**Causality data nodes.**

**Метод dataNodes(causalityToken, zoneId)** возвращает значение data nodes для зоны актуальное для заданного causalityToken. Ревизия (токен) с которой было записано значения data nodes больше ревизии события, которое вызвало пересчет data nodes. Поэтому в случае если scaleUp/scaleDown таймеры immediate, то нужно ждать когда data nodes обновятся в метасторадже.

**Ревизия для которых нужно ждать обновление data nodes в метасторадже.**:

1. Ревизия изменения топологии - ревизия с которой был записан zonesLogicalTopologyPrefix
2. Изменение конфигурации зоны:
   1. создание зоны - ревизия создания конфигурации
   2. удаление зоны - ревизия удаления конфигурации
   3. изменение значения scaleUp - ревизия изменения конфигурации в котором изменилось значение scaleUp
   4. изменение значения scaleDown - ревизия изменения конфигурации в котором изменилось значение scaleDown
   5. изменение фильтра - ревизия изменения конфигурации в котором изменилось значение фильтра

Если запрошены data nodes с токеном на котором произошло такое событие, то метод dataNodes вернёт data nodes записанные в метасторадж после обработки исходного события с применением фильтра зоны актуальным на заданный токен.

**Алгоритм**

работы метода dataNodes(causalityToken, zoneId):

1. Читаем из конфигурацию зоны актуальную на заданный causalityToken.
2. Определяем нужно ли ждать обновления data nodes в метасторадже.
   1. Если scaleUp=immediate и scaleDown=immediate то
      1. Определяем ревизию последнего scaleUp события после которого нужно ждать изменения data nodes в metastorage. (Список таких событий ниже, под алгоритмом). Аналогично определяем ревизию последнего scaleDown события. Максимальное значение ревизий ограничено causalityToken’ом. Назовём эти ревизии scaleUpTopologyRevision и scaleDownTopologyRevision.
      2. Ждём когда значение для ключа zoneScaleUpChangeTriggerKey станет больше или равно scaleUpTopologyRevision.
      3. Ждём когда значение для ключа zoneScaleDownChangeTriggerKey станет больше или равно scaleDownTopologyRevision.
      4. Получим две записи из метастораджа: первая с zoneScaleUpChangeTriggerKey, вторая с zoneScaleDownChangeTriggerKey.
      5. Из двух записей получаем ревизии на которых они были записаны. Выбираем максимальную из них.
      6. По этой ревизии читаем из метастораджа data nodes, так как data nodes записывается в одном инвоуке вместе с zoneScaleUpChangeTriggerKey/zoneScaleDownChangeTriggerKey.
      7. К полученным data nodes применяем фильтр из конфигурации полученной на шаге 1.
      8. Возвращаем из метода отфильтрованные data nodes.
   2. Если в конфигурации только scaleUp=immediate.
      1. Определяем ревизию последнего scaleUp события после которого нужно ждать изменения data nodes в metastorage. (Список таких событий ниже, под алгоритмом). Максимальное значение ревизии ограничено causalityToken’ом. Назовём эту ревизию scaleUpTopologyRevision.
      2. Ждём когда значение для ключа zoneScaleUpChangeTriggerKey станет больше или равно scaleUpTopologyRevision.
      3. Берем ревизию события на котором был изменен zoneScaleUpChangeTriggerKey и получаем по этой ревизии data nodes из метастораджа.
      4. К полученным data nodes применяем фильтр из конфигурации полученной на шаге 1.
      5. Возвращаем из метода отфильтрованные data nodes.
   3. Если в конфигурации только scaleDown=immediate.
      1. Определяем ревизию последнего scaleDown события после которого нужно ждать изменения data nodes в metastorage. (Список таких событий ниже, под алгоритмом). Максимальное значение ревизии ограничено causalityToken’ом. Назовём эту ревизию scaleDownTopologyRevision.
      2. Ждём когда zoneScaleDownChangeTriggerKey станет больше или равен scaleDownTopologyRevision.
      3. Берем ревизию события на котором был изменен zoneScaleDownChangeTriggerKey и получаем по этой ревизии data nodes из метастораджа.
      4. К полученным data nodes применяем фильтр из конфигурации полученной на шаге 1.
      5. Возвращаем из метода отфильтрованные data nodes.
   4. Если если значение обоих таймеров не immediate:
      1. Читаем из метастораджа data nodes актуальный на causalityToken.
      2. К полученным data nodes применяем фильтр из конфигурации полученной на шаге 1.

**Детали алгоритма**

**Хранилище изменений конфигурации**

Для работы алгоритма при создании/изменении/удалении зоны нужно сохранять конфигурацию зоны в версионированном хранилище. Для этого в лисенерах конфигураций зон нужно добавить сохранение конфигурации в in memory структуру, например в ConcurrentSkipListMap. При каждом обновлении нужно сохранять эту структуру в vault, для того чтобы во время рестарта восстанавливать её.

**Ревизии событий которые триггерят обновление data nodes**

Если scaleUp/scaleDown=immediate, то нужно определить последнее событие которое триггерит scaleUp/scaleDown.

Список событий для scaleUp после которых нужно ждать изменения data nodes в metastorage:

1. Изменение топологии при котором добавились ноды.
2. Создание зоны
3. Удаление зоны
4. Изменение значения scaleUp на immediate
5. Изменение фильтра

Список событий для scaleDown после которых нужно ждать изменения data nodes в metastorage:

1. Изменение топологии при котором удалились ноды.
2. Создание зоны
3. Удаление зоны
4. Изменение значения scaleDown на immediate

**Алгоритм определения последнего события каждого типа и его ревизии:**

1. Изменение топологии при котором добавились и удалились ноды.
   1. С помощью метода MetaStorageManager#get(ByteArray key, long revUpperBound) вычитываем топологии вызывая его последовательно начиная с revUpperBound=causalityToken.
   2. Прочитав последнюю запись (entry) актуальную на causalityToken повторно вызываем метод get(key, revUpperBound) c revUpperBound = entry.revision() - 1. Каждая прочитанная топология сравнивается с предыдущей.
   3. И так далее пока не будет найдена последняя топология с добавленной нодой и последняя топология с удаленной нодой.
   4. Читаем ревизию этих событий.
2. Создание зоны.
   1. Достаём первую запись из хранилища изменений конфигурации. Берём ревизию этого события.
3. Удаление зоны
   1. Достаём последнюю запись из хранилища изменений конфигурации. Если это запись о удалениие зоны, то берём ревизию этого события.
4. Изменение значения scaleUp/scaleDown на immediate
   1. Из хранилища изменений конфигурации достаём последнюю запись в которой изменялся scaleUp/scaleDown на immediate. Если такие записи есть, то берём ревизию этого события. Если записи нет, значит scaleUp/scaleDown не изменялись.
5. Изменение фильтра
   1. Из хранилища изменений конфигурации достаём последнюю запись в которой изменялся фильтр. Если такая запись есть, то берём ревизию этого события. Если записи нет, значит фильтр не изменялся.

**Рескедулинг таск**

Рассмотрим пример в котором есть immediate таймер. При изменении топологии в executor помещается таска на пересчет data nodes. Несмотря на то что таймер immediate таска начинает выполняться не сразу и может быть отменена.

**Пример**:

1. dataNodes=[A], topology=[A], scaleUp=immediate
2. Изменилась топология: [A, B], ревизия 10. Заскедулилась scaleUp таска на добавление ноды B.
3. Происходит вызов метода dataNodes с ревизией 20.
   1. Читаем конфигурацию зоны актуальную на ревизию 20: scaleUp=immediate.
   2. Читаем из метастораджа последнюю топологию в которой добавились ноды, определяем её ревизию: 10.
   3. Ждём когда zoneScaleUpChangeTriggerKey в метасторадже станет равен 10 (ревизия последнего изменения топологии в которой добавилась нода B).
   4. Когда получим событие об изменении zoneScaleUpChangeTriggerKey то узнаем ревизию этого события. С этой ревизией можно прочитать значение data nodes из метастораджа.
4. Изменилась топология: [A, B, C], ревизия 30. Зарескедулилась scaleUp таска на добавление нод [B, C].
5. Несмотря на то что scaleUp=immediate таска на добавление в data nodes ноды B так и не успела начать свою работу (например потому что executor был загружен тасками для других зон).
6. Произошло изменение значения scaleUp на 1000. Ревизия 40.
7. В дальнейшем изменений топологии не было. Через 1000 секунд data nodes изменятся на [A, B, C]

Как в таком кейсе как закомплитить фьючу на получение data nodes с ревизией 20? Эта фьюча будет ждать что значение zoneScaleUpChangeTriggerKey изменится на 10, но этого изменения не будет. Вместо этого оно обновится на 40, причем через 1000 секунд.

Способ решить эту проблему: не отменять таски переданные в executor, если они были созданы с immediate таймером. Такие таски должны гарантированно исполниться, в том порядке в каком они были добавлены в executor.

**Использование ScheduledExecutorService#executor’а**

События которые создают таски в executor

1. Добавление ноды в топологию
2. Удаление ноды из топологии.
3. Изменение значения scaleUp в конфигурации зоны.
4. Изменение значения scaleDown в конфигурации зоны.

События которые вызывают методы saveDataNodesToMetaStorageOnScaleUp и saveDataNodesToMetaStorageOnScaleDown

1. Обновление фильтра.

Так как обновления фильтра не создаёт таску в executor’е возможен такой сценарий

1. dataNodes=[A], topology=[A], scaleUp=immediate
2. Изменилась топология: [A, B], ревизия 10. Заскедулилась scaleUp таска на добавление ноды B.
3. Происходит вызов метода dataNodes с ревизией 20.
   1. Читаем конфигурацию зоны актуальную на ревизию 20: scaleUp=immediate.
   2. Читаем из метастораджа последнюю топологию в которой добавились ноды, определяем её ревизию: 10.
   3. Ждём когда zoneScaleUpChangeTriggerKey в метасторадже станет равен 10 (ревизия последнего изменения топологии в которой добавилась нода B).
   4. Когда получим событие об изменении zoneScaleUpChangeTriggerKey то узнаем ревизию этого события. С этой ревизией можно прочитать значение data nodes из метастораджа.
4. На ревизии 30 изменился фильтр зоны. В обработке этого эвента сразу будет вызван метод saveDataNodesToMetaStorageOnScaleUp
5. saveDataNodesToMetaStorageOnScaleUp вызванный изменением фильтра может отработать быстрее чем scaleUp таска в executor’е (например потому что executor был загружен тасками для других зон).
6. В результате когда сфайрится таска на scaleUp в executor’е, то она не обновит data nodes в метасторадже, так как её ревизия будет меньше чем zoneScaleUpChangeTriggerKey.

Этот кейс можно решить если при изменении фильтра создавать таску в executor, вместо того чтобы сразу вызывать метод saveDataNodesToMetaStorageOnScaleUp. Для каждой зоны нужно выполнять таски на обновления data nodes, стриггеренные immediate таймерами, последовательно в порядке их создания.

**UPD.** Изменение фильтра будет как и раньше вызывать метод saveDataNodesToMetaStorageOnScaleUp без создания таски в executor. При ожидании нужного значения для ключа zoneScaleUpChangeTriggerKey/zoneScaleDownChangeTriggerKey нужно ожадать, что оно будет не строго равно, а больше или равно.

**Дожидание когда значение для ключа zoneScale(Up/Down)ChangeTriggerKey станет равно scale(Up/Down)TopologyRevision**

Если scaleUp/scaleDown immediate, то определив последнее событие которое запустило изменение data nodes нужно дождаться что в метасторадже появится запись с ключём zoneScale(Up/Down)ChangeTriggerKey и значением равным ревизии события, которое запустило изменение data nodes.

Для этого нужно:

1. Зарегистрировать watch listener’ы которые будут слушать zoneScale(Up/Down)ChangeTriggerKey.
2. При регистрации listener’а в качестве стартовой ревизии с которой нужно слушать события нужно указать ревизию события которое стриггерило изменение data nodes.
3. Когда нужная запись найдена, то listener можно разрегистрировать.