КВАНТ предоставляет TcpDiscoverySpi в качестве реализации DiscoverySpi по умолчанию, которая использует TCP/IP для обнаружения узлов. SPI обнаружения можно настроить для multicast и статического IP-обнаружения узлов.

4.1.1 MULTICAST ПОИСК IP-АДРЕСОВ

TcpDiscoveryMulticastIpFinder использует multicast для обнаружения других узлов и является средством поиска IP-адресов по умолчанию. Вот пример того, как настроить этот искатель через XML-файл Spring или программно:

```
<bean class="org.apache.ignite.configuration.IgniteConfiguration">


  <property name="discoverySpi">
    <bean class="org.apache.ignite.spi.discovery.tcp.TcpDiscoverySpi">
      <property name="ipFinder">
        <bean class="org.apache.ignite.spi.discovery.tcp.ipfinder.multicast.TcpDiscoveryMulticastIpFinder">
          <property name="multicastGroup" value="228.10.10.157"/>
        </bean>
      </property>
    </bean>
  </property>

</bean>
```

4.1.2 СТАТИЧЕСКИЙ ПОИСК IP

Static IP Finder, реализованный в TcpDiscoveryVmIpFinder, позволяет указать набор IP-адресов и портов, которые будут проверяться на наличие узлов.

Требуется только указать как минимум один IP-адрес удаленного узла, но обычно рекомендуется указать 2 или 3 адреса узлов, которые планируется запустить в будущем. Как только соединение с любым из предоставленных IP-адресов установлено, РЕД КВАНТ автоматически обнаруживает все остальные узлы.

Примечание: Вместо указания адресов в конфигурации их можно указать в переменной окружения IGNITE_TCP_DISCOVERY_ADDRESSES или в одноименном системном свойстве. Адреса должны быть разделены запятыми и могут дополнительно содержать диапазон портов.

Примечание: По умолчанию TcpDiscoveryVmIpFinder используется в режиме «без общего доступа». Если планируется запустить серверный узел, то в этом режиме список IP-адресов должен содержать и адрес локального узла. В этом случае узел не будет ждать, пока другие узлы присоединятся к кластеру; вместо этого он станет первым узлом кластера и начнет работать в обычном режиме.

Можно настроить поиск статического IP-адреса с помощью конфигурации XML или программно:

```
<bean class="org.apache.ignite.configuration.IgniteConfiguration">

  <property name="discoverySpi">
    <bean class="org.apache.ignite.spi.discovery.tcp.TcpDiscoverySpi">
      <property name="ipFinder">
        <bean class="org.apache.ignite.spi.discovery.tcp.ipfinder.vm.TcpDiscoveryVmIpFinder">
          <property name="addresses">
            <list>
              <!--
               Explicitly specifying address of a local node to let it start and
               operate normally even if there is no more nodes in the cluster.
               You can also optionally specify an individual port or port range.
               -->
              <value>1.2.3.4</value>
              <!--
               IP Address and optional port range of a remote node.
               You can also optionally specify an individual port.
               -->
              <value>1.2.3.5:47500..47509</value>
            </list>
          </property>
        </bean>
      </property>
    </bean>
  </property>
</bean>
```

Внимание! Можно указать несколько адресов узлов, только если есть уверенность, что они доступны. Недоступные адреса увеличивают время, необходимое узлам для присоединения к кластеру. Например, было задано пять IP-адресов, и никто не прослушивает входящие соединения на двух адресах из пяти. Если РЕД КВАНТ начнет подключаться к кластеру через эти два недоступных адреса, это повлияет на время запуска узла.

4.1.3 MULTICAST И СТАТИЧЕСКИЙ ПОИСК IP-АДРЕСОВ

Можно использовать одновременно multicast и статическое IP-обнаружение. В этом случае, помимо любых адресов, полученных по multicast, TcpDiscoveryMulticastIpFinder также может работать с предварительно настроенным списком статических IP-адресов, как и описанное выше Static IP-Based Discovery. Вот пример того, как настроить Multicast IP finder со статическими IP-адресами:

```
<bean class="org.apache.ignite.configuration.IgniteConfiguration">

  <property name="discoverySpi">
    <bean class="org.apache.ignite.spi.discovery.tcp.TcpDiscoverySpi">
      <property name="ipFinder">
        <bean class="org.apache.ignite.spi.discovery.tcp.ipfinder.multicast.TcpDiscoveryMulticastIpFinder">
          <property name="multicastGroup" value="228.10.10.157"/>
          <!-- list of static IP addresses-->
          <property name="addresses">
            <list>
              <value>1.2.3.4</value>
              <!--
               IP Address and optional port range.
```

```
                    You can also optionally specify an individual port.
                    -->
                    <value>1.2.3.5:47500..47509</value>
                  </list>
                </property>
              </bean>
            </property>
          </bean>
        </property>
      </bean>
```

4.1.4 Изолированные кластеры на одном наборе машин

РЕД КВАНТ позволяет запустить два изолированных кластера на одном наборе машин. Это можно сделать, если узлы из разных кластеров используют непересекающиеся диапазоны локальных портов для TcpDiscoverySpi и TcpCommunicationSpi.

Например, нужно запустить два изолированных кластера на одной машине для целей тестирования. Для узлов из первого кластера следует использовать следующие конфигурации TcpDiscoverySpi и TcpCommunicationSpi:

```
<bean class="org.apache.ignite.configuration.IgniteConfiguration">

  <!--
  Explicitly configure TCP discovery SPI to provide list of
  initial nodes from the first cluster.
  -->
  <property name="discoverySpi">
    <bean class="org.apache.ignite.spi.discovery.tcp.TcpDiscoverySpi">
      <!-- Initial local port to listen to. -->
      <property name="localPort" value="48500"/>

      <!-- Changing local port range. This is an optional action. -->
      <property name="localPortRange" value="20"/>

      <!-- Setting up IP finder for this cluster -->
      <property name="ipFinder">
        <bean class="org.apache.ignite.spi.discovery.tcp.ipfinder.vm.TcpDiscoveryVmIpFinder">
          <property name="addresses">
            <list>
              <!--
              Addresses and port range of nodes from
              the first cluster.
              127.0.0.1 can be replaced with actual IP addresses
              or host names. Port range is optional.
              -->
              <value>127.0.0.1:48500..48520</value>
            </list>
          </property>
        </bean>
      </property>
    </bean>
  </property>

  <!--
```

```
    Explicitly configure TCP communication SPI changing local
    port number for the nodes from the first cluster.
    -->
    <property name="communicationSpi">
      <bean class="org.apache.ignite.spi.communication.tcp.TcpCommunicationSpi">
        <property name="localPort" value="48100"/>
      </bean>
    </property>
  </bean>
```

Для узлов из второго кластера конфигурация может выглядеть следующим образом:

```
  <bean id="ignite.cfg" class="org.apache.ignite.configuration.IgniteConfiguration">

    <!--
    Explicitly configure TCP discovery SPI to provide list of initial
    nodes from the second cluster.
    -->
    <property name="discoverySpi">
      <bean class="org.apache.ignite.spi.discovery.tcp.TcpDiscoverySpi">
        <!-- Initial local port to listen to. -->
        <property name="localPort" value="49500"/>

        <!-- Changing local port range. This is an optional action. -->
        <property name="localPortRange" value="20"/>

        <!-- Setting up IP finder for this cluster -->
        <property name="ipFinder">
          <bean class="org.apache.ignite.spi.discovery.tcp.ipfinder.vm.TcpDiscoveryVmIpFinder">
            <property name="addresses">
              <list>
                <!--
                Addresses and port range of the nodes from the second cluster.
                127.0.0.1 can be replaced with actual IP addresses or host names. Port range is optional.
                -->
                <value>127.0.0.1:49500..49520</value>
              </list>
            </property>
          </bean>
        </property>
      </bean>
    </property>

    <!--
    Explicitly configure TCP communication SPI changing local port number
    for the nodes from the second cluster.
    -->
    <property name="communicationSpi">
      <bean class="org.apache.ignite.spi.communication.tcp.TcpCommunicationSpi">
        <property name="localPort" value="49100"/>
      </bean>
    </property>
  </bean>
```

Как видно из конфигураций, разница между ними незначительна — различаются только номера портов для SPI и IP finder.

Если необходимо, чтобы узлы из разных кластеров могли искать друг друга по multicast протоколу, нужно заменить TcpDiscoveryVmIpFinder на TcpDiscoveryMulticastIpFinder и установить уникальные TcpDiscoveryMulticastIpFinder.multicastGroups в каждой конфигурации выше.

Внимание: Если изолированные кластеры используют встроенную персистентность, то каждый кластер должен хранить свои файлы персистентности по разным путям в файловой системе.

4.1.5 Поиск IP-адресов на основе JDBC

Примечание: Не поддерживается в .NET/C#/C++.

База данных может быть общим хранилищем исходных IP-адресов. С помощью этого средства поиска IP узлы будут записывать свои IP-адреса в базу данных при запуске. Это делается через TcpDiscoveryJdbcIpFinder.

```
<bean class="org.apache.ignite.configuration.IgniteConfiguration">


 <property name="discoverySpi">
  <bean class="org.apache.ignite.spi.discovery.tcp.TcpDiscoverySpi">
   <property name="ipFinder">
    <bean class="org.apache.ignite.spi.discovery.tcp.ipfinder.jdbc.TcpDiscoveryJdbcIpFinder">
     <property name="dataSource" ref="ds"/>
    </bean>
   </property>
  </bean>
 </property>
</bean>

<!-- Configured data source instance. -->
<bean id="ds" class="some.Datasource">

</bean>
```

4.1.6 Поиск IP-адресов в общей файловой системе

Примечание: Не поддерживается в .NET/C#/C++.

Совместно используемая файловая система может использоваться в качестве хранилища IP-адресов узлов. Узлы будут записывать свои IP-адреса в файловую систему при запуске. Это поведение поддерживается TcpDiscoverySharedFsIpFinder.

```
<bean class="org.apache.ignite.configuration.IgniteConfiguration">

  <property name="discoverySpi">
    <bean class="org.apache.ignite.spi.discovery.tcp.TcpDiscoverySpi">
      <property name="ipFinder">
        <bean class="org.apache.ignite.spi.discovery.tcp.ipfinder.sharedfs.TcpDiscoverySharedFsIpFinder">
          <property name="path" value="/var/ignite/addresses"/>
```

```
            </bean>
          </property>
        </bean>
      </property>
    </bean>
```

4.1.7 Поисковик IP-адресов ZooKeeper

Примечание: Не поддерживается в .NET/C#.

Расширение РЕД КВАНТ ZooKeeper Ip Finder предоставляет средство поиска IP-адресов TCP Discovery, которое использует каталог ZooKeeper для поиска других узлов РЕД КВАНТ для подключения.

4.2 Обнаружение через ZooKeeper

Обнаружение TCP/IP РЕД КВАНТ по умолчанию организует узлы кластера в кольцевую топологию, которая имеет свои преимущества и недостатки. Например, в топологиях с сотнями узлов кластера для прохождения системного сообщения через все узлы может потребоваться много секунд. В результате базовая обработка событий, таких как присоединение новых узлов или обнаружение вышедших из строя узлов, может занять некоторое время, влияя на общую скорость отклика и производительность кластера.

ZooKeeper Discovery предназначен для массовых развертываний, которым необходимо сохранить простоту масштабирования и линейную производительность. Однако использование РЕД КВАНТ и ZooKeeper требует настройки и управления двумя распределенными системами, что может оказаться сложной задачей. Поэтому рекомендуется использовать ZooKeeper Discovery, только если планируется масштабирование до 100 или 1000 узлов. В противном случае лучше использовать обнаружение TCP/IP.

ZooKeeper Discovery использует ZooKeeper в качестве единой точки синхронизации и для организации кластера в звездообразную топологию, где кластер ZooKeeper находится в центре, а узлы РЕД КВАНТ обмениваются событиями обнаружения через него (рисунок 5).

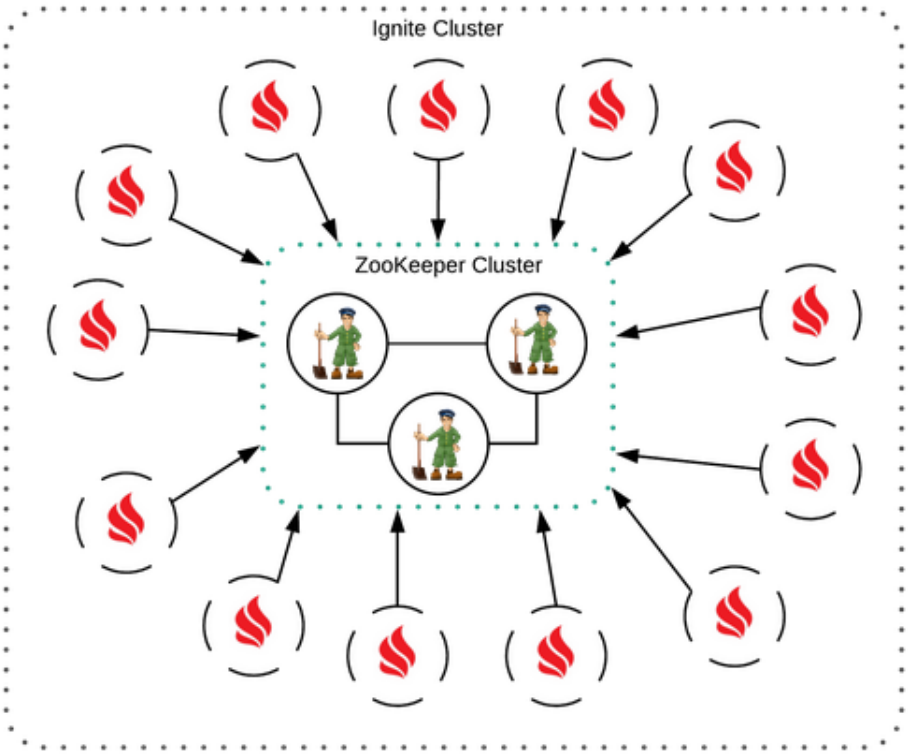

Рисунок 5 - ZooKeeper

Стоит отметить, что ZooKeeper Discovery является альтернативной реализацией Discovery SPI и не влияет на Communication SPI. Как только узлы обнаруживают друг друга с помощью ZooKeeper Discovery, они используют SPI для одноранговой связи.

Чтобы включить ZooKeeper Discovery, необходимо настроить ZookeeperDiscoverySpi следующим образом:

```
<bean class="org.apache.ignite.configuration.IgniteConfiguration">


 <property name="discoverySpi">
  <bean class="org.apache.ignite.spi.discovery.zk.ZookeeperDiscoverySpi">
   <property name="zkConnectionString" value="127.0.0.1:34076,127.0.0.1:43310,127.0.0.1:36745"/>
   <property name="sessionTimeout" value="30000"/>
   <property name="zkRootPath" value="/apacheIgnite"/>
   <property name="joinTimeout" value="10000"/>
  </bean>
 </property>
</bean>
```

Требуются следующие параметры (остальные параметры необязательны):

- zkConnectionString — хранит список адресов серверов ZooKeeper.
- sessionTimeout — указывает время, по истечении которого узел РЕД КВАНТ считается отключенным, если он не реагирует на события, которыми обмениваются через Discovery SPI.

4.2.1 Сбои и обработка Split-brain

В случае разделения сети некоторые из узлов не могут обмениваться данными друг с другом, поскольку они расположены в отдельных сегментах сети, что может привести к сбою обработки запросов пользователей или несогласованному изменению данных.

ZooKeeper Discovery подходит к разделению сети (так называемому «Split-brain») и сбоям связи между отдельными узлами следующим образом:

Внимание: Предполагается, что кластер ZooKeeper всегда виден всем узлам кластера. На самом деле, если узел отключается от ZooKeeper, он отключается, и другие узлы рассматривают его как неисправный или отключенный.

Всякий раз, когда узел обнаруживает, что он не может подключиться к некоторым другим узлам в кластере, он инициирует процесс устранения сбоя связи, публикуя специальные запросы в кластере ZooKeeper. При запуске процесса все узлы пытаются соединиться друг с другом и отправляют результаты попыток соединения узлу, который координирует процесс (узел-координатор). На основе этой информации узел-координатор создает граф связности, отражающий сетевую ситуацию в кластере. Дальнейшие действия зависят от типа сегментации сети. В следующих разделах обсуждаются возможные сценарии.

4.2.1.1 Кластер разбит на несколько непересекающихся компонентов

Если кластер разделен на несколько независимых компонентов, каждый компонент (являющийся кластером) может считать себя главным кластером и продолжать обрабатывать запросы пользователей, что приводит к несогласованности данных. Чтобы избежать этого, поддерживается только компонент с наибольшим количеством узлов; и узлы из других компонентов сбиваются.

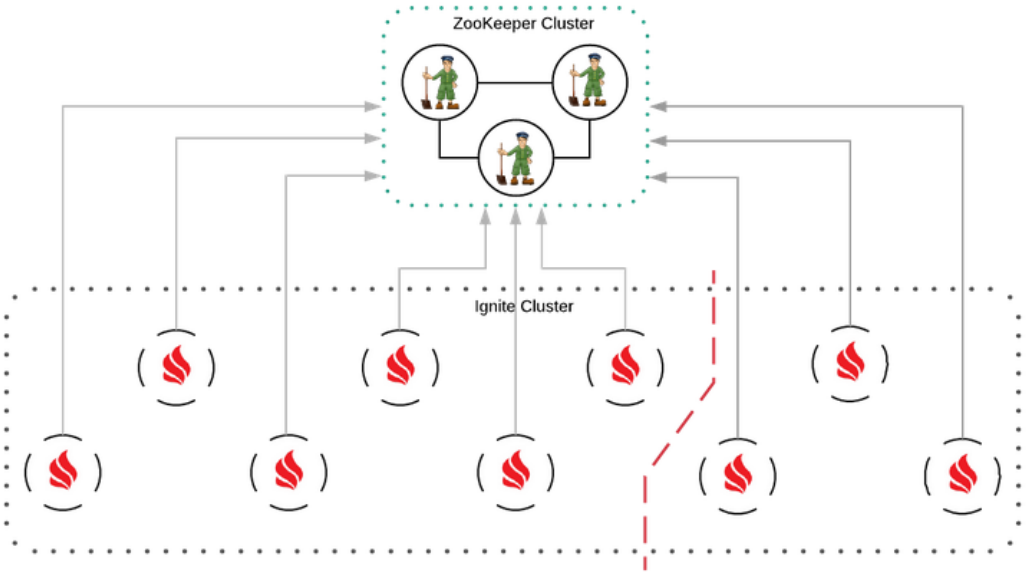

Рисунок 6 - Разделенный кластер

На рисунке 6 показан случай, когда сеть кластера разделена на 2 сегмента. Узлы из меньшего кластера (правый сегмент) отключаются (останавливаются) или завершают работу (рисунок 7).

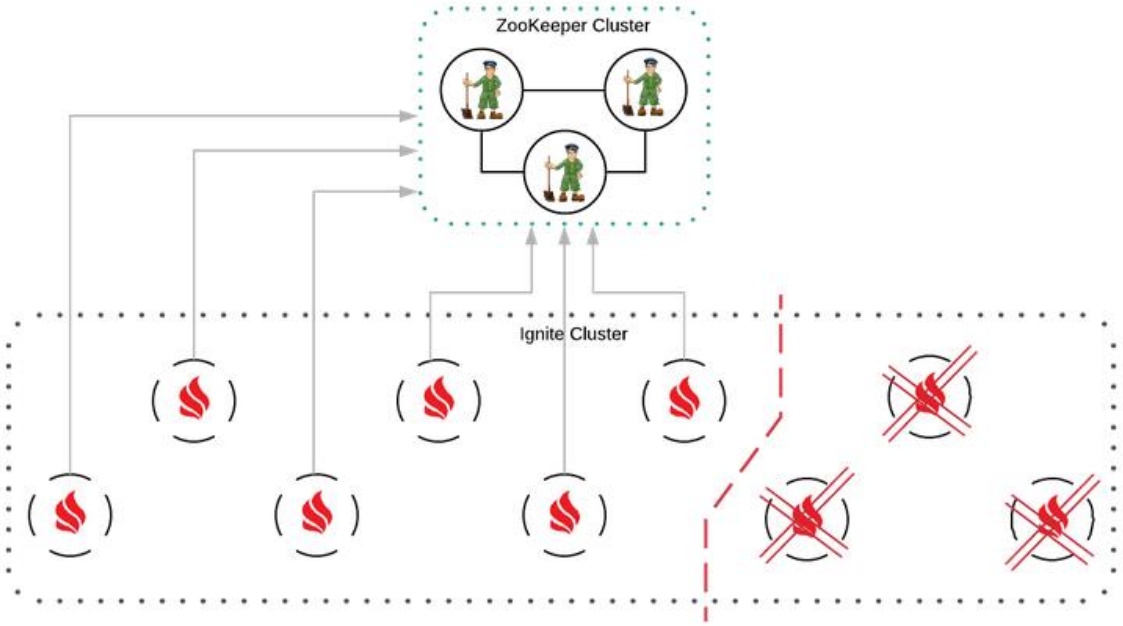

Рисунок 7 - Остановка или завершение работы узлов меньшего кластера

При наличии нескольких самых крупных компонентов активным остается тот, у которого больше всего клиентов, а остальные отключаются.

4.2.1.2 Отсутствуют несколько связей между узлами

Некоторые узлы не могут подключиться к некоторым другим узлам, что означает, что узлы не полностью отключены от кластера, но не могут обмениваться данными с некоторыми из узлов и, следовательно, не могут быть частью кластера. На рисунке 8 один узел не может соединиться с двумя другими узлами.

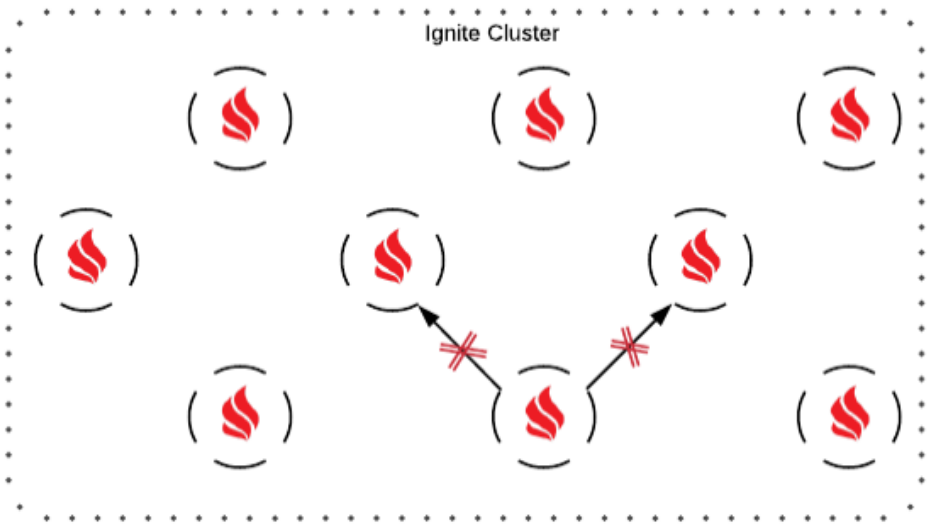

Рисунок 8 - Отсутствие связи с некоторыми узлами

В этом случае задача состоит в том, чтобы найти наибольший компонент, в котором каждый узел может соединиться с каждым другим узлом, что в общем случае является сложной задачей и не может быть решено за приемлемое время. Узел-координатор использует эвристический алгоритм для поиска наилучшего приближенного решения. Узлы, оставшиеся вне решения, закрываются (рисунок 9).

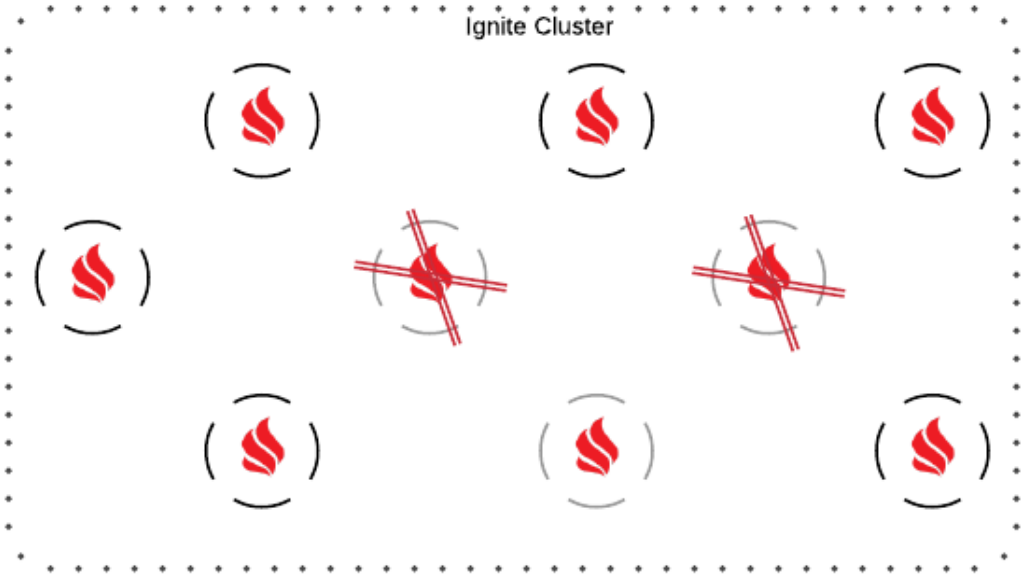

Рисунок 9 - Закрытие узлов вне решения

4.2.1.3 Сегментация кластера ZooKeeper

В крупномасштабных развертываниях, где кластер ZooKeeper может охватывать несколько центров обработки данных и географически разбросанных местоположений, он может быть разделен на несколько сегментов из-за сегментации сети. Если это происходит, ZooKeeper проверяет, существует ли сегмент, содержащий более половины всех узлов ZooKeeper (ZooKeeper требует именно столько узлов для продолжения своей работы), и, если он найден, этот сегмент берет на себя управление кластером РЕД КВАНТ, в то время как другие сегменты закрываются. Если такого сегмента нет, ZooKeeper отключает все свои узлы.

В случае сегментации кластера ZooKeeper кластер РЕД КВАНТ может быть разделен или не разделен. В любом случае, когда узлы ZooKeeper отключены, соответствующие узлы РЕД КВАНТ пытаются подключиться к доступным узлам ZooKeeper и отключаются, если не могут этого сделать.

На рисунке 10 показан пример сегментации сети, которая разделяет кластер РЕД КВАНТ и кластер ZooKeeper на два сегмента. Это может произойти, если кластеры развернуты в двух центрах обработки данных. В этом случае узел ZooKeeper, расположенный в центре обработки данных B, отключается. Узлы РЕД КВАНТ, расположенные в центре обработки данных B, не могут подключиться к остальным узлам ZooKeeper и также отключаются.

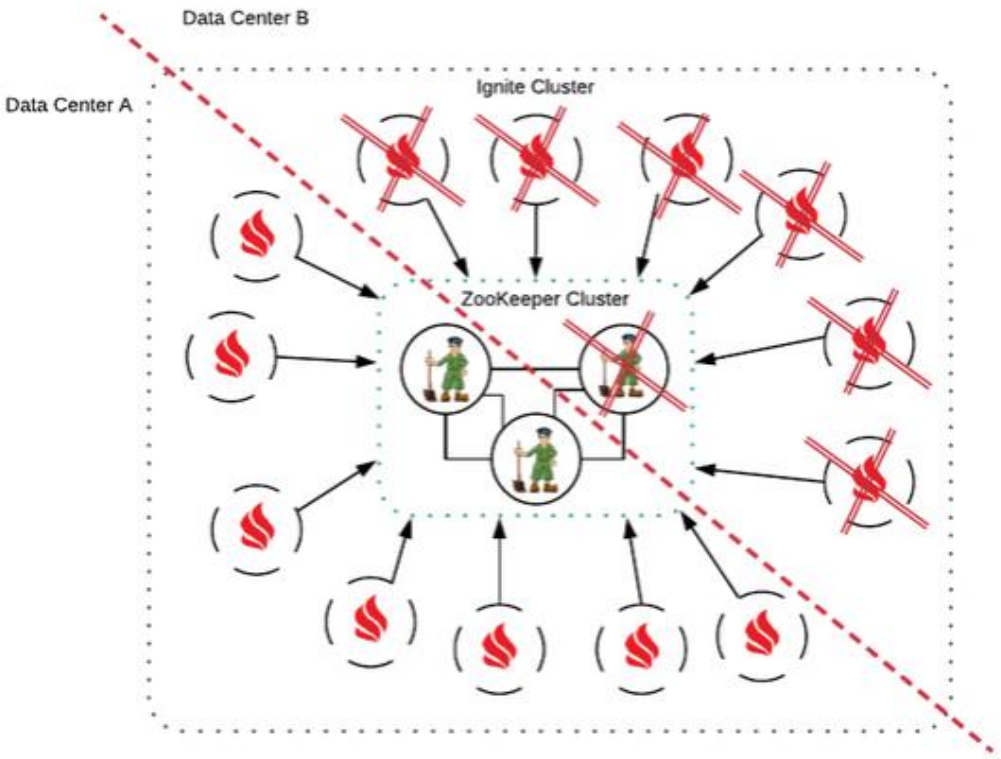

Рисунок 10 - Разделение кластера РЕД КВАНТ и кластера ZooKeeper

4.2.2 Пользовательские события обнаружения

Изменение кольцевой топологии на звездообразную влияет на способ обработки события обнаружения компонентом Discovery SPI. Поскольку кольцевая топология является линейной, это означает, что каждое сообщение обнаружения обрабатывается узлами последовательно.

С ZooKeeper Discovery координатор отправляет сообщения об обнаружении всем узлам одновременно, в результате чего сообщения обрабатываются параллельно. В результате ZooKeeper Discovery запрещает изменение пользовательских событий обнаружения. Например, узлам не разрешено добавлять какую-либо полезную нагрузку к сообщениям обнаружения.

4.2.3 Рекомендации по настройке РЕД КВАНТ и ZooKeeper

При использовании ZooKeeper Discovery необходимо убедиться, что параметры конфигурации кластера ZooKeeper и кластера РЕД КВАНТ совпадают.

Есть следующий пример конфигурации ZooKeeper:

```
# The number of milliseconds of each tick

tickTime=2000

# The number of ticks that can pass between sending a request and getting an acknowledgement
syncLimit=5
```

Настроенный таким образом сервер ZooKeeper обнаруживает свою собственную сегментацию от остальной части кластера ZooKeeper только по истечении tickTime *

syncLimit. Пока это событие не будет обнаружено на уровне ZooKeeper, все узлы РЕД КВАНТ, подключенные к сегментированному серверу ZooKeeper, не будут пытаться повторно подключиться к другим серверам ZooKeeper.

С другой стороны, у РЕД КВАНТ есть параметр sessionTimeout, который определяет, как скоро ZooKeeper закрывает сеанс узла РЕД КВАНТ, если узел отключается от кластера ZooKeeper. Если sessionTimeout меньше, чем tickTime * syncLimit , то узел РЕД КВАНТ получает уведомление от сегментированного сервера ZooKeeper слишком поздно — срок его сеанса истекает до того, как он попытается повторно подключиться к другим серверам ZooKeeper.

Чтобы избежать этой ситуации, sessionTimeout должен быть больше, чем tickTime * syncLimit.

4.3 СЕТЕВАЯ КОНФИГУРАЦИЯ

РЕД КВАНТ пытается поддерживать IPv4 и IPv6, но иногда это может приводить к проблемам, из-за которых кластер отсоединяется. Возможное решение — если не требуется IPv6 — ограничить РЕД КВАНТ до IPv4, установив параметр JVM -Djava.net.preferIPv4Stack=true.

4.3.1 ОБНАРУЖЕНИЕ

В этом пункте описываются сетевые параметры механизма обнаружения по умолчанию, который использует протокол TCP/IP для проверки сообщений обнаружения и реализован в классе TcpDiscoverySpi.

Изменить свойства механизма обнаружения можно следующим образом:

```
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:util="http://www.springframework.org/schema/util" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xsi:schemaLocation=" http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
http://www.springframework.org/schema/util http://www.springframework.org/schema/util/spring-util.xsd">

    <bean class="org.apache.ignite.configuration.IgniteConfiguration">

      <property name="discoverySpi">
        <bean class="org.apache.ignite.spi.discovery.tcp.TcpDiscoverySpi">
          <property name="localPort" value="8300"/>
        </bean>
      </property>

    </bean>
  </beans>
```

В таблице 4 описаны некоторые наиболее важные свойства TcpDiscoverySpi.

Таблица 4 - Свойства TcpDiscoverySpi

Свойство	Описание	Значение по умолчанию
localAddress	IP-адрес локального хоста, используемый для обнаружения. Если установлено, переопределяет параметр IgniteConfiguration.localHost.	По умолчанию узел привязывается ко всем доступным сетевым адресам. Если доступен адрес без обратной связи, используется java.net.InetAddress.getLocalHost().
localPort	Порт, к которому привязывается узел. Если установлено значение, отличное от значения по умолчанию, другие узлы кластера должны знать этот порт, чтобы иметь возможность обнаружить узел.	47500
localPortRange	Если localPort занят, узел пытается выполнить привязку к следующему порту (увеличенному на 1) и продолжает этот процесс, пока не найдет свободный порт. Свойство localPortRange определяет количество портов, которые узел будет пробовать (начиная с localPort).	100
soLinger	Задает тайм-аут задержки при закрытии сокетов TCP, используемых Discovery SPI. В РЕД КВАНТ тайм-аут по умолчанию имеет неотрицательное значение, чтобы предотвратить потенциальные взаимоблокировки с соединениями SSL, но, как побочный эффект, это может продлить обнаружение сбоев узлов кластера. В качестве альтернативы необходимо обновить версию JRE до версии, в которой исправлена проблема с SSL, и соответствующим образом изменить этот параметр.	0
reconnectCount	Количество раз, когда узел пытается (повторно) установить соединение с другим узлом.	10
networkTimeout	Максимальное время ожидания сети в миллисекундах для сетевых операций.	5000
socketTimeout	Тайм-аут операций сокета. Этот тайм-аут используется для ограничения времени подключения и записи в сокет.	5000
ackTimeout	Тайм-аут подтверждения для сообщений об обнаружении. Если подтверждение не получено в течение этого тайм-аута, SPI	5000

Свойство	Описание	Значение по умолчанию
	обнаружения пытается повторно отправить сообщение.	
joinTimeout	Тайм-аут соединения определяет, сколько времени узел ожидает присоединения к кластеру. Если используется не совместно IP-поисковик и узлу не удается подключиться к какому-либо адресу из IP-поиска, узел продолжает попытки подключиться в течение этого тайм-аута. Если все адреса не отвечают, генерируется исключение и узел завершает работу. 0 означает ожидание на неопределенный срок.	0
statisticsPrintFrequency	Определяет, как часто узел выводит статистику обнаружения в журнал. 0 означает отсутствие вывода. Если значение больше 0 и тихий режим отключен, то статистика выводится на информационном уровне один раз за каждый период.	0

Внимание: Необходимо инициализировать параметр IgniteConfiguration.localHost или TcpDiscoverySpi.localAddress с помощью сетевого интерфейса, который будет использоваться для связи между узлами. По умолчанию узел привязывается и прослушивает все доступные IP-адреса среды, в которой он работает. Это может продлить обнаружение сбоев узла, если некоторые адреса узла недоступны с других узлов кластера.

4.3.2 Коммуникация

После того, как узлы обнаруживают друг друга и формируется кластер, узлы обмениваются сообщениями через коммуникационный SPI. Сообщения представляют распределенные кластерные операции, такие как выполнение задач, операции по изменению данных, запросы и т. д. Реализация коммуникационного SPI по умолчанию использует протокол TCP/IP для обмена сообщениями (TcpCommunicationSpi). В этом пункте описываются свойства TcpCommunicationSpi.

Каждый узел открывает локальный коммуникационный порт и адрес, к которым подключаются другие узлы и отправляют сообщения. При запуске узел пытается выполнить привязку к указанному коммуникационному порту (по умолчанию 47100). Если порт уже используется, узел увеличивает номер порта, пока не найдет свободный порт. Количество попыток определяется свойством localPortRange (по умолчанию 100).

```
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:util="http://www.springframework.org/schema/util" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xsi:schemaLocation="
http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
http://www.springframework.org/schema/util
http://www.springframework.org/schema/util/spring-util.xsd">
```

```
  <bean class="org.apache.ignite.configuration.IgniteConfiguration">

    <property name="communicationSpi">
      <bean class="org.apache.ignite.spi.communication.tcp.TcpCommunicationSpi">
        <property name="localPort" value="4321"/>
      </bean>
    </property>

  </bean>
</beans>
```

В таблице 5 приведен список некоторых важных свойств TcpCommunicationSpi.

Таблица 5 - Свойства TcpCommunicationSpi

Свойство	**Описание**	**Значение по умолчанию**
localAddress	Локальный адрес для связи SPI, к которому необходимо подключиться. Если установлено, переопределяет параметр IgniteConfiguration.localHost.	
localPort	Локальный порт, который узел использует для связи.	47100
localPortRange	Диапазон портов, к которым узел пытается последовательно привязаться, пока не найдет свободный.	100
tcpNoDelay	Задает значение параметра сокета TCP_NODELAY. Каждый принятый или созданный сокет будет использовать предоставленное значение. Параметр должен быть установлен в значение true (по умолчанию), чтобы сократить время запроса/ответа при обмене данными по протоколу TCP. В большинстве случаев не рекомендуется изменять этот параметр.	true
idleConnectionTimeout	Максимальное время простоя соединения в режиме ожидания (в миллисекундах), после которого соединение закрывается.	600000
usePairedConnections	Следует ли принудительно устанавливать соединение с двумя сокетами между узлами. Если установлено значение true, между взаимодействующими узлами будут установлены два отдельных соединения: одно для исходящих сообщений и одно для входящих сообщений. Если установлено значение false, для обоих направлений будет использоваться одно TCP-соединение. Этот флаг полезен в некоторых операционных	false

Свойство	Описание	Значение по умолчанию
	системах, когда доставка сообщений занимает слишком много времени.	
directBuffer	Логический флаг, указывающий, следует ли выделять прямой буфер NIO вместо буфера выделения heap NIO. Хотя прямые буферы работают лучше, в некоторых случаях (особенно в Windows) они могут вызывать сбои JVM. Если это происходит в выбранной среде, необходимо установить для этого свойства значение false.	true
directSendBuffer	Использовать ли прямой буфер NIO вместо буфера выделения кучи NIO при отправке сообщений.	false
socketReceiveBuffer	Размер буфера получения для сокетов, созданных или принятых коммуникационным SPI. Если установлено значение 0, используется значение операционной системы по умолчанию.	0
socketSendBuffer	Размер буфера отправки для сокетов, созданных или принятых коммуникационным SPI. Если установлено значение 0, используется значение операционной системы по умолчанию.	0

4.3.3 Тайм-ауты подключения

Есть несколько свойств, которые определяют время ожидания соединения:

- IgniteConfiguration.failureDetectionTimeout - Тайм-аут для основных сетевых операций для серверных узлов. Значение по умолчанию 10000.
- IgniteConfiguration.clientFailureDetectionTimeout - Тайм-аут для основных сетевых операций для клиентских узлов. Значение по умолчанию 30000.

Можно установить время ожидания обнаружения сбоя в конфигурации узла, как показано в примере ниже. Значения по умолчанию позволяют SPI обнаружению надежно работать в большинстве локальных и контейнерных развертываний. Однако в стабильных сетях с малой задержкой можно установить параметр на ~200 миллисекунд, чтобы быстрее обнаруживать сбои и реагировать на них.

```
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:util="http://www.springframework.org/schema/util" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xsi:schemaLocation=" http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
http://www.springframework.org/schema/util http://www.springframework.org/schema/util/spring-util.xsd">
```

```
  <bean class="org.apache.ignite.configuration.IgniteConfiguration">

    <property name="failureDetectionTimeout" value="5000"/>

    <property name="clientFailureDetectionTimeout" value="10000"/>

  </bean>
</beans>
```

4.4 Подключение клиентских узлов

4.4.1 Повторное подключение клиентского узла

Клиентский узел может отключиться от кластера в нескольких случаях:

- Клиентский узел не может восстановить соединение с серверным узлом из-за проблем с сетью.
- Соединение с серверным узлом было прервано на некоторое время; клиентский узел может повторно установить соединение с кластером, но сервер уже удалил клиентский узел, так как не получал клиентских сообщений.
- Кластер может выкинуть медленных клиентов.

Когда клиент определяет, что он отключен от кластера, он присваивает себе новый идентификатор узла и пытается повторно подключиться к кластеру. Следует обратить внимание, что это имеет побочный эффект: свойство ID локального узла кластера изменяется в случае повторного подключения клиента. Это означает, что может быть затронута любая логика приложения, которая полагалась на идентификатор.

Переподключение клиента в конфигурации узла можно отключить:

```
 <bean class="org.apache.ignite.configuration.IgniteConfiguration">

   <property name="clientMode" value="true"/>

   <property name="discoverySpi">
     <bean class="org.apache.ignite.spi.discovery.tcp.TcpDiscoverySpi">
       <!-- prevent this client from reconnecting on connection loss -->
       <property name="clientReconnectDisabled" value="true"/>
       <property name="ipFinder">

         <bean class="org.apache.ignite.spi.discovery.tcp.ipfinder.vm.TcpDiscoveryVmIpFinder">
           <property name="addresses">
             <list>
               <value>127.0.0.1:47500..47509</value>
             </list>
           </property>
         </bean>
       </property>
     </bean>
   </property>
   <property name="communicationSpi">
     <bean class="org.apache.ignite.spi.communication.tcp.TcpCommunicationSpi">
```

```
            <property name="slowClientQueueLimit" value="1000"/>
        </bean>
    </property>
</bean>
```

Пока клиент находится в отключенном состоянии и выполняется попытка повторного подключения, РЕД КВАНТ API создает исключение IgniteClientDisconnectedException. Исключение содержит future, представляющее операцию повторного подключения. Можно использовать future, чтобы дождаться завершения операции.

```
IgniteCache cache = ignite.getOrCreateCache(new CacheConfiguration<>("myCache"));

try {
    cache.put(1, "value");
} catch (IgniteClientDisconnectedException e) {
    if (e.getCause() instanceof IgniteClientDisconnectedException) {
        IgniteClientDisconnectedException cause = (IgniteClientDisconnectedException) e.getCause();

        cause.reconnectFuture().get(); // Wait until the client is reconnected.
        // proceed
    }
}
```

4.4.2 События отключения/повторного подключения клиента

Есть два события обнаружения, которые запускаются на клиентском узле, когда он отключается от кластера или повторно подключается к нему:

- EVT_CLIENT_NODE_DISCONNECTED;
- EVT_CLIENT_NODE_RECONNECTED.

Можно прослушивать эти события и выполнять в ответ пользовательские действия.

4.4.3 Управление медленными клиентскими узлами

Во многих развертываниях клиентские узлы запускаются на более медленных машинах с меньшей пропускной способностью сети. В этих сценариях серверы могут генерировать нагрузку (например, уведомления о непрерывных запросах), с которой клиенты не могут справиться. Это может привести к увеличению очереди исходящих сообщений на серверах, что в конечном итоге может привести либо к нехватке памяти на сервере, либо к блокировке всего кластера.

Чтобы справиться с такими ситуациями, можно настроить максимальное количество исходящих сообщений для клиентских узлов. Если размер исходящей очереди превышает это значение, клиентский узел отключается от кластера.

В приведенном ниже примере показано, как настроить лимит очереди медленных клиентов.

```
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:util="http://www.springframework.org/schema/util" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xsi:schemaLocation=" http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
http://www.springframework.org/schema/util http://www.springframework.org/schema/util/spring-util.xsd">

    <bean class="org.apache.ignite.configuration.IgniteConfiguration">
      <property name="clientMode" value="true"/>

      <property name="communicationSpi">
        <bean class="org.apache.ignite.spi.communication.tcp.TcpCommunicationSpi">
          <property name="slowClientQueueLimit" value="1000"/>
        </bean>
      </property>
    </bean>
  </beans>
```

4.5 ТОПОЛОГИЯ BASELINE

Топология Baseline — это набор узлов, предназначенных для хранения данных. Концепция топологии Baseline была введена для того, чтобы дать возможность контролировать, когда необходим ребаланс данных в кластере. Например, если есть кластер из 3 узлов, в котором данные распределяются между узлами, и добавляется еще 2 узла, процесс ребаланса перераспределяет данные между всеми 5 узлами. Процесс повторной балансировки происходит при изменении топологии Baseline, что может произойти либо автоматически, либо вручную.

Топология Baseline включает только серверные узлы; клиентские узлы никогда не включаются, потому что они не хранят данные.

Назначение топологии Baseline состоит в том, чтобы:

- избежать ненужной передачи данных, когда серверный узел покидает кластер на короткий период времени, например, из-за случайных сетевых сбоев или планового обслуживания сервера;
- предоставить возможность контролировать, когда необходим ребаланс данных.

Топология Baseline изменяется автоматически, когда включена автоматическая настройка топологии Baseline. Это поведение по умолчанию для чистых кластеров в памяти. Для постоянных кластеров функция автоматической настройки базовой топологии должна быть включена вручную. По умолчанию она отключена, и придется вручную изменить топологию Baseline. Топологию Baseline можно изменить с помощью управляющего скрипта.

Внимание: Любая попытка создать кэш во время изменения топологии Baseline приводит к исключению.

4.5.1 БАЗОВАЯ ТОПОЛОГИЯ В КЛАСТЕРАХ С ЧИСТОЙ ПАМЯТЬЮ

В кластерах с чистой памятью поведение по умолчанию заключается в автоматической настройке топологии Baseline для набора всех серверных узлов при добавлении или удалении

серверных узлов из кластера. Данные также перебалансируются автоматически. Можно отключить функцию автоматической настройки Baseline и вручную управлять топологией Baseline.

4.5.2 Базовая топология в кластерах с персистентностью

Если в кластере есть хотя бы одна область данных, в которой включена персистентность, кластер будет неактивен при первом запуске. В неактивном состоянии все операции запрещены. Кластер должен быть активирован, прежде чем появится возможность создавать кэши и загружать данные. Активация кластера устанавливает текущий набор серверных узлов в качестве топологии Baseline. При перезапуске кластера он активируется автоматически, как только все узлы, зарегистрированные в топологии Baseline, присоединяются. Однако, если некоторые узлы не присоединяются после перезапуска, необходимо активировать кластер вручную.

Можно активировать кластер с помощью одного из следующих инструментов:

- скрипт управления;
- API-команда REST;
- Программно:

```
Ignite ignite = Ignition.start();

ignite.cluster().state(ClusterState.ACTIVE);
```

4.5.3 Автоматическая настройка топологии Baseline

Вместо того, чтобы изменять топологию Baseline вручную, можно позволить кластеру сделать это автоматически. Эта функция называется автоматической настройкой топологии Baseline. Когда она включена, кластер отслеживает состояние своих серверных узлов и автоматически устанавливает базовый уровень для текущей топологии, когда топология кластера стабильна в течение настраиваемого периода времени.

Вот что происходит при изменении набора узлов в кластере:

- Кластер ожидает в течение настраиваемого времени (по умолчанию 5 минут).
- Если в течение этого периода нет других изменений топологии, РЕД КВАНТ устанавливает топологию Baseline на текущий набор узлов.
- Если набор узлов изменяется в течение этого периода, тайм-аут обновляется.

Каждое изменение в наборе узлов сбрасывает тайм-аут для автонастройки. Когда тайм-аут истекает и текущий набор узлов отличается от базовой топологии (например, присутствуют новые узлы или осталось несколько старых узлов), РЕД КВАНТ изменяет базовую топологию на текущий набор узлов. Это также вызывает ребаланс данных.

Тайм-аут автонастройки позволяет избежать ребаланса данных, когда узел отключается на короткий период из-за временной проблемы с сетью или когда необходимо быстро

перезапустить узел. Можно установить более высокое значение тайм-аута, если ожидаются временные изменения в наборе узлов и не нужно менять базовую топологию.

Базовая топология настраивается автоматически, только если кластер находится в активном состоянии.

Чтобы включить автоматическую корректировку топологии Baseline, можно использовать сценарий управления или программный метод API, показанный ниже:

```
Ignite ignite = Ignition.start();

ignite.cluster().baselineAutoAdjustEnabled(true);

ignite.cluster().baselineAutoAdjustTimeout(30000);
```

Чтобы отключить автоматическую корректировку базовой топологии, нужно воспользоваться тем же методом с переданным значением false:

```
ignite.cluster().baselineAutoAdjustEnabled(false);
```

4.5.4 Мониторинг топологии Baseline

Можно использовать следующие инструменты для мониторинга и/или управления базовой топологией:

- Скрипт управления;
- Компоненты JMX.

5 КОНФИГУРАЦИЯ ПАМЯТИ

5.1 Конфигурация регионов памяти

РЕД КВАНТ использует концепцию областей данных для управления объемом оперативной памяти, доступной для кэша или группы кэшей. Регион данных — это логическая расширяемая область в оперативной памяти, в которой находятся кэшированные данные. Можно управлять начальным размером области и максимальным размером, который она может занимать. Помимо размера, области данных управляют настройками персистентности кэшей.

По умолчанию существует один регион данных, который может занимать до 20% оперативной памяти, доступной узлу, и все создаваемые кэши размещаются в этом регионе; но можно добавить столько регионов, сколько необходимо. Есть несколько причин, по которым может понадобиться несколько регионов:

- Регионы позволяют настроить объем оперативной памяти, доступной для кэша, или количество кэшей.
- Параметры персистентности настраиваются для каждого региона. Если нужны как кэши, хранящиеся только в памяти, так и кэши, хранящие свое содержимое на диске, то необходимо настроить два (или более) региона данных с разными параметрами персистентности: один для кэшей в памяти и один для постоянных кэшей.
- Некоторые параметры памяти, такие как политики вытеснения, настраиваются для каждого региона данных.

См. пункт Настройка региона данных по умолчанию, чтобы узнать, как изменить параметры региона данных по умолчанию или настроить несколько регионов данных.

5.1.1 Настройка региона данных по умолчанию

По умолчанию новый кэш добавляется в регион данных по умолчанию. Если необходимо изменить свойства региона данных по умолчанию, сделать это можно в конфигурации хранилища данных.

```
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:util="http://www.springframework.org/schema/util" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
http://www.springframework.org/schema/util http://www.springframework.org/schema/util/spring-util.xsd">
  <bean class="org.apache.ignite.configuration.IgniteConfiguration" id="ignite.cfg">
    <property name="dataStorageConfiguration">
      <bean class="org.apache.ignite.configuration.DataStorageConfiguration">
        <!--
        Default memory region that grows endlessly. Any cache will be bound to this memory region
        unless another region is set in the cache's configuration.
        -->
        <property name="defaultDataRegionConfiguration">
          <bean class="org.apache.ignite.configuration.DataRegionConfiguration">
            <property name="name" value="Default_Region"/>
            <!-- 100 MB memory region with disabled eviction. -->
```

```
                <property name="initialSize" value="#{100 * 1024 * 1024}"/>
              </bean>
            </property>
          </bean>
        </property>
        <!-- other properties -->
      </bean>
    </beans>
```

5.1.2 Добавление пользовательских регионов данных

В дополнение к региону данных по умолчанию можно добавить дополнительные регионы данных с пользовательскими настройками. В следующем примере настраивается регион данных, который может занимать до 40 МБ, и используется политика вытеснения Random-2-LRU. Следует обратить внимание, что ниже в конфигурации создается кэш, который находится в новом регионе данных.

```
    <beans xmlns="http://www.springframework.org/schema/beans"
xmlns:util="http://www.springframework.org/schema/util" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xsi:schemaLocation="  http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
http://www.springframework.org/schema/util http://www.springframework.org/schema/util/spring-util.xsd">
      <bean class="org.apache.ignite.configuration.IgniteConfiguration" id="ignite.cfg">
        <property name="dataStorageConfiguration">
          <bean class="org.apache.ignite.configuration.DataStorageConfiguration">
            <!--
            Default memory region that grows endlessly. Any cache will be bound to this memory region
            unless another region is set in the cache's configuration.
            -->
            <property name="defaultDataRegionConfiguration">
              <bean class="org.apache.ignite.configuration.DataRegionConfiguration">
                <property name="name" value="Default_Region"/>
                <!-- 100 MB memory region with disabled eviction. -->
                <property name="initialSize" value="#{100 * 1024 * 1024}"/>
              </bean>
            </property>
            <property name="dataRegionConfigurations">
              <list>
                <!--
                40MB memory region with eviction enabled.
                -->
                <bean class="org.apache.ignite.configuration.DataRegionConfiguration">
                  <property name="name" value="40MB_Region_Eviction"/>
                  <!-- Memory region of 20 MB initial size. -->
                  <property name="initialSize" value="#{20 * 1024 * 1024}"/>
                  <!-- Maximum size is 40 MB. -->
                  <property name="maxSize" value="#{40 * 1024 * 1024}"/>
                  <!-- Enabling eviction for this memory region. -->
                  <property name="pageEvictionMode" value="RANDOM_2_LRU"/>
                </bean>
              </list>
            </property>
          </bean>
        </property>
        <property name="cacheConfiguration">
```

```
        <list>
          <!-- Cache that is mapped to a specific data region. -->
          <bean class="org.apache.ignite.configuration.CacheConfiguration">

            <property name="name" value="SampleCache"/>
            <!--
            Assigning the cache to the `40MB_Region_Eviction` region.
            -->
            <property name="dataRegionName" value="40MB_Region_Eviction"/>
          </bean>
        </list>
      </property>
      <!-- other properties -->
    </bean>
  </beans>
```

5.1.3 Стратегия прогрева кэша

РЕД КВАНТ не требует прогрева памяти с диска при перезагрузке. Как только кластер взаимосвязан, приложение может выполнять запросы и вычисления на нем. В то же время функция прогрева памяти предназначена для приложений с малой задержкой, которые предпочитают, чтобы данные загружались в память до того, как к ним можно будет выполнить запрос.

В настоящее время стратегия прогрева РЕД КВАНТ подразумевает загрузку данных во все или определенные регионы данных РЕД КВАНТ, начиная с индексов, до тех пор, пока не закончится свободное место. Его можно настроить как для всех регионов (по умолчанию), так и для каждого региона отдельно.

Чтобы разогреть все регионы данных, нужно передать параметр конфигурации LoadAllWarmUpStrategy в DataStorageConfiguration#setDefaultWarmUpConfiguration следующим образом:

```
  <bean class="org.apache.ignite.configuration.IgniteConfiguration">

    <property name="dataStorageConfiguration">
      <bean class="org.apache.ignite.configuration.DataStorageConfiguration">
        <property name="defaultWarmUpConfiguration">
          <bean class="org.apache.ignite.configuration.LoadAllWarmUpConfiguration"/>
        </property>
      </bean>
    </property>
  </bean>
```

Чтобы прогреть определенный регион данных, нужно передать параметр конфигурации LoadAllWarmUpStrategy в DataStorageConfiguration#setWarmUpConfiguration следующим образом:

```
  <bean class="org.apache.ignite.configuration.IgniteConfiguration">

  <property name="dataStorageConfiguration">
    <property name="dataRegionConfigurations">
```

```
        <bean class="org.apache.ignite.configuration.DataRegionConfiguration">
          <property name="name" value="NewDataRegion"/>
          <property name="initialSize" value="#{100 * 1024 * 1024}"/>
          <property name="persistenceEnabled" value="true"/>
          <property name="warmUpConfiguration">
            <bean class="org.apache.ignite.configuration.LoadAllWarmUpConfiguration"/>
          </property>
        </bean>
      </property>
    </property>
    </bean>
```

Чтобы остановить прогрев для всех регионов данных, нужно передать параметр конфигурации NoOpWarmUpConfiguration в DataStorageConfiguration#setDefaultWarmUpConfiguration следующим образом:

```
    <bean class="org.apache.ignite.configuration.IgniteConfiguration">

    <property name="dataStorageConfiguration">
      <bean class="org.apache.ignite.configuration.DataStorageConfiguration">
        <property name="defaultWarmUpConfiguration">
          <bean class="org.apache.ignite.configuration.NoOpWarmUpConfiguration"/>
        </property>
      </bean>
    </property>
    </bean>
```

Чтобы остановить прогрев для определенного региона данных, нужно передать параметр конфигурации NoOpWarmUpStrategy в DataStorageConfiguration#setWarmUpConfiguration следующим образом:

```
    <bean class="org.apache.ignite.configuration.IgniteConfiguration">

    <property name="dataStorageConfiguration">
      <property name="dataRegionConfigurations">
        <bean class="org.apache.ignite.configuration.DataRegionConfiguration">
          <property name="name" value="NewDataRegion"/>
          <property name="initialSize" value="#{100 * 1024 * 1024}"/>
          <property name="persistenceEnabled" value="true"/>
          <property name="warmUpConfiguration">
            <bean class="org.apache.ignite.configuration.NoOpWarmUpConfiguration"/>
          </property>
        </bean>
      </property>
    </property>
    </bean>
```

Также можно остановить прогрев кэша с помощью control.sh и JMX.

Чтобы остановить прогрев с помощью control.sh:

```
    control.sh --warm-up --stop --yes
```

Чтобы остановить прогрев с помощью JMX, необходимо воспользоваться методом:

```
org.apache.ignite.mxbean.WarmUpMXBean#stopWarmUp
```

5.2 Политики удаления данных

Когда встроенная персистентность отключена, РЕД КВАНТ хранит все записи кэша в памяти off-heap и выделяет страницы по мере поступления новых данных. Когда достигается предел памяти и РЕД КВАНТ не может выделить страницу, некоторые данные необходимо удалить из памяти, чтобы избежать ошибок OutOfMemory. Этот процесс называется вытеснением. Вытеснение предотвращает нехватку памяти в системе, но за счет потери данных и необходимости перезагружать их, когда они снова понадобятся.

Вытеснение применяется в следующих случаях:

- для памяти off-heap, когда встроенная персистентность отключена;
- для памяти off-heap, когда РЕД КВАНТ используется с внешним хранилищем;
- для кэшей в heap;
- для ближних кэшей, если они настроены.

Когда собственное сохранение включено, аналогичный процесс, называемый заменой страницы, используется для освобождения памяти вне кучи, когда РЕД КВАНТ не может выделить новую страницу. Разница в том, что данные не теряются (поскольку они хранятся в постоянном хранилище), но снижается скорость доступа к ним.

5.2.1 Освобождение памяти off-heap

Освобождение памяти off-heap реализовано следующим образом.

Когда использование памяти превышает заданный предел, РЕД КВАНТ применяет один из предварительно настроенных алгоритмов для выбора страницы памяти, наиболее подходящей для вытеснения. Затем каждая запись кэша, содержащаяся на странице, удаляется со страницы. Однако, если запись заблокирована транзакцией, она сохраняется (рисунок 11). Таким образом, либо вся страница, либо большая ее часть очищается и готова к повторному использованию.

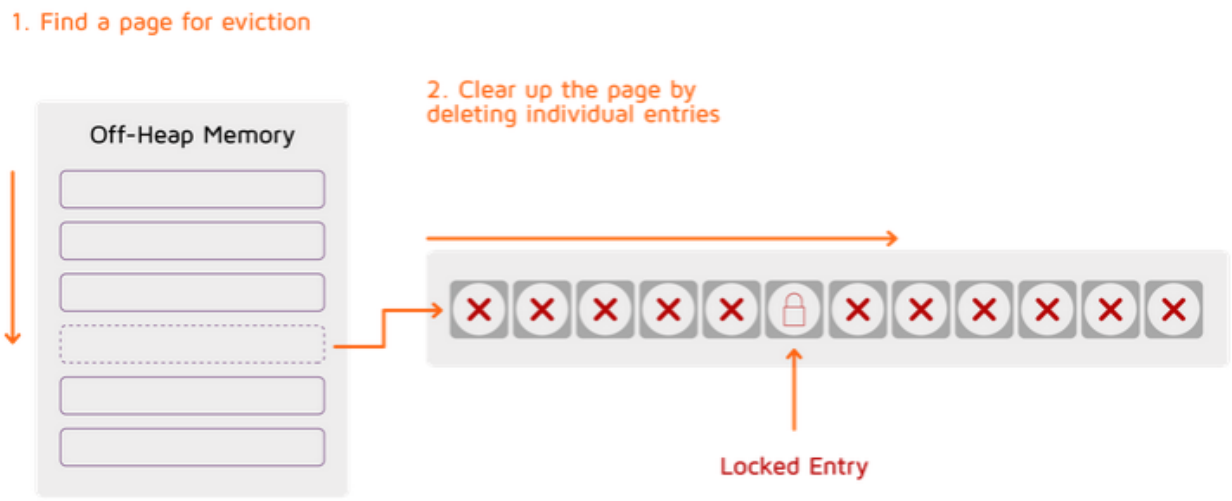

Рисунок 11 - Освобождение памяти

По умолчанию освобождение памяти off-heap отключено, что означает, что используемая память постоянно растет, пока не достигнет своего предела. Чтобы включить вытеснение, необходимо указать режим вытеснения страницы в конфигурации региона данных. Следует обратить внимание, что вытеснение памяти off-heap настраивается для каждого региона данных. Если регионы данных не используются, необходимо явно добавить параметры региона данных по умолчанию в конфигурацию, чтобы иметь возможность настроить вытеснение.

По умолчанию вытеснение начинается, когда общее потребление оперативной памяти регионом достигает 90%. Необходимо использовать параметр DataRegionConfiguration.setEvictionThreshold(…), если нужно инициировать вытеснение раньше или позже.

РЕД КВАНТ поддерживает два алгоритма выбора страницы:

- Random-LRU;
- Random-2-LRU.

Чтобы включить алгоритм вытеснения Random-LRU, нужно настроить регион данных, как показано ниже:

```
<bean class="org.apache.ignite.configuration.IgniteConfiguration">
  <!-- Memory configuration. -->
  <property name="dataStorageConfiguration">
    <bean class="org.apache.ignite.configuration.DataStorageConfiguration">
      <property name="dataRegionConfigurations">
        <list>
          <!-- Defining a data region that will consume up to 20 GB of RAM. -->
          <bean class="org.apache.ignite.configuration.DataRegionConfiguration">
            <!-- Custom region name. -->
            <property name="name" value="20GB_Region"/>
            <!-- 500 MB initial size (RAM). -->
            <property name="initialSize" value="#{500L * 1024 * 1024}"/>
            <!-- 20 GB maximum size (RAM). -->
            <property name="maxSize" value="#{20L * 1024 * 1024 * 1024}"/>
            <!-- Enabling RANDOM_LRU eviction for this region.  -->
```

```
                <property name="pageEvictionMode" value="RANDOM_LRU"/>
              </bean>
            </list>
          </property>
        </bean>
      </property>
    </bean>

    <bean class="org.apache.ignite.configuration.IgniteConfiguration">
      <!-- Memory configuration. -->
      <property name="dataStorageConfiguration">
        <bean class="org.apache.ignite.configuration.DataStorageConfiguration">
          <property name="dataRegionConfigurations">
            <list>
              <!--
                 Defining a data region that consumes up to 20 GB of RAM.
              -->
              <bean class="org.apache.ignite.configuration.DataRegionConfiguration">
                <!-- Custom region name. -->
                <property name="name" value="20GB_Region"/>

                <!-- 500 MB initial size (RAM). -->
                <property name="initialSize" value="#{500L * 1024 * 1024}"/>

                <!-- 20 GB maximum size (RAM). -->
                <property name="maxSize" value="#{20L * 1024 * 1024 * 1024}"/>

                <!-- Enabling RANDOM_LRU eviction for this region.  -->
                <property name="pageEvictionMode" value="RANDOM_LRU"/>
              </bean>
            </list>
          </property>
        </bean>
      </property>
      <!-- The rest of the configuration. -->
    </bean>
```

Алгоритм Random-LRU работает следующим образом:

- После настройки региона памяти, определенного политикой памяти, выделяется массив off-heap для отслеживания метки времени «последнего использования» для каждой отдельной страницы данных.
- При доступе к странице данных ее временная метка обновляется в массиве отслеживания.
- Когда приходит время удалить страницу, алгоритм случайным образом выбирает 5 индексов из массива отслеживания и удаляет страницу с самой старой меткой времени. Если некоторые из индексов указывают на страницы, не содержащие данных (индексные или системные страницы), то алгоритм выбирает другую страницу.

Чтобы включить алгоритм вытеснения Random-2-LRU, который является устойчивой к сканированию версией Random-LRU, необходимо настроить регион данных, как показано в примере ниже:

```
<bean class="org.apache.ignite.configuration.IgniteConfiguration">
  <!-- Memory configuration. -->
  <property name="dataStorageConfiguration">
    <bean class="org.apache.ignite.configuration.DataStorageConfiguration">
      <property name="dataRegionConfigurations">
        <list>
          <!--
             Defining a data region that consumes up to 20 GB of RAM.
          -->
          <bean class="org.apache.ignite.configuration.DataRegionConfiguration">
            <!-- Custom region name. -->
            <property name="name" value="20GB_Region"/>

            <!-- 500 MB initial size (RAM). -->
            <property name="initialSize" value="#{500L * 1024 * 1024}"/>

            <!-- 20 GB maximum size (RAM). -->
            <property name="maxSize" value="#{20L * 1024 * 1024 * 1024}"/>

            <!-- Enabling RANDOM_2_LRU eviction for this region.  -->
            <property name="pageEvictionMode" value="RANDOM_2_LRU"/>
          </bean>
        </list>
      </property>
    </bean>
  </property>
  <!-- The rest of the configuration. -->
</bean>
```

В Random-2-LRU для каждой страницы данных сохраняются две самые последние метки времени доступа. Во время удаления алгоритм случайным образом выбирает 5 индексов из массива отслеживания, и минимум между двумя последними временными метками берется для дальнейшего сравнения с соответствующими минимумами четырех других страниц, выбранных в качестве кандидатов на удаление.

Random-LRU-2 превосходит LRU, решая проблему «одного чуда»: если к странице данных обращаются редко, но случайно один раз, она защищена от вытеснения в течение длительного времени.

5.3 Политики вытеснения данных

Когда включена встроенная персистентность и объем данных, которые РЕД КВАНТ хранит на диске, больше, чем объем памяти off-heap, выделенный для региона данных, другая страница должна быть вытеснена из heap на диск, чтобы предварительно загрузить страницу с диска в полностью заполненную память off-heap. Этот процесс называется ротацией страниц или заменой страниц.

Когда встроенная персистентность отключена, вместо замены страницы используется удаление. Дополнительную информацию см. пункт 53 Политики удаления данных.

Замена страницы реализована следующим образом: когда РЕД КВАНТ запрашивает страницу, он пытается найти эту страницу в памяти off-heap. Если страница в данный момент не находится в памяти off-heap (возникает ошибка страницы), эта страница предварительно

загружается с диска. В то же время, когда память off-heap уже заполнена, следует выбрать другую страницу для замены (сохранить на диск и вытеснить).

РЕД КВАНТ поддерживает три алгоритма поиска страниц для замены:

- алгоритм Random-LRU;
- алгоритм Segmented-LRU;
- алгоритм CLOCK.

Алгоритм замены страницы можно настроить с помощью свойства PageReplacementMode DataRegionConfiguration. По умолчанию используется алгоритм CLOCK.

```
<bean class="org.apache.ignite.configuration.IgniteConfiguration">
  <!-- Memory configuration. -->
  <property name="dataStorageConfiguration">
    <bean class="org.apache.ignite.configuration.DataStorageConfiguration">
      <property name="dataRegionConfigurations">
        <list>
          <!--
            Defining a persistent data region with Segmented LRU page replacement mode.
          -->
          <bean class="org.apache.ignite.configuration.DataRegionConfiguration">
            <!-- Data region name. -->
            <property name="name" value="persistent_data_region"/>

            <!-- Enable persistence. -->
            <property name="persistenceEnabled" value="true"/>

            <!-- 20 GB maximum size (RAM). -->
            <property name="maxSize" value="#{20L * 1024 * 1024 * 1024}"/>

            <!-- Enabling SEGMENTED_LRU page replacement for this region.  -->
            <property name="pageReplacementMode" value="SEGMENTED_LRU"/>
          </bean>
        </list>
      </property>
    </bean>
  </property>
  <!-- The rest of the configuration. -->
</bean>
```

Выбор алгоритма зависит от рабочей нагрузки. В большинстве случаев CLOCK (по умолчанию) является хорошим выбором, но при некоторых рабочих нагрузках другие алгоритмы могут работать лучше.

В алгоритме Random-LRU каждом доступе к странице ее временная метка обновляется. Когда происходит ошибка страницы и требуется заменить некоторые страницы, алгоритм случайным образом выбирает 5 страниц из страничной памяти и удаляет страницу с самой последней меткой времени.

Этот алгоритм не требует затрат на обслуживание, но он не очень эффективен с точки зрения поиска следующей страницы для замены. Рекомендуется использовать его в средах, где

замена страниц не требуется (при работе с достаточно большим регионом данных для хранения всего объема данных) или происходит очень редко.

Алгоритм Segmented-LRU — это устойчивый к сканированию вариант алгоритма наименее недавно используемого (LRU). Список страниц Segmented-LRU разделен на два сегмента: пробный сегмент и защищенный сегмент. Страницы в каждом сегменте упорядочены от наименее посещаемых до самых последних. Новые страницы добавляются в самый последний доступный конец (хвост) пробного сегмента. Существующие страницы удаляются из того места, где они находятся в данный момент, и добавляются к последнему доступному концу защищенного сегмента. Таким образом, страницы в защищенном сегменте были доступны как минимум дважды. Защищенный сегмент конечен, поэтому миграция страницы из пробного сегмента в защищенный сегмент может привести к переносу страницы LRU из защищенного сегмента в последний использовавшийся конец пробного сегмента. Это дает странице еще один шанс получить доступ, прежде чем ее заменят. Замещаемая страница опрашивается с конца (головы) пробного сегмента, к которому последний раз обращались.

Этот алгоритм требует дополнительной памяти для хранения списка страниц, который также необходимо обновлять при каждом доступе к странице. В то же время алгоритм имеет почти оптимальную страницу для замены политики выбора. Таким образом, для сред без замены страниц может быть небольшое падение производительности (по сравнению с Random-LRU и CLOCK), но для сред с высокой скоростью замены страниц и большим количеством однократных сканирований Segmented-LRU может превзойти Random-LRU и CLOCK.

Алгоритм CLOCK хранит в памяти циклический список страниц, где «стрелка» указывает на последний просмотренный фрейм страницы в списке. Когда происходит ошибка страницы и нет пустых фреймов, флаг попадания страницы проверяется в месте расположения «стрелки». Если флаг попадания равен 0, новая страница помещается на место страницы, на которую указывает «стрелка», и стрелка перемещается на одну позицию дальше. В противном случае флаг попадания очищается, затем увеличивается стрелка часов, и процесс повторяется до тех пор, пока страница не будет заменена.

Этот алгоритм имеет почти нулевую стоимость обслуживания и эффективность политики замены между Random-LRU и Segmented-LRU.

6 КОНФИГУРАЦИЯ СЛОЯ ХРАНЕНИЯ

6.1 НАСТРОЙКА ХРАНЕНИЯ ДАННЫХ В РЕД КВАНТ

Персистентность РЕД КВАНТ или native persistence — это набор функций, предназначенных для обеспечения постоянного хранения. Когда он включен, РЕД КВАНТ всегда сохраняет все данные на диске и загружает как можно больше данных в оперативную память для обработки. Например, если имеется 100 записей, а объем оперативной памяти может хранить только 20, то все 100 сохраняются на диске и только 20 кэшируются в оперативную память для повышения производительности.

Когда native persistence отключено и внешнее хранилище не используется, РЕД КВАНТ ведет себя как чистое хранилище в памяти.

Когда персистентность включена, каждый узел сервера сохраняет подмножество данных, которое включает только разделы, назначенные этому узлу (включая резервные разделы, если включено резервное копирование).

Функциональность native persistence основана на следующих функциях:

- хранение разделов данных на диске;
- ведение журнала с предварительной записью;
- checkpointing;
- изменение сбора данных;
- использование подкачки ОС.

Когда включена персистентность, РЕД КВАНТ сохраняет каждый раздел в отдельном файле на диске (рисунок 12). Формат данных файлов разделов такой же, как у данных, хранящихся в памяти. Если резервные копии разделов включены, они также сохраняются на диск. Помимо разделов данных, РЕД КВАНТ хранит индексы и метаданные.

Расположение файлов данных по умолчанию в конфигурации можно изменить.

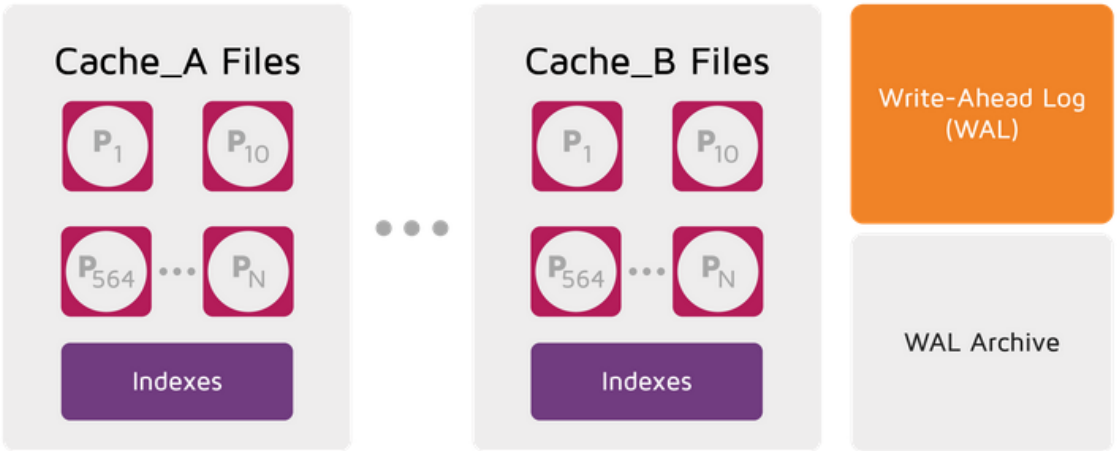

Рисунок 12 - Хранение данных на диске

6.1.1 Включение постоянного хранилища

Встроенная персистентность настраивается для каждого региона данных. Чтобы включить постоянное хранилище, нужно задать для свойства persistenceEnabled значение true в конфигурации региона данных. Можно одновременно иметь регионы данных в памяти и регионы данных с сохранением.

В следующем примере показано, как включить постоянное хранилище для области данных по умолчанию.

```
<bean class="org.apache.ignite.configuration.IgniteConfiguration">

  <property name="dataStorageConfiguration">
    <bean class="org.apache.ignite.configuration.DataStorageConfiguration">
      <property name="defaultDataRegionConfiguration">
        <bean class="org.apache.ignite.configuration.DataRegionConfiguration">
          <property name="persistenceEnabled" value="true"/>
        </bean>
      </property>
    </bean>
  </property>
</bean>
```

6.1.2 Настройка каталога постоянного хранилища

Когда персистентность включена, узел сохраняет пользовательские данные, индексы и файлы WAL в каталоге {IGNITE_WORK_DIR}/db. Этот каталог называется каталогом хранения. Можно изменить каталог хранилища, установив свойство storagePath объекта DataStorageConfiguration, как показано ниже.

Каждый узел поддерживает следующие подкаталоги в каталоге хранилища, предназначенные для хранения данных кэша, файлов WAL и архивных файлов WAL:

- {WORK_DIR}/db/{nodeId} — каталог содержит данные кэша и индексы;
- {WORK_DIR}/db/wal/{nodeId} — каталог содержит файлы WAL;
- {WORK_DIR}/db/wal/archive/{nodeId} — каталог содержит архивные файлы WAL.

nodeId здесь — либо согласованный идентификатор узла (если он определен в конфигурации узла), либо автоматически сгенерированный идентификатор узла. Он используется для обеспечения уникальности каталогов для узла. Если несколько узлов совместно используют один и тот же рабочий каталог, они используют разные подкаталоги.

Если рабочий каталог содержит файлы персистентности для нескольких узлов (имеется несколько подкаталогов {nodeId} с разными идентификаторами узлов), узел выбирает первый неиспользуемый подкаталог. Чтобы убедиться, что узел всегда использует определенный подкаталог и, следовательно, определенные разделы данных даже после перезапуска, нужно задать для IgniteConfiguration.setConsistentId уникальное значение для всего кластера в конфигурации узла.

Изменить каталог хранения можно следующим образом:

```
<bean class="org.apache.ignite.configuration.IgniteConfiguration">

  <property name="dataStorageConfiguration">
    <bean class="org.apache.ignite.configuration.DataStorageConfiguration">
      <property name="defaultDataRegionConfiguration">
        <bean class="org.apache.ignite.configuration.DataRegionConfiguration">
          <property name="persistenceEnabled" value="true"/>
        </bean>
      </property>
      <property name="storagePath" value="/opt/storage"/>
    </bean>
  </property>
</bean>
```

Также можно изменить пути к WAL и архивам WAL, чтобы они указывали на каталоги за пределами каталога хранилища.

6.1.3 Журнал предварительной записи

Журнал предварительной записи (WAL) — это журнал всех операций по изменению данных (включая удаления), которые происходят на узле. Когда страница обновляется в оперативной памяти, обновление не записывается напрямую в файл раздела, а добавляется в конец WAL.

Цель журнала предварительной записи — предоставить механизм восстановления для сценариев, когда один узел или весь кластер выходит из строя. В случае сбоя или перезапуска кластер всегда можно восстановить до последней успешно зафиксированной транзакции, полагаясь на содержимое WAL.

WAL состоит из нескольких файлов (называемых активными сегментами) и архива. Активные сегменты заполняются последовательно и перезаписываются в циклическом порядке. Как только заполняется 1-й сегмент, его содержимое копируется в архив WAL (см. пункт 62 Архив WAL). Пока 1-й сегмент копируется, 2-й сегмент обрабатывается как активный файл WAL и принимает все обновления, поступающие со стороны приложения. По умолчанию активных сегментов 10.

6.1.3.1 WAL-режимы

Существует несколько режимов WAL. Каждый режим отличается тем, как он влияет на производительность и обеспечивает различные гарантии согласованности. Перечень режимов представлен в таблице 6.

Таблица 6 - Режимы WAL

Режим	Описание	Гарантии согласованности
FSYNC	Изменения гарантированно сохраняются на диск при каждой атомарной записи или коммита транзакции.	Обновления данных никогда не теряются при сбоях ОС или процессов, а также при отключении питания.

Режим	Описание	Гарантии согласованности
LOG_ONLY	Режим по умолчанию. Изменения гарантированно будут сброшены либо в буферный кэш ОС, либо в отображаемый в память файл при каждой атомарной записи или коммита транзакции. Файл с отображением памяти используется по умолчанию, и его можно отключить, установив для системного свойства IGNITE_WAL_MMAP значение false.	Обновления данных сохраняются после сбоя процесса.
BACKGROUND	Когда свойство IGNITE_WAL_MMAP включено (по умолчанию), этот режим ведет себя как режим LOG_ONLY. Если подход с отображенным в память файлом отключен, то изменения остаются во внутреннем буфере узла и периодически сбрасываются на диск. Частота сброса задается параметром walFlushFrequency.	Когда свойство IGNITE_WAL_MMAP включено (по умолчанию), режим обеспечивает те же гарантии, что и режим LOG_ONLY. В противном случае последние обновления данных могут быть потеряны в случае сбоя процесса или других сбоев.
NONE	WAL отключен. Изменения сохраняются только в том случае, если узел будет корректно отключен. Чтобы деактивировать кластер и закрыть узел нужно воспользоваться Ignite.active(false).	Возможна потеря данных. Если узел внезапно завершает работу во время операций обновления, весьма вероятно, что данные, хранящиеся на диске, рассинхронизируются или будут повреждены.

6.1.3.2 Архив WAL

Архив WAL используется для хранения сегментов WAL, которые могут понадобиться для восстановления узла после сбоя. Количество сегментов, хранящихся в архиве, такое, чтобы общий размер всех сегментов не превышал заданный размер архива WAL.

По умолчанию максимальный размер архива WAL (общее пространство, которое он занимает на диске) определяется как 4-кратный размер буфера checkpointing. Можно изменить это значение в конфигурации.

Внимание: Установка размера для архива WAL меньшего значения, чем значение по умолчанию, может повлиять на производительность и должна быть протестирована перед использованием в рабочей среде.

6.1.3.3 Изменение размера сегмента WAL

Размер сегмента WAL по умолчанию (64 МБ) может быть неэффективным в сценариях с высокой нагрузкой, поскольку он приводит к слишком частому переключению WAL между

сегментами, а переключение/ротация является дорогостоящей операцией. Больший размер сегментов WAL может помочь повысить производительность при высоких нагрузках за счет увеличения общего размера файлов WAL и архива WAL.

Можно изменить размер файлов сегментов WAL в конфигурации хранилища данных. Значение должно быть от 512 КБ до 2 ГБ.

```
<bean class="org.apache.ignite.configuration.IgniteConfiguration" id="ignite.cfg">

  <property name="dataStorageConfiguration">
    <bean class="org.apache.ignite.configuration.DataStorageConfiguration">

      <!-- set the size of wal segments to 128MB -->
      <property name="walSegmentSize" value="#{128 * 1024 * 1024}"/>

      <property name="defaultDataRegionConfiguration">
        <bean class="org.apache.ignite.configuration.DataRegionConfiguration">
          <property name="persistenceEnabled" value="true"/>
        </bean>
      </property>

    </bean>
  </property>
</bean>
```

6.1.3.4 Отключение WAL

Внимание! Отключение или включение WAL следует выполнять только в стабильной топологии: должны присутствовать все базовые узлы, ни один узел не должен присоединяться к кластеру или покидать его на время этой операции. В противном случае кэш может зависнуть в несогласованном состоянии. Если это произойдет, рекомендуется уничтожить затронутые кэши.

Бывают ситуации, когда разумно отключить WAL для повышения производительности. Например, полезно отключить WAL во время первоначальной загрузки данных и включить его после завершения предварительной загрузки.

```
IgniteConfiguration cfg = new IgniteConfiguration();

DataStorageConfiguration storageCfg = new DataStorageConfiguration();
storageCfg.getDefaultDataRegionConfiguration().setPersistenceEnabled(true);

cfg.setDataStorageConfiguration(storageCfg);

Ignite ignite = Ignition.start(cfg);

ignite.cluster().state(ClusterState.ACTIVE);

String cacheName = "myCache";

ignite.getOrCreateCache(cacheName);

ignite.cluster().disableWal(cacheName);
```

```
//load data
ignite.cluster().enableWal(cacheName);
```

Внимание! Если WAL отключен и выполняется перезапуск узла, все данные удалятся из постоянного хранилища на этом узле. Это реализовано потому, что без WAL не может быть гарантирована согласованность данных в случае сбоя или перезапуска узла.

6.1.3.5 Сжатие архива WAL

Существует возможность включения сжатия архива WAL, чтобы уменьшить пространство, занимаемое архивом WAL. По умолчанию архив WAL содержит сегменты для последних 20 checkpointing (это число можно настроить). Если сжатие включено, все заархивированные сегменты старше 1 контрольной точки сжимаются в формате ZIP. Если сегменты необходимы (например, для ребаланса данных между узлами), они не сжимаются в формат RAW.

См. пункт Свойства конфигурации, чтобы узнать, как включить сжатие архива WAL.

6.1.3.6 Сжатие записей WAL

Физические и логические записи, представляющие собой обновления данных, записываются в файлы WAL до подтверждения операции пользователя. РЕД КВАНТ может сжимать записи WAL в памяти перед их записью на диск для экономии места.

Для сжатия записей WAL требуется, чтобы был включен модуль 'ignite-compress'. См. Включение модулей.

По умолчанию сжатие записей WAL отключено. Чтобы включить его, необходимо задать алгоритм сжатия и уровень сжатия в конфигурации хранилища данных:

```
IgniteConfiguration cfg = new IgniteConfiguration();

DataStorageConfiguration dsCfg = new DataStorageConfiguration();
dsCfg.getDefaultDataRegionConfiguration().setPersistenceEnabled(true);

//WAL page compression parameters
dsCfg.setWalPageCompression(DiskPageCompression.LZ4);
dsCfg.setWalPageCompressionLevel(8);

cfg.setDataStorageConfiguration(dsCfg);
Ignite ignite = Ignition.start(cfg);
```

6.1.3.7 Отключение архива WAL

В некоторых случаях может потребоваться отключить архивирование WAL, например, для уменьшения накладных расходов, связанных с копированием сегментов WAL в архив. Возможна ситуация, когда РЕД КВАНТ записывает данные в сегменты WAL быстрее, чем сегменты копируются в архив. Это может создать узкое место ввода-вывода, которое может заморозить работу узла. Если возникли такие проблемы, нужно попробовать отключить архивирование WAL.

Чтобы отключить архивирование, необходимо установить для пути WAL и пути архива WAL одно и то же значение. В этом случае РЕД КВАНТ не копирует сегменты в архив; вместо этого он создает новые сегменты в папке WAL. Старые сегменты удаляются по мере роста WAL в соответствии с настройкой размера архива WAL.

6.1.4 CHECKPOINTING

Checkpointing — это процесс копирования грязных страниц из оперативной памяти в файлы разделов на диске. Грязная страница — это страница, которая была обновлена в оперативной памяти, но не записана в файл соответствующего раздела (обновление, однако, было добавлено в WAL).

После создания checkpointing все изменения сохраняются на диске и будут доступны в случае сбоя и перезапуска узла.

Checkpointing и ведение журнала с предварительной записью предназначены для обеспечения сохранности данных и восстановления в случае сбоя узла.

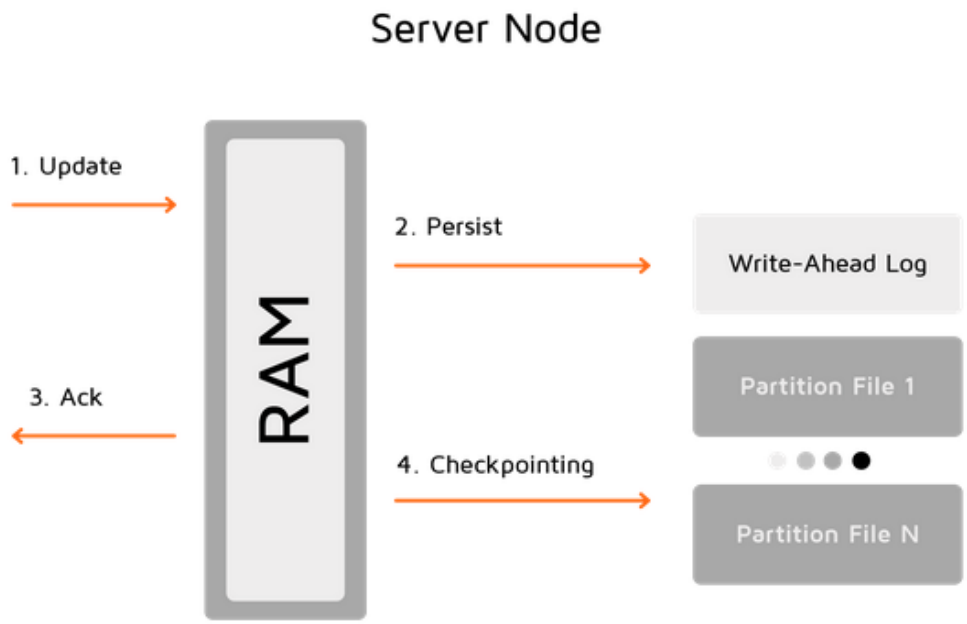

Рисунок 13 - Процесс сохранения данных

Этот процесс помогает экономно использовать дисковое пространство, сохраняя страницы на диске в самом актуальном состоянии. После прохождения checkpointing можно удалить сегменты WAL, которые были созданы до этого момента времени.

6.1.5 СВОЙСТВА КОНФИГУРАЦИИ

В таблице представлены некоторые свойства DataStorageConfiguration.

Таблица 7 - Свойства DataStorageConfiguration

Наименование свойства	Описание	Значение по умолчанию
persistenceEnabled	Необходимо установить для этого свойства значение true, чтобы включить встроенное сохранение.	false
storagePath	Путь, по которому хранятся данные.	${IGNITE_HOME}/work/db/node{IDX}-{UUID}
walPath	Путь к каталогу, в котором хранятся активные сегменты WAL.	${IGNITE_HOME}/work/db/wal/
walArchivePath	Путь к архиву WAL.	${IGNITE_HOME}/work/db/wal/archive/
walCompactionEnabled	Необходимо установить значение true, чтобы включить сжатие архива WAL.	false
walSegmentSize	Размер файла сегмента WAL в байтах.	64МБ
walMode	WAL-режимы	LOG_ONLY
walCompactionLevel	Уровень сжатия архива WAL. 1 указывает на самую высокую скорость, а 9 — на лучшее сжатие.	1
maxWalArchiveSize	Максимальный размер (в байтах), который архив WAL может занимать в файловой системе.	В четыре раза больше размера буфера checkpointing.

6.2 ВНЕШНЕЕ ХРАНИЛИЩЕ

РЕД КВАНТ можно использовать в качестве уровня кэширования поверх существующей базы данных, такой как RDBMS или базы данных NoSQL, например, Apache Cassandra или MongoDB. Этот вариант использования ускоряет работу основной базы данных за счет обработки в памяти.

РЕД КВАНТ обеспечивает готовую интеграцию с Apache Cassandra. Для других баз данных NoSQL, для которых интеграция недоступна в готовом виде, можно предоставить собственную реализацию интерфейса CacheStore.

Два основных варианта использования внешнего хранилища включают в себя:

- Уровень кэширования существующей базы данных. В этом сценарии можно повысить скорость обработки, загрузив данные в память. Также можно внедрить поддержку SQL в базу данных, в которой ее нет (когда все данные загружены в память).
- Хранение данных во внешней базе данных (вместо использования native Persistence).

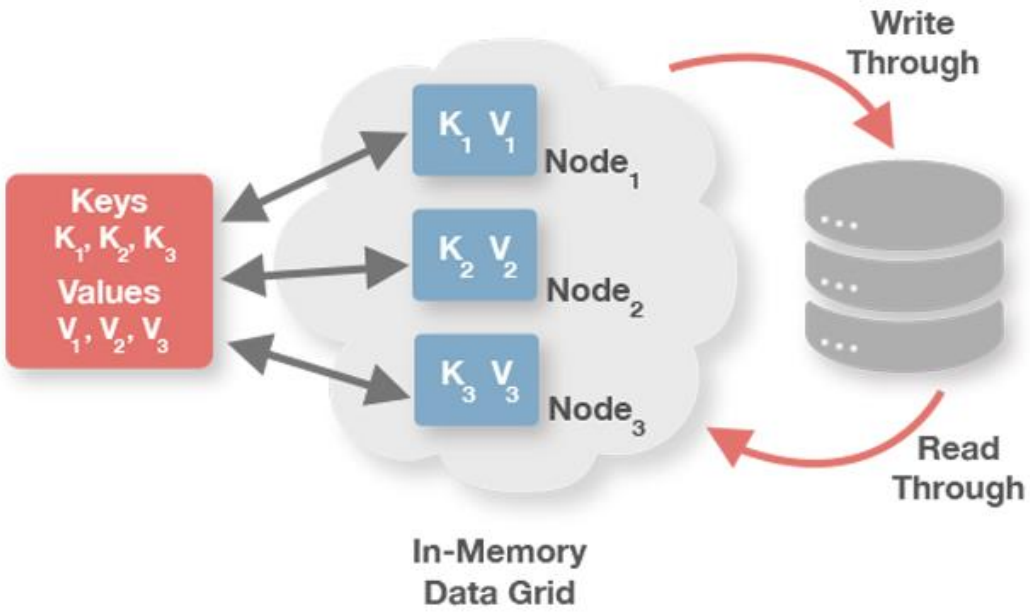

Рисунок 14 - Внешнее хранилище

Интерфейс CacheStore расширяет как javax.cache.integration.CacheLoader, так и javax.cache.integration.CacheWriter, которые используются для функций чтения и записи соответственно. Также можно реализовать каждый из интерфейсов по отдельности и предоставить их конфигурации кэша отдельно.

Примечание: Помимо операций «ключ-значение», РЕД КВАНТ записывает результаты запросов SQL INSERT, UPDATE и MERGE. Однако запросы SELECT никогда не считывают данные из внешней базы данных.

6.2.1 Сквозное чтение и запись

Сквозное чтение означает, что данные считываются из базового постоянного хранилища, если они недоступны в кэше. Следует отметить, что это верно только для операций получения, выполняемых через API «ключ-значение»; запросы SELECT никогда не считывают данные из внешней базы данных. Для выполнения запросов на выборку данные должны быть предварительно загружены из базы данных в кэш с помощью вызова метода loadCache().

Сквозная запись означает, что данные автоматически сохраняются при их обновлении в кэше. Все операции чтения и записи участвуют в транзакциях кэша и фиксируются или откатываются как единое целое.

6.2.2 Кэширование с отложенной записью

В простом режиме сквозной записи каждая операция размещения и удаления включает соответствующий запрос к постоянному хранилищу; поэтому общая продолжительность операции обновления может быть относительно большой. Кроме того, интенсивная частота обновления кэша может вызвать чрезвычайно высокую нагрузку на хранилище.

Для таких случаев можно включить режим отложенной записи, при котором операции обновления выполняются асинхронно. Ключевой концепцией этого подхода является накопление обновлений и их асинхронная передача в основную базу данных в виде массовой

операции. Можно инициировать сброс данных на основе событий, зависящих от времени (максимальное время, в течение которого запись данных может находиться в очереди, ограничено), событий размера очереди (очередь сбрасывается, когда ее размер достигает определенной точки) или их обоих (в зависимости от того, что произойдет раньше).

Внимание! Включение кэширования с отложенной записью повышает производительность за счет выполнения асинхронных обновлений, но это может привести к потенциальному снижению согласованности, поскольку некоторые обновления могут быть потеряны из-за сбоев или сбоев узлов.

При использовании отложенной записи в базовое хранилище записывается только последнее обновление записи. Если запись кэша с ключом с именем key1 последовательно обновляется значениями value1, value2 и value3 соответственно, то в постоянное хранилище распространяется только один запрос на сохранение для пары (key1, value3).

Примечание: Пакетные операции обычно более эффективны, чем последовательность отдельных операций. Можно использовать эту функцию, включив пакетные операции в режиме отложенной записи. Последовательности обновления похожих типов (установить или удалить) можно сгруппировать в один пакет. Например, если необходимо поместить пары (key1, value1), (key2, value2), (key3, value3) в кэш последовательно, три операции объединятся в одну операцию CacheStore.putAll(…).

6.2.3 Интеграция RDBMS

Чтобы использовать RDBMS в качестве базового хранилища, можно использовать одну из следующих реализаций CacheStore:

- CacheJdbcPojoStore — сохраняет объекты в виде набора полей с использованием отражения. Рекомендуется использовать эту реализацию, если РЕД КВАНТ добавляется поверх существующей базы данных и нужно использовать определенные поля (или все из них) из базовой таблицы.
- CacheJdbcBlobStore — хранит объекты в основной базе данных в формате BLOB-объектов. Этот параметр полезен в сценариях, когда используется внешняя база данных в качестве постоянного хранилища и необходимо хранить данные в простом формате.

Ниже приведены примеры конфигурации для обеих реализаций CacheStore.

6.2.3.1 CacheJdbcPojoStore

С помощью CacheJdbcPojoStore можно хранить объекты в виде набора полей и настраивать сопоставление между столбцами таблицы и полями объектов с помощью конфигурации.

1. Необходимо задать для свойства CacheConfiguration.cacheStoreFactory значение org.apache.ignite.cache.store.jdbc.CacheJdbcPojoStoreFactory и указать следующие свойства:
    - dataSourceBean — учетные данные для подключения к базе данных: URL, пользователь, пароль;

- dialect — класс, реализующий диалект SQL, совместимый с базой данных. РЕД КВАНТ предоставляет готовые реализации для баз данных MySQL, Oracle, H2, SQLServer и DB2. Эти диалекты можно найти в пакете org.apache.ignite.cache.store.jdbc.dialect дистрибутива РЕД КВАНТ.
- types — это свойство требуется для определения сопоставлений между таблицей базы данных и соответствующим POJO (см. пример конфигурации POJO ниже).

2. При необходимости нужно настроить объекты запросов, если требуется выполнять запросы SQL в кэше.

В следующем примере показано, как настроить кэш РЕД КВАНТ поверх таблицы MySQL. В таблице есть 2 столбца: id (INTEGER) и name (VARCHAR), которые сопоставляются с объектами класса Person.

CacheJdbcPojoStore можно настроить как с помощью конфигурации XML, так и с помощью кода Java:

```
IgniteConfiguration igniteCfg = new IgniteConfiguration();

CacheConfiguration<Integer, Person> personCacheCfg = new CacheConfiguration<>();

personCacheCfg.setName("PersonCache");
personCacheCfg.setCacheMode(CacheMode.PARTITIONED);
personCacheCfg.setAtomicityMode(CacheAtomicityMode.ATOMIC);

personCacheCfg.setReadThrough(true);
personCacheCfg.setWriteThrough(true);

CacheJdbcPojoStoreFactory<Integer, Person> factory = new CacheJdbcPojoStoreFactory<>();
factory.setDialect(new MySQLDialect());
factory.setDataSourceFactory((Factory<DataSource>)() -> {
   MysqlDataSource mysqlDataSrc = new MysqlDataSource();
   mysqlDataSrc.setURL("jdbc:mysql://[host]:[port]/[database]");
   mysqlDataSrc.setUser("YOUR_USER_NAME");
   mysqlDataSrc.setPassword("YOUR_PASSWORD");
   return mysqlDataSrc;
});

JdbcType personType = new JdbcType();
personType.setCacheName("PersonCache");
personType.setKeyType(Integer.class);
personType.setValueType(Person.class);
// Specify the schema if applicable
// personType.setDatabaseSchema("MY_DB_SCHEMA");
personType.setDatabaseTable("PERSON");

personType.setKeyFields(new JdbcTypeField(java.sql.Types.INTEGER, "id", Integer.class, "id"));

personType.setValueFields(new JdbcTypeField(java.sql.Types.INTEGER, "id", Integer.class, "id"));
personType.setValueFields(new JdbcTypeField(java.sql.Types.VARCHAR, "name", String.class, "name"));

factory.setTypes(personType);

personCacheCfg.setCacheStoreFactory(factory);
```

```
QueryEntity qryEntity = new QueryEntity();

qryEntity.setKeyType(Integer.class.getName());
qryEntity.setValueType(Person.class.getName());
qryEntity.setKeyFieldName("id");

Set<String> keyFields = new HashSet<>();
keyFields.add("id");
qryEntity.setKeyFields(keyFields);

LinkedHashMap<String, String> fields = new LinkedHashMap<>();
fields.put("id", "java.lang.Integer");
fields.put("name", "java.lang.String");

qryEntity.setFields(fields);

personCacheCfg.setQueryEntities(Collections.singletonList(qryEntity));

igniteCfg.setCacheConfiguration(personCacheCfg);
```

Класс Person:

```
class Person implements Serializable {

    private static final long serialVersionUID = 0L;

    private int id;

    private String name;

    public Person() {
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }
}
```

6.2.3.2 CacheJdbcBlobStore

CacheJdbcBlobStore хранит объекты в основной базе данных в формате BLOB-объектов. Он создает таблицу с именем «ENTRIES» со столбцами «akey» и «val» (оба имеют двоичный тип).

Можно изменить определение таблицы по умолчанию, предоставив пользовательский запрос на создание таблицы и запросы DML, используемые для загрузки, удаления и обновления данных.

В приведенном ниже примере объекты класса Person хранятся в виде массива байтов в одном столбце.

```
IgniteConfiguration igniteCfg = new IgniteConfiguration();

CacheConfiguration<Integer, Person> personCacheCfg = new CacheConfiguration<>();
personCacheCfg.setName("PersonCache");

CacheJdbcBlobStoreFactory<Integer, Person> cacheStoreFactory = new CacheJdbcBlobStoreFactory<>();

cacheStoreFactory.setUser("USER_NAME");

MysqlDataSource mysqlDataSrc = new MysqlDataSource();
mysqlDataSrc.setURL("jdbc:mysql://[host]:[port]/[database]");
mysqlDataSrc.setUser("USER_NAME");
mysqlDataSrc.setPassword("PASSWORD");

cacheStoreFactory.setDataSource(mysqlDataSrc);

personCacheCfg.setCacheStoreFactory(cacheStoreFactory);

personCacheCfg.setWriteThrough(true);
personCacheCfg.setReadThrough(true);

igniteCfg.setCacheConfiguration(personCacheCfg);
```

6.2.4 Загрузка данных

После настройки хранилища кэша и запуска кластера, необходимо загрузить данные из базы данных в кластер следующим образом:

```
// Load data from person table into PersonCache.
IgniteCache<Integer, Person> personCache = ignite.cache("PersonCache");

personCache.loadCache(null);
```

6.2.5 Интеграция с базой данных NoSQL

РЕД КВАНТ можно интегрировать с любой базой данных NoSQL, внедрив интерфейс CacheStore.

Внимание: Несмотря на то, что РЕД КВАНТ поддерживает распределенные транзакции, он не делает базу данных NoSQL транзакционной, если только база данных не поддерживает транзакции из коробки.

6.2.5.1 Интеграция Cassandra

РЕД КВАНТ предоставляет готовую реализацию CacheStore, которая позволяет использовать Apache Cassandra в качестве постоянного хранилища. Эта реализация использует асинхронные запросы Cassandra для обеспечения высокопроизводительных пакетных операций, таких как loadAll(), writeAll() и deleteAll(), и автоматически создает все необходимые таблицы и пространства имен в Cassandra.

6.3 Конфигурация снимков

6.3.1 Настройка конфигурации снимков

По умолчанию сегмент моментального снимка хранится в рабочем каталоге соответствующего узла РЕД КВАНТ. Этот сегмент использует тот же носитель, на котором РЕД КВАНТ Persistence хранит данные, индекс, WAL и другие файлы. Поскольку моментальный снимок может занимать столько же места, сколько уже занято файлами персистентности, и может влиять на производительность приложений, разделяя дисковый ввод-вывод с подпрограммами РЕД КВАНТ Persistence, рекомендуется хранить моментальные снимки и файлы персистентности на разных носителях.

Избежать этого взаимодействия между сохранением РЕД КВАНТ Native и моментальными снимками можно, изменив каталоги хранения файлов персистентности, либо переопределив расположение моментальных снимков по умолчанию, как показано ниже:

```
<bean class="org.apache.ignite.configuration.IgniteConfiguration">
  <!--
    Sets a path to the root directory where snapshot files will be persisted.
    By default, the `snapshots` directory is placed under the `IGNITE_HOME/db`.
  -->
  <property name="snapshotPath" value="/snapshots"/>

  <property name="cacheConfiguration">
    <bean class="org.apache.ignite.configuration.CacheConfiguration">
      <property name="name" value="snapshot-cache"/>
    </bean>
  </property>

</bean>
```

6.4 Сжатие диска

Сжатие диска относится к процессу сжатия страниц данных при их записи на диск, что уменьшает размер дискового хранилища. Страницы хранятся в памяти в несжатом виде, но когда данные сбрасываются на диск, они сжимаются с использованием настроенного алгоритма. Это относится только к страницам данных, которые хранятся в постоянном

хранилище и не сжимают индексы или записи WAL. Сжатие записей WAL можно включить отдельно.

Сжатие страниц диска можно включить отдельно для каждого кэша в конфигурации кэша. Кэш должен находиться в постоянной области данных. На данный момент нет возможности включить глобальное сжатие страниц диска. Кроме того, должны быть соблюдены следующие обязательные условия:

- Свойство pageSize в конфигурации хранилища данных должно быть установлено как минимум в 2 раза больше размера страницы файловой системы. Это означает, что размер страницы должен быть либо 8K, либо 16K.
- Должен быть включен модуль ignite-compress.

Подробности о тонкой настройке хранения данных см. в пункте Настройка производительности слоя хранения.

Чтобы включить сжатие страниц диска для кэша, необходимо указать один из доступных алгоритмов сжатия в конфигурации кэша, как показано в следующем примере:

```
<bean class="org.apache.ignite.configuration.IgniteConfiguration">
  <property name="dataStorageConfiguration">
    <bean class="org.apache.ignite.configuration.DataStorageConfiguration">
      <property name="pageSize" value="#{4096 * 2}"/>
      <property name="defaultDataRegionConfiguration">
        <bean class="org.apache.ignite.configuration.DataRegionConfiguration">
          <property name="persistenceEnabled" value="true"/>
        </bean>
      </property>
    </bean>
  </property>
  <property name="cacheConfiguration">
    <bean class="org.apache.ignite.configuration.CacheConfiguration">
      <property name="name" value="myCache"/>
      <!-- enable disk page compression for this cache -->
      <property name="diskPageCompression" value="LZ4"/>
      <!-- optionally set the compression level -->
      <property name="diskPageCompressionLevel" value="10"/>
    </bean>
  </property>
</bean>
```

6.4.1 Поддерживаемые алгоритмы

Поддерживаемые алгоритмы сжатия включают в себя:

- ZSTD — поддерживает уровни сжатия от -131072 до 22 (по умолчанию: 3);
- LZ4 — поддерживает уровни сжатия от 0 до 17 (по умолчанию: 0);
- SNAPPY — алгоритм Snappy;
- SKIP_GARBAGE — этот алгоритм извлекает полезные данные только из наполовину заполненных страниц и не сжимает данные.

6.5 CHANGE DATA CAPTURE (CDC)

Сбор измененных данных (CDC) — это шаблон обработки данных, используемый для асинхронного получения записей, которые были изменены на локальном узле, чтобы можно было выполнить действие с использованием измененной записи.

Ниже приведены некоторые варианты использования CDC:

- Потоковые изменения в Warehouse;
- Обновление поискового индекса;
- Подсчет статистики (потоковые запросы);
- Журналы аудита;
- Асинхронное взаимодействие с внешней системой: модерация, вызов бизнес-процессов и т. д.

РЕД КВАНТ реализует CDC с помощью приложения ignite-cdc.sh и Java API.

На рисунке 15 представлены приложение CDC и узел РЕД КВАНТ, интегрированные через сегменты архива WAL.

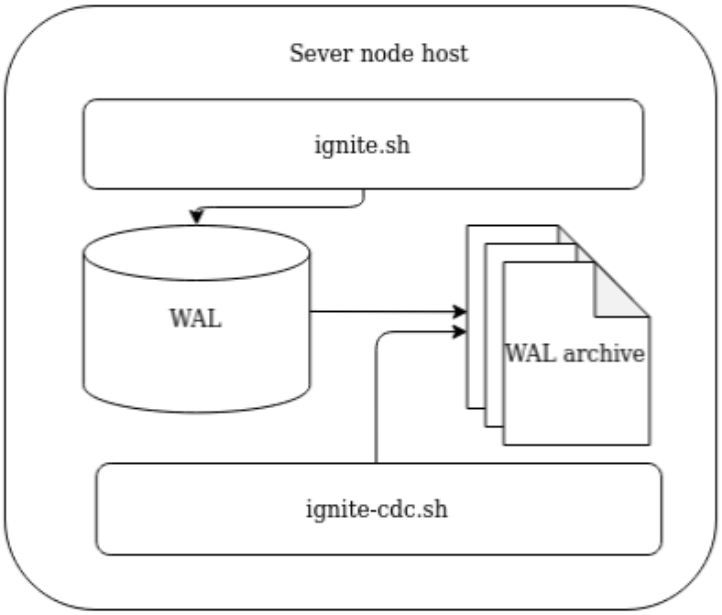

Рисунок 15 - Приложение CDC и узел РЕД КВАНТ

Когда CDC включен, серверный узел РЕД КВАНТ создает жесткую ссылку на каждый сегмент архива WAL в специальном каталоге db/cdc/{consistency_id}. Приложение ignite-cdc.sh работает на другой JVM и обрабатывает только что заархивированные сегменты WAL. Когда сегмент полностью обработан ignite-cdc.sh, он удаляется. Фактическое место на диске освобождается, когда удаляются обе ссылки (архив и CDC).

Состояние потребления — это указатель на последнее обработанное событие. Потребитель может сообщить ignite-cdc.sh, чтобы сохранить состояние потребления. При запуске обработка событий будет продолжена с последнего сохраненного состояния.

6.5.1 Конфигурация

6.5.1.1 Узел РЕД КВАНТ

Параметры узла РЕД КВАНТ представлены в таблице 8.

Таблица 8 - Параметры узла РЕД КВАНТ

Наименование	Описание	Значение по умолчанию
DataStorageConfiguration#cdcEnabled	Флаг для включения CDC на узле сервера.	false
DataStorageConfiguration#cdcWalPath	Путь к каталогу CDC.	«db/wal/cdc»
DataStorageConfiguration#walForceArchiveTimeout	Тайм-аут для принудительного архивирования сегмента WAL, даже если он не завершен.	-1 (отключено)

6.5.1.2 Приложение CDC

CDC настраивается так же, как узел РЕД КВАНТ — через XML-файл spring:

- ignite-cdc.sh требует для запуска конфигурации РЕД КВАНТ и CDC;
- IgniteConfiguration используется для определения общих параметров, таких как путь к каталогу CDC, согласованный идентификатор узла и другие параметры;
- CdcConfiguration содержит параметры, специфичные для ignite-cdc.sh.

Параметры CDC представлены в таблице 9.

Таблица 9 - Параметры CDC

Наименование	Описание	Значение по умолчанию
lockTimeout	Тайм-аут ожидания получения блокировки. CDC блокирует каталог при запуске, чтобы гарантировать отсутствие параллельного ignite-cdc.sh, обрабатывающего один и тот же каталог.	1000 миллисекунд.
checkFrequency	Промежуток времени, в течение которого приложение переходит в спящий режим между последующими проверками, когда новые файлы недоступны.	1000 миллисекунд.
keepBinary	Флаг, указывающий, должны ли ключ и значение измененных записей предоставляться в двоичном формате.	true
consumer	Реализация org.apache.ignite.cdc.CdcConsumer, использующая изменения записей.	null
metricExporterSpi	Массив SPI для экспорта метрик CDC.	null

6.5.2 API

6.5.2.1 org.apache.ignite.cdc.CdcEvent

В таблице 10 приведено одно изменение данных, отраженное CdcEvent.

Таблица 10 - Изменение данных, отраженное CdcEvent

Наименование	Описание
key()	Ключ для измененной записи.
value()	Значение измененной записи. Этот метод вернет значение null, если событие отражает удаление.
cacheId()	ID кэша, в котором происходит изменение. Значение равно CACHE_ID из SYS.CACHES.
partition()	Раздел измененной записи.
primary()	Флаг, чтобы различать, происходит операция на основном или резервном узле.
version()	Сопоставимая версия измененной записи. Внутри РЕД КВАНТ поддерживает упорядоченные версии каждой записи, поэтому любые изменения одной и той же записи можно отсортировать.

6.5.2.2 org.apache.ignite.cdc.CdcConsumer

Потребитель событий изменения. Это должно быть реализовано пользователем.

Методы представлены в таблице 11.

Таблица 11 - Методы org.apache.ignite.cdc.CdcConsumer

Наименование	Описание
void start(MetricRegistry)	Вызывается однократно при запуске приложения CDC. MetricRegistry следует использовать для экспорта метрик, специфичных для потребителя.
boolean onEvents(Iterator<CdcEvent> events)	Основной метод, обрабатывающий изменения. Когда этот метод возвращает значение true, состояние сохраняется на диске. Состояние указывает на событие, следующее за последним прочитанным событием. В случае любого сбоя потребление продолжится с последнего сохраненного состояния.
void stop()	Вызывается однократно при остановке приложения CDC.

6.5.3 Метрики

ignite-cdc.sh использует тот же SPI для экспорта метрик, что и РЕД КВАНТ. В таблице 12 представлены метрики, которые предоставляются приложением (дополнительные метрики могут быть предоставлены потребителем).

Таблица 12 - Метрики ignite-cdc.sh

Наименование	Описание
CurrentSegmentIndex	Индекс текущего обрабатываемого сегмента WAL.
CommittedSegmentIndex	Индекс сегмента WAL, содержащего последнее зафиксированное состояние.
CommittedSegmentOffset	Фиксированное смещение в байтах внутри сегмента WAL.
LastSegmentConsumptionTime	Метка времени (в миллисекундах), указывающая начало обработки последнего сегмента.
BinaryMetaDir	Двоичный метакаталог, из которого приложение считывает данные.
MarshallerDir	Каталог Marshaller, из которого приложение считывает данные.
CdcDir	Каталог CDC, из которого приложение считывает данные.

6.5.4 Ведение журнала

ignite-cdc.sh использует ту же конфигурацию ведения журнала, что и узел РЕД КВАНТ. Разница лишь в том, что лог записывается в файл «ignite-cdc.log».

6.5.5 Жизненный цикл

Внимание: ignite-cdc.sh реализует отказоустойчивый подход. Он просто терпит неудачу в случае любой ошибки. Процедура перезапуска должна быть настроена средствами операционной системы.

1. Найти необходимые общие каталоги. Взять значения из предоставленного IgniteConfiguration.
2. Заблокировать каталог CDC.
3. Загрузить сохраненное состояние.
4. Запустить потребителя.
5. Бесконечно ждать нового доступного сегмента и обрабатывать его.
6. Остановить потребитель в случае сбоя или получения стоп-сигнала.

6.5.6 cdc-ext

В проекте Ignite extensions есть модуль cdc-ext, который предоставляет два способа настройки межкластерной репликации на основе CDC.

7 КОНФИГУРАЦИЯ СНИМКОВ

РЕД КВАНТ предоставляет возможность создавать полные снимки кластера для развертываний с использованием РЕД КВАНТ Persistence. Моментальный снимок РЕД КВАНТ включает в себя согласованную копию всех записей данных, сохраненных на диске, и некоторых других файлов, необходимых для процедуры восстановления.

Структура моментального снимка аналогична структуре каталога хранилища РЕД КВАНТ Persistence за некоторыми исключениями. На рисунке 15 представлен снимок кластера, который имеет следующую структуру:

- Снимок находится в каталоге work\snapshots и называется backup23012020, где work — это рабочий каталог РЕД КВАНТ.
- Моментальный снимок создается для кластера из 3 узлов, все узлы которого работают на одном компьютере. В этом примере узлы называются node1, node2 и node3, хотя на практике имена совпадают с согласованными идентификаторами узлов.
- Моментальный снимок содержит копию кэша my-sample-cache.
- В папке db хранятся копии записей данных в файлах part-N.bin и cache_data.dat. Предварительная запись и контрольные точки не добавляются в моментальный снимок, если они не требуются для текущей процедуры восстановления.
- Каталоги binary_meta и marshaller хранят метаданные и информацию, относящуюся к marshaller.

Примечание: В примере показан моментальный снимок, созданный для кластера, работающего на той же физической машине. Таким образом, весь снимок находится в одном месте. На практике все узлы будут работать на разных машинах, а данные снимка будут распределены по всему кластеру. Каждый узел хранит сегмент снимка с данными, принадлежащими этому конкретному узлу. Процедура восстановления объясняет, как связать вместе все сегменты во время восстановления.

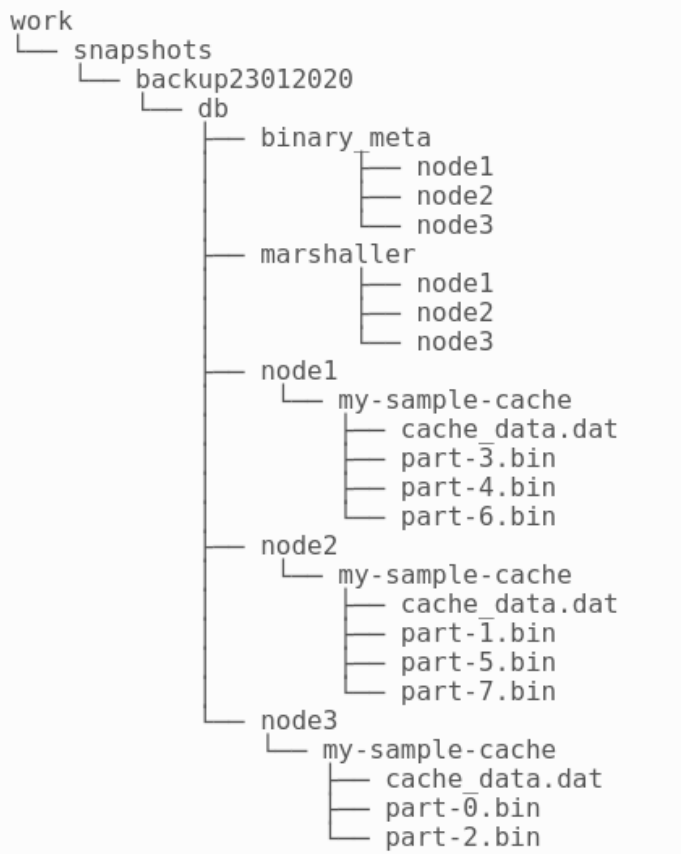

Рисунок 16 - Моментальный снимок кластера

7.1 Конфигурация

7.1.1 Каталог снимков

По умолчанию сегмент моментального снимка хранится в рабочем каталоге соответствующего узла РЕД КВАНТ и использует тот же носитель, на котором РЕД КВАНТ Persistence хранит данные, индекс, WAL и другие файлы. Поскольку моментальный снимок может занимать столько же места, сколько уже занято файлами персистентности, и может влиять на производительность приложения из-за совместного использования дискового ввода-вывода с подпрограммами РЕД КВАНТ Persistence, рекомендуется хранить моментальные снимки и файлы персистентности на разных носителях.

7.1.2 Пул выполнения моментальных снимков

По умолчанию размер пула потоков моментальных снимков имеет значение 4. Уменьшение количества потоков, участвующих в процессе создания моментального снимка, увеличивает общее время создания моментального снимка. Однако это позволяет поддерживать нагрузку на диск в разумных пределах.

7.2 Создание снимка

РЕД КВАНТ предоставляет несколько API для создания моментальных снимков. Ниже описаны доступные варианты API.

7.2.1 Использование сценария управления

РЕД КВАНТ поставляет сценарий управления, который поддерживает перечисленные ниже команды, связанные со снимками:

```
#Create a cluster snapshot in the background:
control.(sh|bat) --snapshot create snapshot_name

#Create a cluster snapshot (wait for the entire operation to complete):
control.(sh|bat) --snapshot create snapshot_name --sync

#Cancel a running snapshot:
control.(sh|bat) --snapshot cancel snapshot_name

#Kill a running snapshot:
control.(sh|bat) --kill SNAPSHOT snapshot_name
```

7.2.2 Использование JMX

Необходимо использовать интерфейс SnapshotMXBean для выполнения специфичных для моментального снимка процедур через JMX:

- createSnapshot(String snpName) — создание моментального снимка;
- cancelSnapshot(String snpName) — отмена снимка на узле, инициировавшем его создание.

7.2.3 Использование Java API

Также на Java можно создать снимок программно:

```
CacheConfiguration<Integer, String> ccfg = new CacheConfiguration<>("snapshot-cache");

try (IgniteCache<Integer, String> cache = ignite.getOrCreateCache(ccfg)) {
   cache.put(1, "Maxim");

   // Start snapshot operation.
   ignite.snapshot().createSnapshot("snapshot_02092020").get();
}
finally {
   ignite.destroyCache(ccfg.getName());
}
```

7.3 Проверка согласованности снимков

Обычно все узлы кластера работают на разных машинах, поэтому данные снимка распределены по всему кластеру. Каждый узел хранит свой собственный сегмент моментального снимка, поэтому в некоторых случаях может потребоваться проверка моментального снимка на предмет полноты данных и согласованности данных в кластере перед восстановлением из моментального снимка.

Для таких случаев РЕД КВАНТ поставляется со встроенными командами проверки согласованности моментальных снимков, которые позволяют проверить внутреннюю согласованность данных, вычислить хэши разделов данных и контрольные суммы страниц и распечатать результат в случае обнаружения проблемы. Команда проверки также сравнивает хэши основных разделов с соответствующими резервными разделами и сообщает о любых различиях.

7.4 Восстановление из снимка

Моментальный снимок можно восстановить либо вручную на остановленном кластере, либо автоматически на активном кластере. Обе процедуры описаны ниже, однако предпочтительнее использовать команду восстановления только из Control Script.

7.4.1 Процедура ручного восстановления моментального снимка

Структура моментального снимка аналогична структуре собственной персистентности РЕД КВАНТ, поэтому для ручного восстановления моментального снимка необходимо выполнять восстановление моментального снимка только в том же кластере с тем же consistentId узла и в той же топологии, на которой был сделан снимок. Если необходимо восстановить моментальный снимок в другом кластере или в другой топологии кластера, следует использовать процедуру автоматического восстановления моментального снимка.

Как правило, необходимо остановить кластер, затем заменить данные персистентности и другие файлы данными из моментального снимка и перезапустить узлы.

Подробная процедура выглядит следующим образом:

1. Остановить кластер, который необходимо восстановить.
2. Удалить все файлы из каталога контрольной точки $IGNITE_HOME/work/cp.
3. На каждом узле выполнить следующие действия:
    - Удалить файлы, связанные с {nodeId}, из каталога $IGNITE_HOME/work/db/binary_meta.
    - Удалить файлы, связанные с {nodeId}, из каталога $IGNITE_HOME/work/db/marshaller.
    - Удалить файлы и подкаталоги, связанные с {nodeId}, в каталоге $IGNITE_HOME/work/db. Отдельно очистить каталог db/{node_id}, если он не находится в рабочем каталоге РЕД КВАНТ.
    - Скопировать файлы, принадлежащие узлу с идентификатором {node_id}, из моментального снимка в каталог $IGNITE_HOME/work/. Если каталог db/{node_id} не находится в рабочем каталоге РЕД КВАНТ, необходимо скопировать туда файлы данных.
4. Перезапустить кластер.

7.4.2 Процедура автоматического восстановления моментального снимка

Процедура автоматического восстановления позволяет пользователю восстанавливать группы кэша из моментального снимка в активном кластере с помощью Java API или сценария командной строки.

В настоящее время эта процедура имеет несколько ограничений, которые будут устранены в будущих версиях:

- Восстановление возможно только в том случае, если все части снимка присутствуют в кластере. Каждый узел ищет данные локального снимка в настроенном пути к снимку по заданному имени снимка и согласованному идентификатору узла.
- Процедура восстановления может быть применена только к группам кэша, созданным пользователем.
- Группы кэша, подлежащие восстановлению из моментального снимка, не должны присутствовать в кластере. Если они присутствуют, они должны быть уничтожены пользователем перед началом этой операции.
- Параллельные операции восстановления не допускаются. Таким образом, если была запущена одна операция, другая может быть запущена только после завершения первой.

7.4.2.1 Восстановление группы кэша из моментального снимка

В следующем фрагменте кода показано, как восстановить отдельную группу кэша из моментального снимка.

```
// Restore cache named "snapshot-cache" from the snapshot "snapshot_02092020".
ignite.snapshot().restoreSnapshot("snapshot_02092020", Collections.singleton("snapshot-cache")).get();
```

7.4.2.2 Использование интерфейса командной строки для управления операцией восстановления

Сценарий control.sh|bat позволяет запускать, останавливать и получать статус операции восстановления.

```
# Start restoring all user-created cache groups from the snapshot "snapshot_09062021" in the background.
control.(sh|bat) --snapshot restore snapshot_09062021 --start

# Start restoring all user-created cache groups from the snapshot "snapshot_09062021" and wait for the entire operation to complete.
control.(sh|bat) --snapshot restore snapshot_09062021 --start --sync

# Start restoring only "cache-group1" and "cache-group2" from the snapshot "snapshot_09062021" in the background.
control.(sh|bat) --snapshot restore snapshot_09062021 --start --groups cache-group1,cache-group2

# Get the status of the restore operation for "snapshot_09062021".
control.(sh|bat) --snapshot restore snapshot_09062021 --status

# Cancel the restore operation for "snapshot_09062021".
```

```
control.(sh|bat) --snapshot restore snapshot_09062021 --cancel
```

7.5 Гарантии согласованности

С помощью РЕД КВАНТ все моментальные снимки полностью совместимы с точки зрения одновременных операций в масштабе всего кластера, а также текущих изменений. Сохраняемые данные, индекс, схема, бинарные метаданные, маршаллер и другие файлы на узлах.

Согласованность снимков в масштабе кластера достигается путем запуска процедуры Partition-Map-Exchange. Таким образом, кластер в конечном итоге дойдет до момента, когда все ранее запущенные транзакции будут завершены, а новые приостановлены. Как только это происходит, кластер инициирует процедуру создания моментального снимка. Процедура PME гарантирует, что моментальный снимок включает основную и резервную копии в согласованном состоянии.

Согласованность между файлами постоянного хранилища РЕД КВАНТ и их копиями моментальных снимков достигается путем копирования исходных файлов в целевой каталог моментальных снимков с отслеживанием всех одновременных текущих изменений. Для отслеживания изменений может потребоваться дополнительное место на носителе РЕД КВАНТ Persistence (до однократного размера носителя).

7.6 Текущие ограничения

Процедура моментального снимка имеет некоторые ограничения, о которых следует знать, прежде чем использовать эту функцию в производственной среде:

- Снимки определенных кэшей/таблиц не поддерживаются. Всегда создается полный снимок кластера.
- Кэши/таблицы, которые не сохраняются в РЕД КВАНТ Persistence, не включаются в моментальный снимок.
- Зашифрованные кэши в моментальном снимке должны быть зашифрованы одним и тем же главным ключом.
- Одновременно может выполняться только одна операция моментального снимка.
- Операция моментального снимка запрещена во время изменения главного ключа и/или изменения ключа группы кэша.
- Процедура моментального снимка прерывается, если серверный узел покидает кластер.

Если какое-либо из этих ограничений не позволяет использовать РЕД КВАНТ, следует выбрать альтернативные реализации моментальных снимков для РЕД КВАНТ, предоставляемые корпоративными поставщиками.

8 НАСТРОЙКА ТАБЛИЦ КЭШЕЙ

8.1 КОНФИГУРАЦИЯ КЭШЕЙ

В данном пункте объясняется, как можно установить параметры конфигурации кэша. После того, как создан кэш, нельзя менять параметры его конфигурации.

Примечание: Подход к конфигурации на основе кэша является одним из вариантов конфигурации. Также можно настроить кэши/таблицы, используя стандартные команды SQL, такие как CREATE TABLE.

8.1.1 ПРИМЕР КОНФИГУРАЦИИ

Ниже приведен пример конфигурации кэша.

```
CacheConfiguration cacheCfg = new CacheConfiguration("myCache");

cacheCfg.setCacheMode(CacheMode.PARTITIONED);
cacheCfg.setBackups(2);
cacheCfg.setRebalanceMode(CacheRebalanceMode.SYNC);
cacheCfg.setWriteSynchronizationMode(CacheWriteSynchronizationMode.FULL_SYNC);
cacheCfg.setPartitionLossPolicy(PartitionLossPolicy.READ_ONLY_SAFE);

IgniteConfiguration cfg = new IgniteConfiguration();
cfg.setCacheConfiguration(cacheCfg);

// Start a node.
Ignition.start(cfg);
```

В таблице 13 представлен список используемых параметров.

Таблица 13 - Параметры конфигурации

Параметр конфигурации	Описание	Значение по умолчанию
name	Имя кэша	Нет
cacheMode	Параметр cacheMode определяет способ распределения данных в кластере. В режиме PARTITIONED (по умолчанию) общий набор данных делится на разделы, и все разделы распределяются между участвующими узлами сбалансированным образом. В режиме REPLICATED все данные реплицируются на каждый узел в кластере.	PARTITIONED
writeSynchronizationMode	Режим синхронизации записи. См. Конфигурация бэкапов разделов	PRIMARY_SYNC
rebalanceMode	Этот параметр управляет способом выполнения процесса перебалансировки. Возможные значения:	ASYNC

Параметр конфигурации	Описание	Значение по умолчанию
	• SYNC — Любые запросы к API кэша блокируются до завершения ребалансировки. • ASYNC (по умолчанию) — Ребалансировка выполняется в фоновом режиме. • NONE — Ребалансировка не запускается.	
backups	Количество резервных разделов для кэша.	0
partitionLossPolicy	Политика потери раздела.	IGNORE
readFromBackup	Чтение запрошенной записи кэша из резервного раздела, если он доступен на локальном узле, вместо того, чтобы делать запрос к основному разделу (который может быть расположен на удаленных узлах).	true
queryPrallelism	Количество потоков в одном узле для обработки SQL-запроса, выполняемого в кэше.	1

8.1.2 Шаблоны кэша

Шаблон кэша — это экземпляр CacheConfiguration, который можно зарегистрировать в кластере и использовать позже в качестве основы для создания новых кэшей или таблиц SQL. Кэш или таблица, созданная на основе шаблона, наследует все свойства шаблона.

Шаблоны полезны при создании таблицы с помощью команды CREATE TABLE, так как эта команда не поддерживает все доступные параметры кэша.

Примечание: В настоящее время шаблоны поддерживаются для команд CREATE TABLE и REST.

Чтобы создать шаблон, необходимо определить конфигурацию кэша и добавить ее в экземпляр РЕД КВАНТ, как показано ниже. Если нужно определить шаблон кэша в файле конфигурации XML, необходимо добавить звездочку к имени шаблона. Это необходимо, чтобы указать, что конфигурация является шаблоном, а не фактическим кэшем.

```
IgniteConfiguration igniteCfg = new IgniteConfiguration();

try (Ignite ignite = Ignition.start(igniteCfg)) {
  CacheConfiguration cacheCfg = new CacheConfiguration("myCacheTemplate");

  cacheCfg.setBackups(2);
  cacheCfg.setCacheMode(CacheMode.PARTITIONED);

  // Register the cache template
  ignite.addCacheConfiguration(cacheCfg);
```

```
}
```

Как только шаблон кэша будет зарегистрирован в кластере, как показано во фрагменте кода выше, его можно использовать для создания другого кэша с той же конфигурацией.

8.2 КОНФИГУРАЦИЯ БЭКАПОВ РАЗДЕЛОВ

По умолчанию РЕД КВАНТ хранит одну копию каждого раздела (одну копию всего набора данных). В этом случае, если один или несколько узлов становятся недоступными, теряется доступ к разделам, хранящимся на этих узлах. Чтобы избежать этого, можно настроить РЕД КВАНТ на сохранение резервных копий каждого раздела.

Внимание: По умолчанию резервное копирование отключено.

Резервные копии настраиваются для каждого кэша (таблицы). Если настроить 2 резервные копии, кластер будет поддерживать 3 копии каждого раздела. Один из разделов называется основным, а два других — резервными. В более широком смысле узел, имеющий основной раздел, называется первичным узлом для ключей, хранящихся в этом разделе. Узел с резервными разделами называется резервным узлом.

Когда узел с основным разделом для некоторого ключа покидает кластер, РЕД КВАНТ запускает процесс обмена картами разделов (PME). PME помечает один из резервных разделов (если они настроены) для ключа как первичный.

Резервные разделы повышают доступность данных, а в некоторых случаях и скорость операций чтения, поскольку РЕД КВАНТ считывает данные из резервных копий разделов, если они доступны на локальном узле (это поведение по умолчанию, которое можно отключить. См. readFromBackup в пункте Пример конфигурации). Однако они также увеличивают потребление памяти или размер постоянного хранилища (если они включены).

8.2.1 НАСТРОЙКА РЕЗЕРВНЫХ КОПИЙ

Чтобы настроить количество резервных копий, необходимо задать свойство backups в конфигурации кэша.

```
CacheConfiguration cacheCfg = new CacheConfiguration();

cacheCfg.setName("cacheName");
cacheCfg.setCacheMode(CacheMode.PARTITIONED);
cacheCfg.setBackups(1);

IgniteConfiguration cfg = new IgniteConfiguration();

cfg.setCacheConfiguration(cacheCfg);

// Start the node.
Ignite ignite = Ignition.start(cfg);
```

8.2.2 Синхронные и асинхронные резервные копии

Существует несколько режимов синхронизации основных и резервных записей. Режим синхронизации записи можно установить в конфигурации кэша:

```
CacheConfiguration cacheCfg = new CacheConfiguration();

cacheCfg.setName("cacheName");
cacheCfg.setBackups(1);
cacheCfg.setWriteSynchronizationMode(CacheWriteSynchronizationMode.FULL_SYNC);
IgniteConfiguration cfg = new IgniteConfiguration();

cfg.setCacheConfiguration(cacheCfg);

// Start the node.
Ignition.start(cfg);
```

Для режима синхронизации записи можно установить следующие значения:

- FULL_SYNC - Клиентский узел будет ожидать завершения записи или коммита на всех участвующих удаленных узлах (основном и резервном).
- FULL_ASYNC - Клиентский узел не ожидает ответов от участвующих узлов, и в этом случае удаленные узлы могут немного обновить свое состояние после завершения любого из методов записи в кэш или после завершения метода Transaction.commit().
- PRIMARY_SYNC - Это режим «по умолчанию». Клиентский узел будет ожидать завершения записи или коммита на основном узле, но не будет ожидать обновления резервных копий.

8.3 Политика потери раздела

На протяжении жизненного цикла кластера может случиться так, что некоторые разделы данных будут потеряны из-за отказа основного и резервного узлов для разделов. Такая ситуация приводит к частичной потере данных и требует решения в соответствии с выбранным вариантом использования.

Раздел считается потерянным, когда и основная копия, и все резервные копии раздела недоступны для кластера, т.е. когда основной и резервный узлы для раздела становятся недоступными. Это означает, что для данного кэша нельзя позволить потерять больше узлов, чем number_of_backups. Можно установить количество резервных разделов для кэша в конфигурации кэша.

При изменении топологии кластера РЕД КВАНТ проверяет, не привело ли это изменение к потере разделов, и, в зависимости от настроенной политики потери разделов и базовых параметров автонастройки, разрешает или запрещает операции с кэшами.

Для чистых кэшей в памяти, когда раздел потерян, данные из раздела не могут быть восстановлены, пока они не будут снова загружены в кластер. Для постоянных кэшей данные физически не теряются, поскольку они сохраняются на диск. Когда отказавшие или отключившиеся узлы возвращаются в кластер (после перезагрузки), данные загружаются с

диска. В этом случае необходимо сбросить состояние потерянных разделов, чтобы продолжить использовать данные. См. Обработка потери раздела.

8.3.1 Настройка политики потери раздела

РЕД КВАНТ поддерживает следующие политики потери разделов:

- IGNORE - Потеря раздела игнорируется. Кластер обрабатывает потерянные разделы так, как если бы они были пустыми. Когда запрашиваются данные из таких разделов, кластер возвращает пустые значения, как будто данных никогда не было. Эта политика может использоваться только в in-memory кластерах, где базовая автоматическая настройка включена с тайм-аутом 0, и является значением по умолчанию для таких конфигураций. Во всех других конфигурациях (кластеры, в которых есть хотя бы один регион данных с персистентностью) политика IGNORE заменяется на READ_WRITE_SAFE, даже если было явно задано IGNORE в конфигурации кэша.
- READ_WRITE_SAFE - Любая попытка чтения из потерянного раздела кэша или записи в него приводит к возникновению исключения. Однако можно читать/записывать доступные разделы.
- READ_ONLY_SAFE - Кэш доступен в режиме только для чтения. Операции записи в кэш приводят к исключению. Операции чтения из потерянных разделов также приводят к исключению. См. пункт Обработка потери раздела.

Политика потери разделов настраивается для каждого кэша.

```
CacheConfiguration cacheCfg = new CacheConfiguration("myCache");

cacheCfg.setPartitionLossPolicy(PartitionLossPolicy.READ_ONLY_SAFE);
```

8.3.2 Прослушивание событий потери раздела

Можно прослушивать событие EVT_CACHE_REBALANCE_PART_DATA_LOST, чтобы получать уведомления о потере раздела. Это событие запускается для каждого потерянного раздела и содержит номер потерянного раздела и идентификатор узла, на котором находился раздел. События потери раздела инициируются только при использовании политики READ_WRITE_SAFE или READ_ONLY_SAFE.

Сначала необходимо включить событие в конфигурации кластера.

```
Ignite ignite = Ignition.start();

IgnitePredicate<Event> locLsnr = evt -> {
  CacheRebalancingEvent cacheEvt = (CacheRebalancingEvent) evt;

  int lostPart = cacheEvt.partition();

  ClusterNode node = cacheEvt.discoveryNode();

  System.out.println(lostPart);
```

```
    return true; // Continue listening.
};

ignite.events().localListen(locLsnr, EventType.EVT_CACHE_REBALANCE_PART_DATA_LOST);
```

8.3.3 Обработка потери раздела

Если данные физически не потеряны, можно вернуть узлы, покинувшие кластер, и сбросить состояние потерянных разделов, чтобы продолжить работу с данными. Можно сбросить состояние потерянного раздела, вызвав Ignite.resetLostPartitions(cacheNames) для определенных кэшей или через сценарий управления.

Вызов Ignite.resetLostPartitions(cacheNames):

```
ignite.resetLostPartitions(Arrays.asList("myCache"));
```

Команда скрипта управления:

```
control.sh --cache reset_lost_partitions myCache
```

Если не сбросить потерянные разделы, операции чтения и записи (в зависимости от политики, настроенной для кэша) из потерянных разделов вызовут исключение CacheException. Можно проверить, вызвано ли исключение состоянием разделов, проанализировав его основную причину, например:

```
IgniteCache<Integer, Integer> cache = ignite.cache("myCache");

try {
    Integer value = cache.get(3);
    System.out.println(value);
} catch (CacheException e) {
    if (e.getCause() instanceof CacheInvalidStateException) {
        System.out.println(e.getCause().getMessage());
    } else {
        e.printStackTrace();
    }
}
```

Через IgniteCache.lostPartitions() можно получить список потерянных разделов для кэша.

```
IgniteCache<Integer, String> cache = ignite.cache("myCache");

cache.lostPartitions();
```

8.3.4 Восстановление после потери раздела

В следующих пунктах объясняется, как можно восстановиться после потери раздела в различных конфигурациях кластера.

8.3.4.1 Чистый in-memory кластер с политикой IGNORE

В этой конфигурации политика IGNORE применима только в том случае, если базовая автоматическая настройка включена с тайм-аутом 0, что является настройкой по умолчанию для in-memory кластеров. Для таких конфигураций потеря раздела игнорируется. Кэш продолжает работать, а потерянные разделы считаются пустыми.

Если базовая автоматическая корректировка отключена или тайм-аут больше 0, политика IGNORE заменяется на READ_WRITE_SAFE.

8.3.4.2 Чистый in-memory кластер с политикой READ_WRITE_SAFE или READ_ONLY_SAFE

Пользовательские операции блокируются до тех пор, пока не будут сброшены потерянные разделы. После сброса можно продолжать использовать кэш, но данные будут потеряны.

Когда базовая автоматическая настройка отключена или время ожидания больше 0, необходимо вернуть узлы (как минимум один владелец раздела для каждого раздела) в базовую топологию перед сбросом потерянных разделов. В противном случае Ignite.resetLostPartitions(cacheNames) сгенерирует исключение ClusterTopologyCheckedException с сообщением *«Невозможно сбросить потерянные разделы, поскольку ни один базовый узел не подключен к сети [cache=someCahe, partition=someLostPart]»*, указывающее, что безопасное восстановление невозможно. Если вернуть узлы не представляется возможным по какой-либо причине (например, из-за аппаратного сбоя), необходимо исключить их из базовой топологии вручную, прежде чем пытаться сбросить потерянные разделы.

8.3.4.3 Кластеры с персистентностью

В кластерах, где все регионы данных настроены на сохранение данных на диске (регионов в памяти нет), существует два способа восстановления после потери раздела (при условии, что данные не повреждены физически):

1. Вернуть все узлы к базовой топологии,
2. Сбросить потерянные разделы (вызвать Ignite.resetLostPartitions(…) для всех кэшей).

или

1. Остановить все узлы,
2. Запустить все узлы, включая отказавшие, и активировать кластер.

Если некоторые узлы не могут быть возвращены, необходимо исключить их из базовой топологии, прежде чем пытаться сбросить состояние потерянных разделов.

8.3.4.4 Кластеры с кэшем в памяти и постоянным кэшем

В кластерах, где есть как регионы в памяти, так и постоянные регионы, кэши в памяти обрабатываются так же, как в чистых кластерах в памяти с политикой потери разделов,

установленной на READ_WRITE_SAFE, а постоянные кэши обрабатываются так же, как в постоянных кластерах.

8.4 Режимы атомарности

По умолчанию кэш поддерживает только атомарные операции, а массовые операции, такие как putAll() или removeAll(), выполняются как последовательность отдельных операций размещения и удаления. Можно включить поддержку транзакций и сгруппировать несколько операций кэширования для одного или нескольких ключей в одну атомарную транзакцию. Эти операции выполняются без каких-либо других чередующихся операций над указанными ключами, и либо все завершаются успешно, либо все терпят неудачу. Частичного выполнения операций нет.

Чтобы включить поддержку транзакций для кэша, необходимо установить для параметра atomicityMode в конфигурации кэша значение TRANSACTIONAL.

Внимание! Если настраивается несколько кэшей в одной группе кэшей, все кэши должны быть либо атомарными, либо транзакционными. Нельзя иметь кэша TRANSACTIONAL и ATOMIC в одной группе кэшей.

РЕД КВАНТ поддерживает 3 режима атомарности, которые представлены в таблице 14.

Таблица 14 - Режимы атомарности

Режим атомарности	Описание
ATOMIC	Режим по умолчанию. Все операции выполняются атомарно, по одной за раз. Транзакции не поддерживаются. Режим ATOMIC обеспечивает более высокую производительность за счет исключения блокировок транзакций, обеспечивая при этом атомарность данных и согласованность для каждой отдельной операции. Массовые записи, такие как методы putAll(…) и removeAll(…), не выполняются за одну транзакцию и могут частично завершиться сбоем. В этом случае создается исключение CachePartialUpdateException, содержащее список ключей, для которых не удалось выполнить обновление.
TRANSACTIONAL	Включает поддержку ACID-совместимых транзакций, выполняемых через API «ключ-значение». Транзакции SQL не поддерживаются. Транзакции в этом режиме могут иметь разные режимы параллелизма и уровни изоляции. Данный режим следует включать в том слуае, если нужна поддержка ACID-совместимых операций. **Примечание:** Режим TRANSACTIONAL увеличивает производительность операций кэширования и должен быть включен только в том случае, если нужны транзакции.
TRANSACTIONAL_SNAPSHOT	Экспериментальный режим, который реализует управление параллелизмом с несколькими версиями

Режим атомарности	Описание
	(MVCC) и поддерживает как транзакции «ключ-значение», так и транзакции SQL. **Внимание!** TRANSACTIONAL_SNAPSHOT устарел с версии 2.12 и будет удален в следующих выпусках.

Транзакции для кэша можно включить в конфигурации кэша.

```
CacheConfiguration cacheCfg = new CacheConfiguration();

cacheCfg.setName("cacheName");

cacheCfg.setAtomicityMode(CacheAtomicityMode.TRANSACTIONAL);

IgniteConfiguration cfg = new IgniteConfiguration();

cfg.setCacheConfiguration(cacheCfg);

// Optional transaction configuration. Configure TM lookup here.
TransactionConfiguration txCfg = new TransactionConfiguration();

cfg.setTransactionConfiguration(txCfg);

// Start a node
Ignition.start(cfg);
```

9 НАСТРОЙКА ПАРАМЕТРОВ РЕБАЛАНСА

Когда новый узел присоединяется к кластеру, некоторые разделы перемещаются на новый узел, чтобы данные оставались равномерно распределенными в кластере. Этот процесс называется ребалансировкой данных.

Если существующий узел навсегда покидает кластер, а резервные копии не настроены, разделы, хранящиеся на этом узле, теряются. Когда настроено резервное копирование, одна из резервных копий потерянных разделов становится основным разделом, и инициируется процесс ребалансировки.

Внимание: Ребалансировка данных инициируется изменениями в базовой топологии. В кластерах, работающих исключительно в памяти, по умолчанию ребалансировка начинается немедленно, когда узел покидает кластер или присоединяется к нему (базовая топология изменяется автоматически). В кластерах с сохранением базовая топология должна быть изменена вручную (поведение по умолчанию) или может быть изменена автоматически, если включена автоматическая корректировка базовой линии.

Ребалансировка настраивается для каждого кэша.

9.1 НАСТРОЙКА РЕЖИМА РЕБАЛАНСИРОВКИ

РЕД КВАНТ поддерживает как синхронную, так и асинхронную ребалансировку. В синхронном режиме любые операции с данными кэша блокируются до завершения ребалансировки. В асинхронном режиме процесс ребалансировки выполняется асинхронно. Также можно отключить ребалансировку для определенного кэша.

Чтобы изменить режим ребалансировки, необходимо установить одно из следующих значений в конфигурации кэша.

- SYNC — режим синхронной ребалансировки. В этом режиме любой вызов общедоступного API кэша блокируется до завершения ребалансировки.
- ASYNC — режим асинхронной ребалансировки. Распределенные кэши доступны сразу и загружают все необходимые данные с других доступных узлов кластера в фоновом режиме.
- NONE — в этом режиме ребалансировка не выполняется, что означает, что кэши либо загружаются по запросу из постоянного хранилища при каждом доступе к данным, либо заполняются явно.

```
package org.apache.ignite.snippets;

import org.apache.ignite.Ignite;
import org.apache.ignite.Ignition;
import org.apache.ignite.cache.CacheRebalanceMode;
import org.apache.ignite.configuration.CacheConfiguration;
import org.apache.ignite.configuration.IgniteConfiguration;

public class RebalancingConfiguration {

    public static void main(String[] args) {
```

```
        RebalancingConfiguration rc = new RebalancingConfiguration();

        rc.configure();
    }

    void configure() {
        IgniteConfiguration cfg = new IgniteConfiguration();

        CacheConfiguration cacheCfg = new CacheConfiguration("mycache");

        cacheCfg.setRebalanceMode(CacheRebalanceMode.SYNC);

        cfg.setCacheConfiguration(cacheCfg);

        // Start a node.
        Ignite ignite = Ignition.start(cfg);

        ignite.close();
    }

}
```

9.2 НАСТРОЙКА ПУЛА ПОТОКОВ РЕБАЛАНСИРОВКИ

По умолчанию ребалансировка выполняется в один поток на каждом узле. Это означает, что в каждый момент времени используется только один поток для передачи пакетов от одного узла к другому или для обработки пакетов, поступающих с удаленного узла.

Можно увеличить количество потоков, которые берутся из пула системных потоков и используются для ребалансировки. Системный поток берется из пула каждый раз, когда узлу необходимо отправить пакет данных на удаленный узел или обработать пакет, пришедший с удаленного узла. Поток освобождается после обработки пакета.

```
package org.apache.ignite.snippets;

import org.apache.ignite.Ignite;
import org.apache.ignite.Ignition;
import org.apache.ignite.cache.CacheRebalanceMode;
import org.apache.ignite.configuration.CacheConfiguration;
import org.apache.ignite.configuration.IgniteConfiguration;

public class RebalancingConfiguration {

    public static void main(String[] args) {
        RebalancingConfiguration rc = new RebalancingConfiguration();

        rc.configure();
    }

    void configure() {
        IgniteConfiguration cfg = new IgniteConfiguration();

        cfg.setRebalanceThreadPoolSize(4);
```

```
        CacheConfiguration cacheCfg = new CacheConfiguration("mycache");
        cfg.setCacheConfiguration(cacheCfg);

        // Start a node.
        Ignite ignite = Ignition.start(cfg);

        ignite.close();
    }

}
```

Внимание: Пул системных потоков широко используется внутри всех операций, связанных с кэшем (put, get и т. д.), механизма SQL и других модулей. Установка большого значения для размера пула потоков ребалансировки может значительно повысить производительность ребалансировки за счет снижения пропускной способности.

9.3 РЕБАЛАНСИРОВКА РЕГУЛИРОВАНИЯ СООБЩЕНИЙ

Когда данные передаются с одного узла на другой, весь набор данных разбивается на пакеты, и каждый пакет отправляется в отдельном сообщении. Можно настроить размер пакета и время ожидания узла между сообщениями.

```
package org.apache.ignite.snippets;


import org.apache.ignite.Ignite;
import org.apache.ignite.Ignition;
import org.apache.ignite.cache.CacheRebalanceMode;
import org.apache.ignite.configuration.CacheConfiguration;
import org.apache.ignite.configuration.IgniteConfiguration;

public class RebalancingConfiguration {

    public static void main(String[] args) {
        RebalancingConfiguration rc = new RebalancingConfiguration();

        rc.configure();
    }

    void configure() {
        IgniteConfiguration cfg = new IgniteConfiguration();

        CacheConfiguration cacheCfg = new CacheConfiguration("mycache");

        cfg.setRebalanceBatchSize(2 * 1024 * 1024);
        cfg.setRebalanceThrottle(100);

        cfg.setCacheConfiguration(cacheCfg);

        // Start a node.
        Ignite ignite = Ignition.start(cfg);

        ignite.close();
    }
```

```
    }
```

9.4 Другие свойства

В таблице 15 перечислены свойства CacheConfiguration, связанные с ребалансировкой.

Таблица 15 - Свойства CacheConfiguration

Свойство	Описание	Значение по умолчанию
rebalanceBatchSize	Размер в байтах одного сообщения перебалансировки. Алгоритм ребалансировки разбивает данные на каждом узле на несколько пакетов перед отправкой на другие узлы.	512 КБ
rebalanceThrottle	См. Ребалансировка регулирования сообщений.	0 (регулирование отключено)
rebalanceOrder	Порядок, в котором следует выполнять ребалансировку. Порядок ребалансировки может быть установлен на ненулевое значение только для кэшей с режимами перебалансировки SYNC или ASYNC. Сначала выполняется ребалансировка кэшей с меньшим порядком ребалансировки. По умолчанию ребалансировка не задана.	0
rebalanceTimeout	Тайм-аут для ожидающих сообщений о ребалансировке при обмене ими между узлами.	10 секунд

9.5 Мониторинг процесса ребалансировки

Отслеживать процесс ребалансировки для определенных кэшей можно с помощью JMX.

10 НАСТРОЙКА ПРОИЗВОДИТЕЛЬНОСТИ И ОСНОВНЫЕ ПРОБЛЕМЫ

10.1 Общие советы по повышению производительности

РЕД КВАНТ как распределенное хранилище и платформа требует определенных методов оптимизации. Прежде чем углубиться в более сложные методы, следует рассмотреть следующий базовый контрольный список:

- РЕД КВАНТ разработан и оптимизирован для сценариев распределенных вычислений. Следует разворачивать и тестировать кластер с несколькими узлами, а не с одним.
- РЕД КВАНТ одинаково хорошо масштабируется как по горизонтали, так и по вертикали. Таким образом, следует рассмотреть возможность выделения узлу РЕД КВАНТ всех ресурсов процессора и оперативной памяти, доступных на локальной машине. Рекомендуемая конфигурация — один узел на физическую машину.
- В тех случаях, когда РЕД КВАНТ развернут в виртуальной или облачной среде, идеально (но не обязательно) закрепить узел РЕД КВАНТ на одном хосте. Это предоставляет два преимущества:
  - Позволяет избежать проблемы «шумного соседа», когда РЕД КВАНТ VM будет конкурировать за ресурсы хоста с другими приложениями. Это может вызвать скачки производительности в кластере РЕД КВАНТ.
  - Обеспечивает высокую доступность. Если хост выходит из строя, и к нему прикреплены две или более виртуальных машин узла сервера РЕД КВАНТ, это может привести к потере данных.
- Если позволяют ресурсы, стоит хранить весь набор данных в оперативной памяти. Несмотря на то, что РЕД КВАНТ может хранить данные на диске и работать с ними, его архитектура ориентирована прежде всего на память. Другими словами, чем больше данных кэшируется в оперативной памяти, тем выше производительность. Следует правильно сконфигурировать и настроить память.
- Это может показаться противоречащим приведенному выше пункту, но недостаточно просто поместить данные в оперативную память и ожидать повышения производительности на порядок. Нужно быть готовым скорректировать модель данных и существующие приложения, если таковые имеются. Необходимо использовать концепцию совместного размещения на этапе моделирования данных для правильного распределения данных. Например, если данные размещены правильно, можно выполнять SQL-запросы с JOIN в больших масштабах и ожидать значительного повышения производительности.
- Если используется встроенная персистентность, нужно следовать этим методам оптимизации персистентности.
- Если планируется запускать SQL с помощью РЕД КВАНТ, стоит ознакомиться с оптимизацией, связанной с SQL.

- Чтобы обеспечить более быстрое выполнение ребалансировки при изменении топологии кластера необходимо настроить параметры перебалансировки данных.

10.2 Настройка производительности RAM и JVM

В данном пункте представлены рекомендации по настройке памяти, которые подходят для развертываний с собственной сохраняемостью или внешним хранилищем и без них. Несмотря на то, что РЕД КВАНТ хранит данные и индексы в Java heap, Java heap по-прежнему используется для хранения объектов, созданных запросами и операциями, выполняемыми приложениями. Таким образом, следует рассмотреть определенные рекомендации по оптимизации, связанной с JVM и сборкой мусора (GC).

10.2.1 Настройка параметров подкачки

Операционная система начинает подкачивать страницы из оперативной памяти на диск, когда общее использование оперативной памяти достигает определенного порога. Замена может повлиять на производительность кластера РЕД КВАНТ. Чтобы этого не произошло, можно изменить настройки операционной системы. Для Unix лучший вариант — либо уменьшить параметр vm.swappiness до 10, либо установить его в 0, если включено собственное сохранение:

```
sysctl -w vm.swappiness=0
```

Значение этого параметра также может увеличить паузы GC. Например, если журналы GC показывают записи low user time, high system time, long GC pause, это может быть вызвано подкачкой страниц Java heap. Чтобы решить эту проблему, необходимо воспользоваться указанной выше настройкой swappiness.

10.2.2 Совместное использование оперативной памяти с ОС и приложениями

Оперативная память отдельной машины совместно используется операционной системой, РЕД КВАНТ и другими приложениями. В качестве общей рекомендации, если кластер РЕД КВАНТ развернут только в режиме in-memory (собственное сохранение отключено), то не следует выделять более 90% объема оперативной памяти для узлов РЕД КВАНТ.

С другой стороны, если используется встроенная персистентность, то ОС требует дополнительной оперативной памяти для кэша страниц, чтобы оптимально синхронизировать данные с диском. Если кэш страниц не отключен, то не стоит отдавать под РЕД КВАНТ более 70% оперативной памяти сервера.

Примеры конфигурации см. в пункте Конфигурация регионов памяти.

В дополнение к этому, поскольку использование native persistence может привести к высокой загрузке кэша страниц, демон kswapd может не успевать за восстановлением страниц, которое используется кэшем страниц в фоновом режиме. В результате это может вызвать

большие задержки из-за прямого восстановления страниц и привести к длительным паузам GC.

Чтобы обойти эффекты, вызванные освобождением памяти страниц в Linux, необходимо добавить дополнительные байты между wmark_min и wmark_low с помощью /proc/sys/vm/extra_free_kbytes:

```
sysctl -w vm.extra_free_kbytes=1240000
```

10.2.3 НАСТРОЙКА JAVA HEAP И GC

Несмотря на то, что РЕД КВАНТ хранит данные в своих собственных регионах памяти off-heap, невидимых для сборщиков мусора Java, Java heap по-прежнему используется для объектов, генерируемых рабочими нагрузками приложений. Например, всякий раз, когда запускаются SQL-запросы к кластеру РЕД КВАНТ, запросы будут обращаться к данным и индексам, хранящимся в памяти off-heap, в то время как наборы результатов таких запросов будут храниться в Java Heap до тех пор, пока приложение не прочитает наборы результатов. Таким образом, в зависимости от пропускной способности и типа операций, Java Heap по-прежнему может интенсивно использоваться, и это может потребовать настройки, связанной с JVM и GC для рабочих нагрузок.

Ниже представлены некоторые общие рекомендации. Можно начать с них и вносить дополнительные коррективы по мере необходимости в зависимости от специфики приложений.

10.2.3.1 Общие настройки GC

Ниже приведены наборы примеров конфигураций JVM для приложений, которые могут интенсивно использовать Java Heap на серверных узлах, вызывая, таким образом, длительные - или частые, кратковременные - паузы GC stop-the-world.

Для развертываний JDK 1.8+ следует использовать сборщик мусора G1. Приведенные ниже настройки являются хорошей отправной точкой, если 10 ГБ heap более чем достаточно для серверных узлов:

```
-server
-Xms10g
-Xmx10g
-XX:+AlwaysPreTouch
-XX:+UseG1GC
-XX:+ScavengeBeforeFullGC
-XX:+DisableExplicitGC
```

Если G1 не работает, стоит рассмотреть возможность использования сборщика CMS и начать со следующих настроек. Стоит обратить внимание, что в качестве примера используется heap размером 10 ГБ, и для выбранного варианта использования может быть достаточно heap меньшего размера:

```
-server
-Xms10g
```

```
-Xmx10g
-XX:+AlwaysPreTouch
-XX:+UseParNewGC
-XX:+UseConcMarkSweepGC
-XX:+CMSClassUnloadingEnabled
-XX:+CMSPermGenSweepingEnabled
-XX:+ScavengeBeforeFullGC
-XX:+CMSScavengeBeforeRemark
-XX:+DisableExplicitGC
```

Примечание: Если используется встроенная персистентность РЕД КВАНТ, то рекомендуется установить для параметра JVM MaxDirectMemorySize значение walSegmentSize * 4. При настройках WAL по умолчанию это значение равно 256 МБ.

10.2.3.2 Расширенная настройка памяти

В средах Linux и Unix возможно, что приложение может столкнуться с длительными паузами GC или снижением производительности из-за ввода-вывода или нехватки памяти из-за конкретных настроек ядра. Ниже приведены некоторые рекомендации о том, как изменить настройки ядра, чтобы избежать длительных пауз GC.

Внимание! Все приведенные ниже команды оболочки были протестированы на RedHat 7. Они могут отличаться для используемого дистрибутива Linux. Перед изменением настроек ядра необходимо обязательно проверить системную статистику/журналы, чтобы убедиться, что действительно есть проблема. Перед внесением изменений на уровне ядра Linux в рабочую среду следует проконсультироваться с IT-отделом.

Если журналы GC показывают low user time, high system time, long GC pause, то, скорее всего, ограничения памяти вызывают подкачку или сканирование свободного места в памяти.

- Необходимо проверить и настроить параметры swappiness.
- Добавить -XX:+AlwaysPreTouch в настройки JVM при запуске.
- Отключить оптимизацию восстановления зоны NUMA.

```
sysctl -w vm.zone_reclaim_mode = 0
```

- Отключить Transparent Huge Pages, если используется дистрибутив RedHat.

```
echo never > /sys/kernel/mm/redhat_transparent_hugepage/enabled
echo never > /sys/kernel/mm/redhat_transparent_hugepage/defrag
```

10.2.3.3 Расширенная настройка ввода/вывода

Если журналы GC показывают low user time, low system time, long GC pause, возможно, потоки GC проводят слишком много времени в пространстве ядра, блокируемом различными действиями ввода-вывода. Например, это может быть вызвано блокировкой журнала, gzip или процедурами переноса журнала.

В качестве решения можно попробовать изменить интервал очистки страницы с 30 секунд по умолчанию на 5 секунд:

```
sysctl -w vm.dirty_writeback_centisecs=500
sysctl -w vm.dirty_expire_centisecs=500
```

См. пункт Настройка производительности слоя хранения, чтобы узнать об оптимизации, связанной с диском. Эти оптимизации могут положительно сказаться на сборщике мусора.

10.3 Настройка производительности слоя хранения

10.3.1 Настройка размера страницы

Параметр DataStorageConfiguration.pageSize должен быть не меньше наименьшего из следующих значений: размер страницы носителя (SSD, Flash, HDD и т. д.) и размера страницы кэша операционной системы. Значение по умолчанию — 4 КБ.

Размер страницы кэша операционной системы можно легко проверить с помощью системных инструментов и параметров.

Размер страницы устройства хранения, такого как SSD, обычно указывается в спецификации устройства. Если производитель не раскрывает эту информацию, можно запустить тесты SSD, чтобы выяснить число. Многим производителям приходится адаптировать свои драйверы для рабочих нагрузок с произвольной записью 4 КБ, потому что множество стандартных тестов используют 4 КБ по умолчанию.

Как только будет выбран наиболее оптимальный размер страницы, необходимо применить его в конфигурации кластера:

```
IgniteConfiguration cfg = new IgniteConfiguration();

// Durable memory configuration.
DataStorageConfiguration storageCfg = new DataStorageConfiguration();

// Changing the page size to 8 KB.
storageCfg.setPageSize(8192);

cfg.setDataStorageConfiguration(storageCfg);
```

10.3.2 Хранение WAL отдельно

Можно рассмотреть возможность использования отдельных дисков для файлов данных и ведения журнала с опережающей записью (WAL). РЕД КВАНТ активно записывает как файлы данных, так и файлы WAL.

В примере ниже показано, как настроить отдельные пути для хранилища данных, WAL и архива WAL:

```
IgniteConfiguration cfg = new IgniteConfiguration();

// Configuring Native Persistence.
```

```
DataStorageConfiguration storeCfg = new DataStorageConfiguration();

// Sets a path to the root directory where data and indexes are to be persisted.
// It's assumed the directory is on a separated SSD.
storeCfg.setStoragePath("/ssd/storage");

// Sets a path to the directory where WAL is stored.
// It's assumed the directory is on a separated HDD.
storeCfg.setWalPath("/wal");

// Sets a path to the directory where WAL archive is stored.
// The directory is on the same HDD as the WAL.
storeCfg.setWalArchivePath("/wal/archive");

cfg.setDataStorageConfiguration(storeCfg);

// Starting the node.
Ignite ignite = Ignition.start(cfg);
```

10.3.3 Увеличение размера сегмента WAL

Размер сегмента WAL по умолчанию (64 МБ) может быть неэффективным в сценариях с высокой нагрузкой, поскольку это приводит к слишком частому переключению WAL между сегментами, а переключение/ротация является дорогостоящей операцией. Установка большего размера сегмента (до 2 ГБ) может помочь уменьшить количество операций переключения. Однако компромисс заключается в том, что это увеличит общий объем журнала предварительной записи.

10.3.4 Изменение режима WAL

Также можно рассмотреть другие режимы WAL в качестве альтернатив режиму по умолчанию. Каждый режим обеспечивает разную степень надежности в случае отказа узла, и эта степень обратно пропорциональна скорости, т.е. чем надежнее режим WAL, тем он медленнее. Поэтому, если выбранный вариант использования не требует высокой надежности, можно переключиться на менее надежный режим.

Подробнее см. в пункте Изменение размера сегмента WAL.

10.3.5 Отключение WAL

Бывают ситуации, когда Отключение WAL может помочь повысить производительность.

10.3.6 Троттлинг записи страниц

РЕД КВАНТ периодически запускает процесс создания checkpointing, который записывает грязные страницы из памяти на диск. Грязная страница — это страница, которая была обновлена в оперативной памяти, но не была записана в соответствующий файл раздела (изменения были просто добавлены в WAL). Этот процесс происходит в фоновом режиме, не затрагивая логику приложения.

Однако, если грязная страница, запланированная для checkpointing, обновляется перед записью на диск, ее предыдущее состояние копируется в специальную область, называемую

буфером checkpointing. Если буфер переполнится, РЕД КВАНТ прекратит обработку всех обновлений до тех пор, пока контрольная точка не будет завершена. В результате производительность записи может упасть до нуля до завершения цикла checkpointing (рисунок 17).

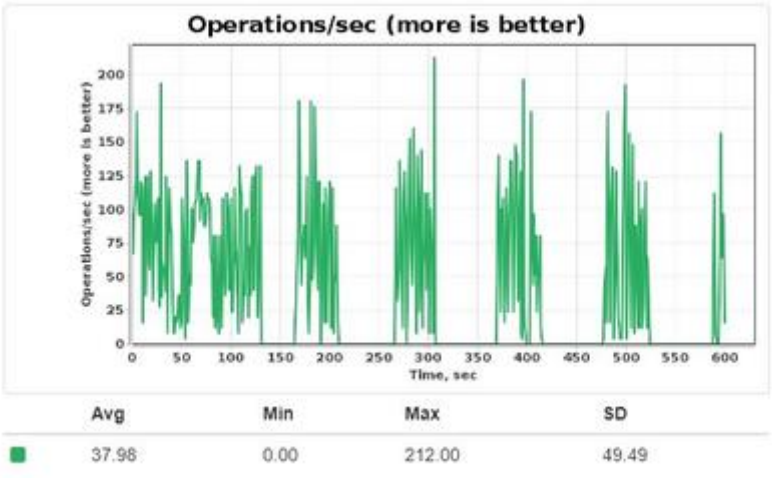

Рисунок 17 - Отслеживание производительности

Похожая ситуация возникает, если порог грязных страниц снова достигается во время выполнения контрольной точки. Это заставит РЕД КВАНТ запланировать еще одно выполнение checkpointing и остановить все операции обновления до завершения первого цикла checkpointing.

Обе ситуации обычно возникают, когда либо дисковое устройство работает медленно, либо частота обновления слишком велика. Чтобы смягчить и предотвратить эти падения производительности, можно рассмотреть возможность включения алгоритма регулирования записи страниц. Алгоритм снижает производительность операций обновления до скорости дискового устройства всякий раз, когда буфер checkpointing заполняется слишком быстро или быстро увеличивается процент грязных страниц.

В приведенном ниже примере показано, как включить регулирование записи:

```
IgniteConfiguration cfg = new IgniteConfiguration();

// Configuring Native Persistence.
DataStorageConfiguration storeCfg = new DataStorageConfiguration();

// Enabling the writes throttling.
storeCfg.setWriteThrottlingEnabled(true);

cfg.setDataStorageConfiguration(storeCfg);
// Starting the node.
Ignite ignite = Ignition.start(cfg);
```

10.3.7 НАСТРОЙКА РАЗМЕРА БУФЕРА CHECKPOINTING

Размер буфера checkpointing, описанный в пункте 102 Троттлинг записи страниц, является одним из триггеров процесса создания checkpointing.

Размер буфера по умолчанию рассчитывается как функция размера региона данных:

− для региона данных размером менее 1 ГБ размер буфера checkpointing по умолчанию минимальный (256 МБ);

− для региона данных размером от 1 ГБ до 8 ГБ — 4-кратный размер региона данных;

− для региона данных размером более 8 ГБ — размер буфера checkpointing 2 ГБ.

Размер буфера по умолчанию может быть неоптимальным для рабочих нагрузок с интенсивной записью, поскольку алгоритм регулирования записи страниц будет замедлять запись всякий раз, когда размер достигает критической отметки. Чтобы поддерживать производительность записи на нужном уровне во время выполнения checkpointing, необходимо рассмотреть возможность увеличения DataRegionConfiguration.checkpointPageBufferSize и включения регулирования записи для предотвращения падения производительности:

```
IgniteConfiguration cfg = new IgniteConfiguration();

// Configuring Native Persistence.
DataStorageConfiguration storeCfg = new DataStorageConfiguration();

// Enabling the writes throttling.
storeCfg.setWriteThrottlingEnabled(true);

// Increasing the buffer size to 1 GB.
storeCfg.getDefaultDataRegionConfiguration().setCheckpointPageBufferSize(1024L * 1024 * 1024);

cfg.setDataStorageConfiguration(storeCfg);

// Starting the node.
Ignite ignite = Ignition.start(cfg);
```

В приведенном выше примере размер буфера checkpointing региона по умолчанию установлен равным 1 ГБ.

10.3.8 Включение прямого ввода/вывода

Обычно всякий раз, когда приложение считывает данные с диска, ОС сначала получает данные и помещает их в кэш файлового буфера. Точно так же для каждой операции записи ОС сначала записывает данные в кэш, а затем передает их на диск. Чтобы исключить этот процесс, можно включить прямой ввод-вывод, и в этом случае данные считываются и записываются напрямую с/на диск, минуя кэш файлового буфера.

Модуль прямого ввода/вывода в РЕД КВАНТ используется для ускорения процесса создания checkpointing, который записывает грязные страницы из оперативной памяти на диск. Следует рассмотреть возможность использования подключаемого модуля прямого ввода-вывода для рабочих нагрузок с интенсивной записью.

Примечание: Прямой ввод-вывод нельзя включить специально для файлов WAL. Однако включение модуля прямого ввода/вывода дает небольшое преимущество в отношении файлов WAL: данные WAL не будут храниться в буферном кэше ОС слишком долго; они будут

сброшены (в зависимости от режима WAL) при следующем сканировании кэша страниц и удалены из кэша страниц.

Можно включить прямой ввод/вывод, переместить папку {IGNITE_HOME}/libs/optional/ignite-direct-io в папку libs/optional/ignite-direct-io верхнего уровня в дистрибутиве РЕД КВАНТ или в качестве зависимости Maven.

Можно использовать системное свойство IGNITE_DIRECT_IO_ENABLED, чтобы включить или отключить плагин во время выполнения.

10.3.9 Приобретение твердотельных накопителей производственного уровня

Следует отметить, что производительность native persistence РЕД КВАНТ может снизиться после нескольких часов интенсивной записи из-за особенностей проектирования и эксплуатации твердотельных накопителей. Необходимо задуматься о покупке быстрых твердотельных накопителей производственного уровня, чтобы поддерживать высокую производительность, или переключиться на устройства энергонезависимой памяти, такие как энергонезависимая память Intel Optane.

10.3.10 Резервное копирование SSD

Производительность произвольной записи на диске, заполненном на 50%, намного лучше, чем на диске, заполненном на 90%, из-за избыточного выделения SSD.

Необходимо задуматься о покупке твердотельных накопителей с более высокими показателями избыточного выделения ресурсов и убедиться, что производитель предоставляет инструменты для его настройки.

Примечание: Следует рассмотреть возможность использования дисков 3D XPoint вместо обычных твердотельных накопителей, чтобы избежать узких мест, вызванных низкой настройкой избыточного выделения ресурсов и постоянной сборкой мусора на уровне твердотельных накопителей.

10.4 Настройка производительности SQL

В данном пункте описаны базовые и расширенные методы оптимизации запросов РЕД КВАНТ SQL. Некоторые разделы также полезны для отладки и устранения неполадок.

10.4.1 Основные тезисы: РЕД КВАНТ и RDBMS

РЕД КВАНТ часто сравнивают с реляционными базами данных из-за их возможностей SQL, ожидая, что существующие SQL-запросы, созданные для СУБД, будут работать «из коробки» и выполняться быстрее в РЕД КВАНТ без каких-либо изменений. Обычно такое предположение основывается на том, что РЕД КВАНТ хранит и обрабатывает данные в памяти. Однако недостаточно просто поместить данные в оперативную память и ожидать увеличения производительности на порядок. Как правило, требуется дополнительная настройка. Ниже представлен стандартный перечень рекомендаций, которые следует учитывать, прежде чем сравнивать РЕД КВАНТ с СУБД или проводить какое-либо тестирование производительности:

- РЕД КВАНТ оптимизирован для многоузловых развертываний с оперативной памятью в качестве основного хранилища. Не стоит сравнивать одноузловой кластер РЕД КВАНТ с реляционной базой данных. Следует развернуть многоузловой кластер РЕД КВАНТ со всей копией данных в оперативной памяти.
- Стоит иметь ввиду, что потребуется скорректировать модель данных и существующие SQL-запросы. Следует использовать концепцию совместного размещения на этапе моделирования данных для правильного распределения данных. Не стоит забывать, что недостаточно просто поместить данные в оперативную память. Если данные размещены правильно, можно запускать SQL-запросы с JOIN в больших масштабах и ожидать значительного повышения производительности.
- Следует определить вторичные индексы и использовать другие стандартные и специфичные для РЕД КВАНТ методы настройки, описанные ниже.
- Стоит иметь в виду, что реляционные базы данных используют методы локального кэширования, и, в зависимости от общего размера данных, RDBMS может выполнять некоторые запросы даже быстрее, чем РЕД КВАНТ, даже в конфигурации с несколькими узлами. Если набор данных составляет около 10-100 ГБ, а RDBMS имеет достаточно оперативной памяти для локального кэширования данных, то она, например, может превзойти многоузловой кластер РЕД КВАНТ, поскольку последний будет использовать сеть. Чтобы увидеть разницу, стоит попробовать хранить гораздо больше данных в РЕД КВАНТ.

10.4.2 Использование оператора EXPLAIN

РЕД КВАНТ поддерживает оператор EXPLAIN, который можно использовать для чтения плана выполнения запроса. Следует использовать эту команду для анализа запросов на предмет возможной оптимизации. Стоит обратить внимание, что план будет содержать несколько строк: последняя будет содержать запрос для уменьшающей стороны (обычно приложения), остальные — для узлов карты (обычно узлов сервера).

```
EXPLAIN SELECT name FROM Person WHERE age = 26;
```

10.4.3 Оператор OR и селективность

Если запрос содержит оператор OR, индексы могут использоваться не так, как ожидалось, в зависимости от сложности запроса. Например, для запроса *select name from Person where gender='M' and (age = 20 or age = 30)*, будет использоваться индекс в поле gender вместо индекса в поле age, хотя последний является более селективным индексом. В качестве обходного пути для этой проблемы можно переписать запрос с помощью UNION ALL (стоит обратить внимание, что UNION без ALL будет возвращать строки DISTINCT, что изменит семантику запроса и еще больше снизит производительность запроса):

```
SELECT name FROM Person WHERE gender='M' and age = 20
UNION ALL
SELECT name FROM Person WHERE gender='M' and age = 30
```

10.4.4 Отказ от слишком большого числа столбцов

Стоит избегать слишком большого числа столбцов в результирующем наборе запроса SELECT. Из-за ограничений анализатора запросов H2 запросы с более чем 100 столбцами могут работать хуже, чем ожидалось.

10.4.5 Ленивая загрузка

По умолчанию РЕД КВАНТ пытается загрузить весь набор результатов в память и отправить его обратно инициатору запроса (обычно это приложение). Такой подход обеспечивает оптимальную производительность для запросов с небольшими или средними наборами результатов. Однако, если результирующий набор слишком велик для размещения в доступной памяти, это может привести к длительным паузам GC и даже к исключениям OutOfMemoryError.

Чтобы свести к минимуму потребление памяти за счет умеренного падения производительности, можно лениво загружать и обрабатывать наборы результатов, передавая параметр lazy в строки подключения JDBC и ODBC, или использовать аналогичный метод, доступный для API Java, .NET и C++. :

```
SqlFieldsQuery query = new SqlFieldsQuery("SELECT * FROM Person WHERE id > 10");

// Result set will be loaded lazily.
query.setLazy(true);
```

10.4.6 Запрос совмещенных данных

Когда РЕД КВАНТ выполняет распределенный запрос, он отправляет подзапросы на отдельные узлы кластера для получения данных и группирует результаты на reducer node (обычно это приложение). Если заранее известно, что запрашиваемые данные размещены совместно с условием GROUP BY, можно использовать SqlFieldsQuery.collocated = true, чтобы указать механизму SQL выполнять группировку на удаленных узлах. Это сократит сетевой трафик между узлами и время выполнения запросов. Когда для этого флага установлено значение true, запрос сначала выполняется на отдельных узлах, а результаты отправляются на reducer node для окончательного расчета.

В следующем примере предполагается, что данные размещены по department_id (другими словами, поле department_id настроено как affinity ключ).

```
SELECT SUM(salary) FROM Employee GROUP BY department_id
```

Из-за характера операции SUM РЕД КВАНТ суммирует зарплаты по элементам, хранящимся на отдельных узлах, а затем отправляет эти суммы на узел-редуктор, где будет рассчитан окончательный результат. Эта операция уже распределена, и включение флага colocated лишь немного улучшит производительность.

Ниже представлен немного другой пример:

```
SELECT AVG(salary) FROM Employee GROUP BY department_id
```

В этом примере РЕД КВАНТ должен извлечь все пары (salary, department_id) на reducer node и вычислить там результаты. Однако, если сотрудники находятся в одном месте по полю

department_id, т. е. данные о сотрудниках одного и того же отдела хранятся на одном узле, установка SqlFieldsQuery.collocated = true сократит время выполнения запроса, поскольку РЕД КВАНТ будет вычислять средние значения для каждого отдела на отдельных узлах и отправлять результаты на reducer node для окончательного расчета.

10.4.7 Принудительный порядок присоединения

Если этот флаг установлен, оптимизатор запросов не будет переупорядочивать таблицы в соединениях. Другими словами, порядок, в котором применяются объединения во время выполнения запроса, будет таким же, как указано в запросе. Без этого флага оптимизатор запросов может изменить порядок соединений для повышения производительности. Однако иногда он может принять неверное решение. Этот флаг помогает контролировать и явно указывать порядок соединений, не полагаясь на оптимизатор.

Есть следующий запрос:

```
SELECT * FROM Person p
JOIN Company c ON p.company = c.name where p.name = 'John Doe'
AND p.age > 20
AND p.id > 5000
AND p.id < 100000
AND c.name NOT LIKE 'O%';
```

Этот запрос содержит соединение между двумя таблицами: Person и Company. Чтобы получить наилучшую производительность, стоит понимать, какое соединение вернет наименьший результирующий набор. Таблица с меньшим размером результирующего набора должна быть указана первой в паре соединения. Чтобы получить размер каждого набора результатов, стоит протестировать каждую часть.

Q1:

```
SELECT count(*)
FROM Person p
where
p.name = 'John Doe'
AND p.age > 20
AND p.id > 5000
AND p.id < 100000;
```

Q2:

```
SELECT count(*)
FROM Company c
where
c.name NOT LIKE 'O%';
```

После выполнения Q1 и Q2 можно получить два разных результата:

Результат 1:

- Q1 — 30000;
- Q2 — 100000.

Q2 возвращает больше записей, чем Q1. В этом случае не нужно изменять исходный запрос, потому что меньшее подмножество уже находится в левой части соединения.

Результат 2:

- Q1 — 50000;
- Q2 — 10000.

Q1 возвращает больше записей, чем Q2. Поэтому нужно изменить первоначальный запрос следующим образом:

```
SELECT *
FROM Company c
JOIN Person p
ON p.company = c.name
where
p.name = 'John Doe'
AND p.age > 20
AND p.id > 5000
AND p.id < 100000
AND c.name NOT LIKE 'O%';
```

Подсказка порядка принудительного соединения может быть указана следующим образом:

- параметр подключения драйвера JDBC;
- атрибут подключения драйвера ODBC;
- если для выполнения SQL-запросов используется SqlFieldsQuery, можно установить подсказку принудительного порядка соединения, вызвав метод SqlFieldsQuery.setEnforceJoinOrder(true).

10.4.8 Увеличение размера inline индекса

Каждая запись в индексе имеет постоянный размер, который рассчитывается при создании индекса. Этот размер называется встроенным размером индекса. В идеале этого размера должно быть достаточно для хранения полной проиндексированной записи в сериализованной форме. Когда значения не полностью включены в индекс, РЕД КВАНТ может потребоваться выполнить чтение дополнительных страниц данных во время поиска по индексу, что может снизить производительность, если включена сохраняемость.

На рисунке 18 представлено, как значения хранятся в индексе.

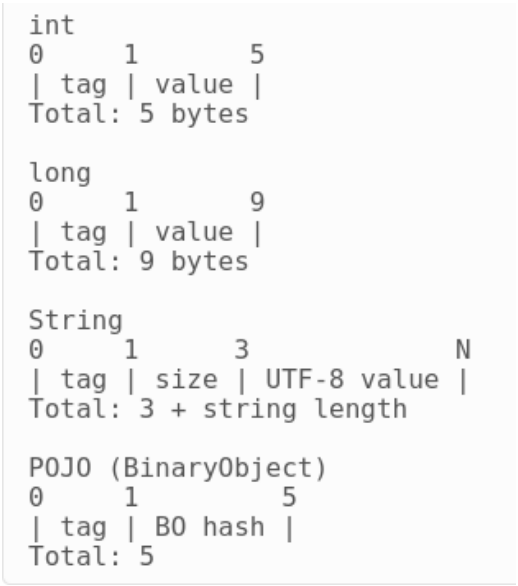

Рисунок 18 - Пример хранения значений в индексе

Для примитивных типов данных (bool, byte, short, int и т. д.) РЕД КВАНТ автоматически вычисляет inline размер индекса, чтобы значения включались полностью. Например, для полей int inline размер равен 5 (1 байт для тега и 4 байта для самого значения). Для полей long inline размер равен 9 (1 байт для тега + 8 байт для значения).

Для бинарных объектов индекс включает хэш каждого объекта, чего достаточно, чтобы избежать коллизий. Inline размер 5.

Для данных переменной длины индексы включают только первые несколько байтов значения. Поэтому при индексировании полей с данными переменной длины рекомендуется оценить длину значений поля и установить inline размер на значение, включающее большинство (около 95%) или все значения. Например, если у есть поле String, 95% значений которого содержат 10 или менее символов, можно установить inline размер индекса в этом поле равным 13.

Встроенные размеры, описанные выше, применяются к индексам с одним полем. Однако, когда индекс определяется для поля в объекте значения или для столбца, не являющегося первичным ключом, РЕД КВАНТ создает составной индекс, добавляя первичный ключ к индексированному значению. Поэтому при расчете встроенного размера для составных индексов нужно суммировать inline размер первичного ключа.

Ниже приведен пример расчета inline размера индекса для кэша, где и ключ, и значение являются сложными объектами.

```
public class Key {
    @QuerySqlField
    private long id;

    @QuerySqlField
    @AffinityKeyMapped
    private long affinityKey;
}

public class Value {
```

```
    @QuerySqlField(index = true)
    private long longField;

    @QuerySqlField(index = true)
    private int intField;

    @QuerySqlField(index = true)
    private String stringField; // we suppose that 95% of the values are 10 symbols
}
```

В таблице 16 приведены размеры inline индексов для индексов, определенных в приведенном выше примере.

Таблица 16 - Размеры встроенных индексов

Индекс	**Тип**	**Рекомендуемый встроенный размер**	**Комментарий**
(_key)	Индекс первичного ключа	5	Inline хэш бинарного объекта (5)
(affinityKey, _key)	Индекс affinity ключа	14	Inline long (9) + хэш бинарного объекта (5)
(longField, _key)	Дополнительный индекс	14	Inline long (9) + хэш бинарного объекта (5)
(intField, _key)	Дополнительный индекс	10	Inline int (5) + бинарный объект с точностью до хэша (5)
(stringField, _key)	Дополнительный индекс	18	Inline string (13) + хэш бинарного объекта (5) (при условии, что строка состоит из ~10 символов)

Следует обратить внимание, что нужно установить inline размер только для индекса в stringField. Для других индексов РЕД КВАНТ автоматически рассчитает inline размер.

Внимание: Поскольку РЕД КВАНТ кодирует строки в UTF-8, некоторые символы занимают более 1 байта.

10.4.9 Параллелизм запросов

По умолчанию SQL-запрос выполняется в одном потоке на каждом участвующем узле РЕД КВАНТ. Этот подход оптимален для запросов, возвращающих небольшие наборы результатов, включающие поиск по индексу. Например:

```
SELECT * FROM Person WHERE p.id = ?;
```

Для некоторых запросов может быть полезно выполняться в нескольких потоках. Это относится к запросам со сканированием таблиц и агрегированием, что часто имеет место для рабочих нагрузок HTAP и OLAP. Например:

```
SELECT SUM(salary) FROM Person;
```

Количество потоков, создаваемых на одном узле для выполнения запросов, настраивается для каждого кэша и по умолчанию равно 1. Изменить это значение можно, задав параметр CacheConfiguration.queryParallelism. Если таблицы SQL создаются с помощью команды CREATE TABLE, можно использовать шаблон кэша для установки этого параметра.

Если запрос содержит JOIN, то все участвующие кэши должны иметь одинаковую степень параллелизма.

10.4.10 Подсказки по индексам

Подсказки по индексам полезны в сценариях, когда известно, что один индекс больше подходит для определенных запросов, чем другой. Можно использовать их, чтобы указать оптимизатору запросов выбрать более эффективный план выполнения. Для этого можно использовать оператор USE INDEX(indexA,…,indexN), как показано в следующем примере.

```
SELECT * FROM Person USE INDEX(index_age)
WHERE salary > 150000 AND age < 35;
```

10.4.11 Обрезка раздела

Сокращение разделов — это метод, который оптимизирует запросы, использующие affinity ключи в условии WHERE. При выполнении такого запроса РЕД КВАНТ будет сканировать только те разделы, где хранятся запрошенные данные. Это сократит время запроса, поскольку запрос будет отправлен только тем узлам, которые хранят запрошенные разделы.

В следующем примере объекты сотрудников размещаются по полю id (если affinity ключ не задан явно, то в качестве affinity ключа используется первичный ключ):

```
CREATE TABLE employee (id BIGINT PRIMARY KEY, department_id INT, name VARCHAR)

/* This query is sent to the node where the requested key is stored */
SELECT * FROM employee WHERE id=10;

/* This query is sent to all nodes */
SELECT * FROM employee WHERE department_id=10;
```

В следующем примере ключ сходства задается явно и, следовательно, будет использоваться для совместного размещения данных и прямых запросов к узлам, которые хранят первичные копии данных:

```
CREATE TABLE employee (id BIGINT PRIMARY KEY, department_id INT, name VARCHAR) WITH "AFFINITY_KEY=department_id"

/* This query is sent to all nodes */
SELECT * FROM employee WHERE id=10;

/* This query is sent to the node where the requested key is stored */
SELECT * FROM employee WHERE department_id=10;
```

10.4.12 Пропуск редуктора при обновлении

Когда РЕД КВАНТ выполняет операцию DML, он сначала извлекает все затронутые промежуточные строки для анализа на reducer node (обычно приложение) и только затем подготавливает пакеты обновленных значений, которые будут отправлены на удаленные узлы.

Такой подход может повлиять на производительность и перегрузить сеть, если операция DML должна переместить много записей.

Стоит использовать этот флаг в качестве подсказки для механизма SQL, чтобы выполнить анализ всех промежуточных строк и обновления «на месте» на узлах сервера. Подсказка поддерживается для соединений JDBC и ODBC.

```
//jdbc connection string
jdbc:ignite:thin://192.168.0.15/skipReducerOnUpdate=true
```

10.4.13 Кэш строк SQL в heap

РЕД КВАНТ хранит данные и индексы в собственном пространстве памяти вне Java heap. Это означает, что при каждом доступе к данным часть данных будет копироваться из пространства off-heap в Java heap, потенциально десериализоваться и храниться в heap до тех пор, пока на нее ссылается приложение или серверный узел.

Кэш строк SQL on-heap предназначен для хранения «горячих» строк (объектов «ключ-значение») в Java heap, что позволяет минимизировать затраты ресурсов на копирование и десериализацию данных. Каждая кэшированная строка относится к записи в области off-heap и может быть признана недействительной в одном из следующих случаев:

- основная запись, хранящаяся в области off-heap, обновляется или удаляется;
- страница данных, на которой хранится основная запись, удаляется из оперативной памяти.

Кэш строк on-heap можно включить для определенного кэша/таблицы (если для создания таблиц и кэшей SQL используется CREATE TABLE, параметр можно передать через шаблон кэша):

```
<bean class="org.apache.ignite.configuration.IgniteConfiguration">
  <property name="cacheConfiguration">
    <bean class="org.apache.ignite.configuration.CacheConfiguration">
      <property name="name" value="myCache"/>
      <property name="sqlOnheapCacheEnabled" value="true"/>
    </bean>
  </property>
</bean>
```

Если кэш строк включен, можно обменять оперативную память на производительность. Можно увеличить производительность до 2 раз для некоторых SQL-запросов и вариантов использования, выделив больше оперативной памяти для целей кэширования строк.

Внимание! В настоящее время кэш не ограничен и может занимать столько оперативной памяти, сколько выделено для регионов данных памяти. Следует убедиться в том, что:

- установлен максимальный размер кучи JVM, равный общему размеру всех регионов данных, в которых хранятся кэши, для которых включен этот кэш строк on-heap.
- сборка мусора JVM настроена соответствующим образом.

10.4.14 Использование TIMESTAMP вместо DATE

По возможности следует использовать тип TIMESTAMP вместо DATE. В настоящее время тип DATE сериализуется/десериализуется очень неэффективно, что приводит к снижению производительности.

10.5 Настройка производительности пулов потоков

РЕД КВАНТ создает и поддерживает различные пулы потоков, которые используются для разных целей. В этом пункте перечислены некоторые из наиболее распространенных внутренних пулов и описано, как можно создать пользовательский пул.

10.5.1 Системный пул

Системный пул обрабатывает все операции, связанные с кэшем, за исключением SQL и некоторых других типов запросов, которые направляются в пул запросов. Кроме того, этот пул отвечает за обработку операций отмены вычислительных задач.

Размер пула по умолчанию — max(8, общее количество ядер). Чтобы изменить размер пула, нужно использовать IgniteConfiguration.setSystemThreadPoolSize(…) или аналогичный API из используемого языка программирования.

10.5.2 Пул запросов

Пул запросов орабатывает все запросы SQL, Scan и SPI, отправляемые и выполняемые в кластере.

Размер пула по умолчанию — max(8, общее количество ядер). Чтобы изменить размер пула, нужно использовать IgniteConfiguration.setQueryThreadPoolSize(…) или аналогичный API из используемого языка программирования.

10.5.3 Public пул

Public пул — это рабочая лошадка вычислительной сети. Все вычисления принимаются и обрабатываются этим пулом.

Размер пула по умолчанию — max(8, общее количество ядер). Чтобы изменить размер пула, нужно использовать IgniteConfiguration.setPublicThreadPoolSize(…) или аналогичный API из используемого языка программирования.

10.5.4 Сервисный пул

Вызовы Service Grid направляются в пул потоков служб. Наличие выделенных пулов для компонентов службы и вычислений позволяет избежать нехватки потоков и взаимоблокировок, когда реализация службы хочет вызвать вычисление или наоборот.

Размер пула по умолчанию — max(8, общее количество ядер). Чтобы изменить размер пула, нужно использовать IgniteConfiguration.setServiceThreadPoolSize(…) или аналогичный API из используемого языка программирования.

10.5.5 Чередующийся пул

Чередующийся пул помогает ускорить основные операции кэширования и транзакции, распределяя выполнение операций по нескольким полосам, которые не конкурируют друг с другом за ресурсы.

Размер пула по умолчанию — max(8, общее количество ядер). Чтобы изменить размер пула, нужно использовать IgniteConfiguration.setStripedPoolSize(…) или аналогичный API из используемого языка программирования.

10.5.6 Пул потоков данных

Пул потоков данных обрабатывает все сообщения и запросы, поступающие от IgniteDataStreamer и различных адаптеров потоковой передачи, которые используют IgniteDataStreamer для внутреннего использования.

Размер пула по умолчанию — max(8, общее количество ядер). Чтобы изменить размер пула, нужно использовать IgniteConfiguration.setDataStreamerThreadPoolSize(…) или аналогичный API из используемого языка программирования.

10.5.7 Snapshot пул

Пул моментальных снимков используется для обработки всех операций кластера, связанных с получением или восстановлением моментальных снимков РЕД КВАНТ.

Размер пула по умолчанию — 4 (см. IgniteConfiguration.DFLT_SNAPSHOT_THREAD_POOL_SIZE). Для изменения размера пула необходимо воспользоваться IgniteConfiguration.setSnapshotThreadPoolSize(…).

10.5.8 Создание пользовательского пула потоков

Можно настроить пользовательский пул потоков для вычислительных задач. Это полезно, если необходимо выполнять одну вычислительную задачу из другой, избегая взаимоблокировок. Чтобы гарантировать это, необходимо убедиться, что вложенная задача выполняется в пуле потоков, отдельном от пула потоков родительских задач.

Пользовательский пул определяется в IgniteConfiguration и должен иметь уникальное имя:

```
IgniteConfiguration cfg = new IgniteConfiguration();

cfg.setExecutorConfiguration(new ExecutorConfiguration("myPool").setSize(16));
```

Далее, например, необходимо выполнить следующую вычислительную задачу в потоке из определенного выше myPool:

```
public class InnerRunnable implements IgniteRunnable {
  @Override
```

```
    public void run() {
        System.out.println("Hello from inner runnable!");
    }
}
```

Для этого необходимо воспользоваться IgniteCompute.withExecutor(), который немедленно выполнит задачу из родительской задачи, как показано ниже:

```
public class OuterRunnable implements IgniteRunnable {
    @IgniteInstanceResource
    private Ignite ignite;

    @Override
    public void run() {
        // Synchronously execute InnerRunnable in a custom executor.
        ignite.compute().withExecutor("myPool").run(new InnerRunnable());
        System.out.println("outer runnable is executed");
    }
}
```

Выполнение родительской задачи может быть запущено следующим образом, и в этом сценарии она будет выполняться public пулом:

```
ignite.compute().run(new OuterRunnable());
```

Внимание: Если приложение попытается выполнить вычислительную задачу в пользовательском пуле, который не определен в конфигурации узла, в журналы будет записано специальное предупреждающее сообщение, и задача будет принята для выполнения public пулом.

10.6 Отладка и выявление проблем

В данном пункте рассматриваются некоторые общие советы и рекомендации по отладке и устранению неполадок при развертывании РЕД КВАНТ.

10.6.1 Средства отладки: Команда проверки согласованности

Утилита ./control.sh|bat включает набор команд проверки согласованности, которые помогают проверять инварианты непротиворечивости внутренних данных.

10.6.2 Файлы сохранения исчезают при перезапуске

В некоторых системах расположение файлов постоянного хранения РЕД КВАНТ по умолчанию может находиться во временной папке. Это может привести к ситуациям, когда файлы постоянного хранения удаляются операционной системой при каждом перезапуске процесса узла. Чтобы избежать этого:

- Необходимо убедиться, что для РЕД КВАНТ включен уровень ведения журнала WARN. Если файлы сохранения будут записаны во временный каталог, пользователю будет вывеедно предупреждение.

- Необходимо изменить расположение всех файлов сохраняемости с помощью API-интерфейсов DataStorageConfiguration, таких как setStoragePath(…), setWalPath(…) и setWalArchivePath(…).

10.6.3 Отладка проблем GC

В данном пункте содержится информация, которая может оказаться полезной при отладке и устранении неполадок, связанных с использованием Java heap или паузами GC.

10.6.3.1 Дампы heap

Если JVM генерирует исключения OutOfMemoryException, то автоматически выгружает heap при следующем возникновении исключения. Это помогает, если первопричина этого исключения не ясна и требуется более глубокий анализ состояния heap в момент сбоя:

```
-XX:+HeapDumpOnOutOfMemoryError
-XX:HeapDumpPath=/path/to/heapdump
-XX:OnOutOfMemoryError=“kill -9 %p”
-XX:+ExitOnOutOfMemoryError
```

10.6.3.2 Подробные журналы GC

Чтобы получить подробную информацию о действиях, связанных со сборщиком мусора, необходимо убедиться, что в настройках JVM узлов кластера настроены указанные ниже параметры:

```
-XX:+PrintGCDetails
-XX:+PrintGCTimeStamps
-XX:+PrintGCDateStamps
-XX:+UseGCLogFileRotation
-XX:NumberOfGCLogFiles=10
-XX:GCLogFileSize=100M
-Xloggc:/path/to/gc/logs/log.txt
```

Необходимо заменить /path/to/gc/logs/ фактическим путем в файловой системе.

Кроме того, для сборщика G1 необходимо установить свойство, указанное ниже. Оно предоставляет множество дополнительных сведений, которые намеренно не включены в параметр -XX:+PrintGCDetails:

```
-XX:+PrintAdaptiveSizePolicy
```

10.6.3.3 Анализ производительности с помощью Flight Recorder

В тех случаях, когда нужно отладить проблемы с производительностью или памятью, можно использовать Java Flight Recorder для непрерывного сбора низкоуровневой статистики времени выполнения, что позволяет анализировать инциденты постфактум. Чтобы включить Java Flight Recorder, нужно использовать следующие настройки:

```
-XX:+UnlockCommercialFeatures
-XX:+FlightRecorder
```

```
-XX:+UnlockDiagnosticVMOptions
-XX:+DebugNonSafepoints
```

Для начала записи состояния на конкретном узле РЕД КВАНТ, нужно воспользоваться следующей командой:

```
jcmd <PID> JFR.start name=<recordcing_name> duration=60s filename=/var/recording/recording.jfr settings=profile
```

10.6.3.4 Паузы JVM

Иногда можно увидеть предупреждающее сообщение о слишком долгой паузе JVM. Это может произойти, например, при массовой загрузке.

Настройка параметра тайм-аута IGNITE_JVM_PAUSE_DETECTOR_THRESHOLD может дать процессу время для завершения без генерации предупреждения. Можно установить порог через переменную среды или передать его как аргумент JVM (-DIGNITE_JVM_PAUSE_DETECTOR_THRESHOLD=5000) или как параметр для ignite.sh (-J-DIGNITE_JVM_PAUSE_DETECTOR_THRESHOLD=5000).

Значение выражено в миллисекундах.

10.7 Обработка исключений

В данном пункте представлены основные исключения, которые могут быть сгенерированы РЕД КВАНТ, и описано, как настроить и использовать обработчик критических сбоев.

10.7.1 Обработка исключений РЕД КВАНТ

Исключения, поддерживаемые РЕД КВАНТ API, и действия, которые можно предпринять в связи с этими исключениями, описаны в таблице 17.

Таблица 17 - Исключения, поддерживаемые РЕД КВАНТ API

Исключение	Описание	Действия	Исключение во время выполнения
CacheInvalidStateException	Выдается при попытке выполнить операцию с кэшем, в котором были потеряны некоторые разделы. В зависимости от политики потери разделов, настроенной для кэша, это исключение выдается либо при операциях чтения и/или записи. Подробности см. в	Сбросить потерянные разделы. Можно восстановить данные, вернув в кластер узлы, вызвавшие потерю раздела.	Да

Исключение	Описание	Действия	Исключение во время выполнения
	Политика потери раздела.		
IgniteException	Указывает на состояние ошибки в кластере.	Операция завершилась неудачей. Необходимо выйти из кластера.	Да
IgniteClientDisconnectedException	Выдается РЕД КВАНТ API, когда клиентский узел отключается от кластера. Выдается из операций кэширования, вычислительного API и структур данных.	Необходимо подождать и повторить попытку.	Да
IgniteAuthenticationException	Генерируется при сбое аутентификации узла или сбое аутентификации безопасности.	Операция завершилась неудачей. Необходимо выйти из метода.	Нет
IgniteClientException	Может вызываться из операций кэша.	Нужно проверить сообщение об исключении на предмет действия, которое необходимо предпринять.	Да
IgniteDeploymentException	Возникает, когда РЕД КВАНТ API не удается развернуть задание или задачу на узле. Выброшено из Compute API.	Операция завершилась неудачей. Необходимо выйти из метода.	Да
IgniteInterruptedException	Используется для переноса стандартного InterruptedException в IgniteException.	Необходимо повторить попытку после очистки флага прерывания.	Да
IgniteSpiException	Генерируется различными SPI (CollisionSpi, LoadBalancingSpi, TcpDiscoveryIpFinder, FailoverSpi, UriDeploymentSpi и т. д.)	Операция завершилась неудачей. Необходимо выйти из метода.	Да
IgniteSQLException	Генерируется при ошибке обработки SQL-запроса. Это исключение также предоставляет	Операция завершилась неудачей. Необходимо выйти из метода.	Да

Исключение	Описание	Действия	Исключение во время выполнения
	определенные коды ошибок запроса.		
IgniteAccessControlException	Выдается при сбое аутентификации / авторизации.	Операция завершилась неудачей. Необходимо выйти из метода.	Нет
IgniteCacheRestartingException	Вызывается из РЕД КВАНТ cache API, если кэш перезапускается.	Необходимо подождать и повторить попытку.	Да
IgniteFutureTimeoutException	Генерируется, когда истекает время ожидания будущих вычислений.	Необходимо либо увеличить лимит времени ожидания, либо выйти из метода.	Да
IgniteFutureCancelledException	Возникает, когда невозможно получить будущее вычисление, поскольку оно было отменено.	Повторить попытку.	Да
IgniteIllegalStateException	Указывает, что экземпляр РЕД КВАНТ находится в недопустимом состоянии для запрошенной операции.	Операция завершилась неудачей. Необходимо выйти из метода.	Да
IgniteNeedReconnectException	Указывает, что узел должен попытаться повторно подключиться к кластеру.	Воспользоваться логикой повторной попытки.	Нет
IgniteDataIntegrityViolationException	Выдается при обнаружении нарушения целостности данных.	Операция завершилась неудачей. Необходимо выйти из метода.	Да
IgniteOutOfMemoryException	Вызывается, когда в системе недостаточно памяти для обработки операций РЕД КВАНТ. Генерируется из операций кэширования.	Операция завершилась неудачей. Необходимо выйти из метода.	Да
IgniteTxOptimisticCheckedException	Вызывается, когда транзакция завершается неудачей.	Повторить попытку.	Нет
IgniteTxRollbackCheckedException	Возникает при автоматическом откате транзакции.	Повторить попытку.	Нет

Исключение	Описание	Действия	Исключение во время выполнения
IgniteTxTimeoutCheckedException	Возникает, когда время ожидания транзакции истекло.	Повторить попытку.	Нет
ClusterTopologyException	Указывает на ошибку в топологии кластера (например, сбой узла и т.д.). Выдается из Compute and Events API	Необходимо подождать и повторить попытку.	Да

10.7.2 Обработка критических сбоев

РЕД КВАНТ — надежная и отказоустойчивая система. Но в реальном мире возникают некоторые непредсказуемые проблемы, которые могут повлиять на состояние как отдельного узла, так и всего кластера. Такие проблемы можно обнаружить во время выполнения и соответствующим образом обработать с помощью предварительно настроенного обработчика критических сбоев.

10.7.2.1 Критические сбои

К критическим относятся следующие отказы:

- системные критические ошибки (например, OutOfMemoryError);
- непреднамеренное завершение работы системы (например, из-за необработанного исключения);
- зависли системные обработчики;
- сегментация узлов кластера.

Системная критическая ошибка — это ошибка, приводящая к неработоспособности системы. Например:

- Ошибки ввода-вывода файлов — обычно IOException вызывается операциями чтения/записи файла. Это возможно, когда включена встроенная персистентность РЕД КВАНТ (например, в случаях, когда не осталось места или при ошибке устройства), а также для режима в памяти, поскольку РЕД КВАНТ использует дисковое хранилище для хранения некоторых метаданных (например, в случаях, когда превышен лимит файловых дескрипторов или доступ к файлу запрещен).
- Ошибка нехватки памяти — когда системе управления памятью РЕД КВАНТ не удается выделить больше места (IgniteOutOfMemoryException).
- Ошибка нехватки памяти — когда на узле кластера заканчивается Java heap (OutOfMemoryError).

10.7.2.2 Обработка сбоев

Когда РЕД КВАНТ обнаруживает критический сбой, он обрабатывает его в соответствии с предварительно настроенным обработчиком сбоев. Обработчик сбоев можно настроить следующим образом:

```
IgniteConfiguration cfg = new IgniteConfiguration();
cfg.setFailureHandler(new StopNodeFailureHandler());
Ignite ignite = Ignition.start(cfg);
```

Необходимо включить поддержку следующих обработчиков сбоев:

- NoOpFailureHandler — Игнорирует любые сбои. Полезно для тестирования и отладки.
- RestartProcessFailureHandler - Особая реализация, которую можно использовать только с ignite.sh|bat. Процесс должен быть завершен с помощью метода Ignition.restart(true).
- StopNodeFailureHandler - Останавливает узел в случае критических ошибок, вызывая методы Ignition.stop(true) или Ignition.stop(nodeName, true).
- StopNodeOrHaltFailureHandler - Это обработчик по умолчанию, который пытается остановить узел. Если узел не может быть остановлен, то обработчик завершает процесс JVM.

10.7.2.3 Проверка работоспособности критически важных рабочих процессов

В РЕД КВАНТ есть ряд внутренних рабочих процессов, которые необходимы для правильной работы кластера. При выходе из строя одного из них узел может выйти из строя.

Следующие системные обработчики считаются критически важными:

- Discovery worker — обработка событий обнаружения.
- TCP communication worker - одноранговая связь между узлами.
- Exchange worker - обмен картами разделов.
- Обработчики чередующегося пула системы.
- Обработчики пула чередующихся потоков Data Streamer.
- Timeout worker — обработка таймаутов.
- Checkpoint thread — контрольные точки РЕД КВАНТ.
- WAL workers — ведение журнала с опережением записи, архивирование сегментов и сжатие.
- Expiration worker — срок действия на основе TTL.
- NIO workers — базовая сеть.

В РЕД КВАНТ есть внутренний механизм для проверки работоспособности критически важных рабочих процессов. Каждый рабочий процесс регулярно проверяется, чтобы убедиться, что он активен. Если рабочий процесс не активен и не обновляется, он считается

заблокированным, и РЕД КВАНТ выводит сообщение в файл журнала. Можно установить период бездействия через свойство IgniteConfiguration.systemWorkerBlockedTimeout.

Несмотря на то, что РЕД КВАНТ считает невосприимчивость системного работника критической ошибкой, он не обрабатывает эту ситуацию автоматически, кроме как выводит сообщения в файл журнала. Если необходимо включить конкретный обработчик сбоев для не отвечающих системных рабочих всех типов, нужно очистить свойство обработчика ignoredFailureTypes, как показано ниже:

```
StopNodeFailureHandler failureHandler = new StopNodeFailureHandler();
failureHandler.setIgnoredFailureTypes(Collections.EMPTY_SET);

IgniteConfiguration cfg = new IgniteConfiguration().setFailureHandler(failureHandler);

Ignite ignite = Ignition.start(cfg);
```

10.8 Бенчмарки

Тесты РЕД КВАНТ написаны на основе Yardstick Framework, что позволяет измерять производительность различных компонентов и модулей РЕД КВАНТ. В данном пункте описывается, как выполнять и настраивать предварительно собранные тесты. Если нужно добавить новые тесты или собрать существующий, стоит обратиться к инструкциям из файла РЕД КВАНТ DEVNOTES.txt в исходном каталоге.

10.8.1 Локальный запуск РЕД КВАНТ Benchmarks

Самый простой способ начать бенчмаркинг — использовать один из исполняемых скриптов, доступных в каталоге benchmarks/bin:

```
./bin/benchmark-run-all.sh config/benchmark-sample.properties
```

Приведенная выше команда будет сравнивать операции put кэша для распределенного атомарного кэша. Результаты теста будут добавлены в автоматически сгенерированный каталог output/results-{DATE-TIME}+\.

Если команду ./bin/benchmark-run-all.sh выполнить как есть, без каких-либо параметров и изменений в файле конфигурации, то все доступные бенчмарки будут выполняться на локальной машине с использованием конфигурации config/benchmark.properties. В случае возникновения каких-либо проблем следует обратиться к журналам, которые добавляются в автоматически сгенерированный каталог output/logs-{DATE-TIME}.

Дополнительные сведения о доступных тестах и параметрах конфигурации см. в пунктах Существующие контрольные показатели и Свойства и аргументы командной строки.

10.8.2 Удаленный запуск тестов РЕД КВАНТ

Чтобы сравнить РЕД КВАНТ с несколькими удаленными хостами:

1. Нужно перейти в config/ignite-remote-config.xml и заменить <value>127.0.0.1:47500..47509</value> фактическими IP-адресами всех удаленных хостов.

2. Нужно перейти в config/benchmark-remote-sample.properties и заменить localhost фактическими IP-адресами удаленных хостов в следующих местах: SERVERS=localhost,localhost DRIVERS=localhost,localhost, где DRIVER — это хост (обычно клиентский узел РЕД КВАНТ), который выполняет логику бенчмаркинга. SERVERS — это узлы РЕД КВАНТ, которые проходят бенчмаркинг. Нужно заменить вхождения localhost в тех же местах в файле config/benchmark-remote.properties, если планируется выполнение полного набора доступных тестов.
3. Нужно загрузить тесты РЕД КВАНТ Yardstick Benchmark на один из хостов DRIVERS в его собственном рабочем каталоге.
4. Нужно войти на удаленный хост, который будет DRIVER, и выполнить следующую команду:

```
./bin/benchmark-run-all.sh config/benchmark-remote-sample.properties
```

По умолчанию все необходимые файлы будут автоматически загружены с хоста, на котором запускается указанная выше команда, на каждый другой хост по тому же пути. Если предпочительнее сделать это вручную, нужно установить для переменной AUTO_COPY в файле свойств значение false.

Приведенная выше команда проведет сравнительный анализ операции cache put для распределенного атомарного кэша. Результаты теста будут добавлены в автоматически сгенерированный каталог output/results-{DATE-TIME}.

Если необходимо выполнить все доступные тесты на удаленных хостах, нужно выполнить следующую команду на стороне DRIVER:

```
./bin/benchmark-run-all.sh config/benchmark-remote.properties
```

10.8.3 СУЩЕСТВУЮЩИЕ КОНТРОЛЬНЫЕ ПОКАЗАТЕЛИ

По умолчанию предоставляются следующие бенчмарки:

1. GetBenchmark — тестирует операцию получения атомарного распределенного кэша.
2. PutBenchmark — тестирует операцию размещения атомарного распределенного кэша.
3. PutGetBenchmark — сопоставляет атомарный распределенный кэш с операциями put и get вместе.
4. PutTxBenchmark — тестирует операцию размещения транзакционного распределенного кэша.
5. PutGetTxBenchmark — сравнивает транзакционный распределенный кэш с операциями put и get вместе.
6. SqlQueryBenchmark — тестирует распределенный SQL-запрос по кэшированным данным.
7. SqlQueryJoinBenchmark — тестирует распределенный SQL-запрос с соединением по кэшированным данным.

8. SqlQueryPutBenchmark — тестирует распределенный SQL-запрос с одновременным обновлением кэша.
9. AffinityCallBenchmark — бенчмарки операций affinity-вызова.
10. ApplyBenchmark — бенчмарки применяют операцию.
11. BroadcastBenchmark — бенчмарки операций вещания.
12. ExecuteBenchmark — бенчмарки выполнения операции.
13. RunBenchmark — бенчмарки выполнения операций задач.
14. PutGetOffHeapBenchmark — тестирует атомарный распределенный кэш, объединяющий операции put и get из heap.
15. PutGetOffHeapValuesBenchmark — тестирует атомарный распределенный кэш, извлекая операции со значениями из heap.
16. PutOffHeapBenchmark — тесты атомарного распределенного кэша, помещающие операции off-heap.
17. PutOffHeapValuesBenchmark — тесты атомарного распределенного кэша, получающие операции со значениями off-heap.
18. PutTxOffHeapBenchmark — тестирует транзакционный распределенный кэш, выводящий операции off-heap.
19. PutTxOffHeapValuesBenchmark — тестирует транзакционный распределенный кэш, помещая значение off-heap.
20. SqlQueryOffHeapBenchmark — тесты распределенного SQL-запроса по кэшированным данным off-heap.
21. SqlQueryJoinOffHeapBenchmark — тестирует распределенный SQL-запрос с соединением по кэшированным данным off-heap.
22. SqlQueryPutOffHeapBenchmark — тестирует распределенный SQL-запрос с одновременным обновлением кэша off-heap.
23. PutAllBenchmark — тестирует операцию пакетного ввода атомарного распределенного кэша.
24. PutAllTxBenchmark — тестирует транзакционную операцию пакетного ввода распределенного кэша.

10.8.4 Свойства и аргументы командной строки

Стоит обратить внимание, что в данном пункте описываются только параметры конфигурации, относящиеся к тестам РЕД КВАНТ, а не к платформе Yardstick. Чтобы запустить тесты РЕД КВАНТ и построить графики, нужно будет запустить их с помощью сценариев платформы Yardstick в папке bin.

В конфигурации теста могут быть определены следующие свойства теста РЕД КВАНТ:

- -b <num> или --backups <num> — количество резервных копий для каждого ключа;
- -cfg <path> или --Config <path> — путь к файлу конфигурации РЕД КВАНТ;
- -cs или --cacheStore — включить или отключить хранилище кэша readThrough, writeThrough;
- -cl или --client - флаг клиента. Необходимо использовать этот флаг, если запущено более одного DRIVER, иначе дополнительные драйверы будут вести себя как серверы;
- -nc или --nearCache — флаг Near Cache;
- -nn <num> или --nodeNumber <num> — количество узлов (автоматически устанавливается в файле Benchmark.properties); используется для ожидания запуска указанного количества узлов;
- -sm <mode> или -syncMode <mode> — режим синхронизации (определяется в CacheWriteSynchronizationMode`);
- -r <num> или --range — диапазон ключей, который генерируется случайным образом для операций кэширования;
- -rd или --restartdelay — задержка перезапуска в секундах;
- -rs или --restartsleep — перезапуск спящего режима через несколько секунд;
- -rth <host> или --restHost <host> — хост REST TCP;
- -rtp <num> или --restPort <num> — TCP-порт REST, указывает, что узел РЕД КВАНТ готов к обработке клиентов РЕД КВАНТ;
- -ss или --syncSend — флаг, указывающий, используется ли синхронная отправка в TcpCommunicationSpi;
- -txc <value> или --txConcurrency <value> — управление параллелизмом транзакций в кэше, либо OPTIMISTIC, либо PESSIMISTIC (определено в CacheTxConcurrency);
- -txi <value> или --txIsolation <value> — изоляция транзакций кэша (определена в CacheTxIsolation);
- -wb или --writeBehind — включить или отключить запись Behind для хранилища кэша.

Например, если необходимо запустить 2 сервера IgniteNode на локальном хосте с тестом PutBenchmark, числом резервных копий, равным 1, и режимом синхронизации, установленным на PRIMARY_SYNC, то в файле Benchmark.properties необходимо указать следующую конфигурацию:

```
SERVER_HOSTS=localhost,localhost
...

# Note that -dn and -sn, which stand for data node and server node,
# are native Yardstick parameters and are documented in
# Yardstick framework.
CONFIGS="-b 1 -sm PRIMARY_SYNC -dn PutBenchmark`IgniteNode"
```

10.8.5 Создание из источников

Необходимо запустить ./mvnw clean package -Pyardstick -pl modules/yardstick -am -DskipTests в корневом каталоге РЕД КВАНТ.

Эта команда скомпилирует проект, а также распакует скрипты из файла yardstick-resources.zip в каталог modules/yardstick/target/assembly/bin.

Артефакты можно найти в каталоге modules/yardstick/target/assembly.

10.8.6 Пользовательские тесты РЕД КВАНТ

Все тесты расширяют класс AbstractBenchmark. Новый тест также должен расширить этот абстрактный класс и реализовать метод тестирования (это метод, который фактически проверяет производительность).

ЛИСТ РЕГИСТРАЦИИ ИЗМЕНЕНИЙ

№ версии	*Дата изменения*	*Автор*	*Состав изменений*
1.0	15.03.2023	Антонова Ю.В.	Первоначальная версия документа.