Как мы делали высокодоступную платформу или практические приемы использования in-memory grids

Андрей Ершов

О чем доклад?

- Что такое in-memory grid?
- Как in-memory grid помогает нам в разработке
- С какими сложностями мы сталкиваемся при использовании in-memory grid
- Будет много кода

Что мы разрабатываем?

- Сервис аудио-конференций
- … поверх собственной телефонной платформы
- … больших (>1000 человек) конференций
- … горизонтально масштабируемый
- … высокодоступный
- … распределенный по нескольким ЦОДам

Зачем нам In-Memory Grid?

- Сервис аудио-конференций
- … поверх собственной телефонной платформы
- … больших (>1000 человек) конференций
- … горизонтально масштабируемый
- … высокодоступный
- … распределенный по нескольким ЦОДам

In-Memory Grid 101 (0/4). Map

- Map aka dictionary
- Быстрый поиск значения по ключу
- Существуют различные реализации
  - Однопотоковые, потоко-безопасные, конкурентные
  - Для объектов/для примитивов
  - Основанные на хэш-таблицах или бинарных деревьях

In-Memory Grid 101 (1/4). Локальный кэш

- Локальный кэш это map, хранящийся в памяти процесса
- Может иметь несколько уровней
- Поддерживает различные политики вытеснения

In-Memory Grid 101 (2/4). Удаленный кэш

- То же, что и локальный кэш, но данные хранятся в отдельном процессе
- Процесс предоставляет API для доступа к данным

In-Memory Grid 101 (3/4). Распределенный кэш

- То же, что и удаленный кэш, но данные хранятся в нескольких процессах
- Шардирование
- Избыточность

In-Memory Grid 101 (4/4). In-memory Grid

- Расределенный кэш с дополнительными возможностями:
  - EntryProcessors
  - Tasks
  - Topology change events
  - Continuous queries
  - SQL-like queries
  - Transactions support

Что используем мы?

Стартапы используют

Задача №1

Количество участников в конференции

Задача №1. Формулировка задачи

- В конференции участвуют сессии с нескольких серверов
- Информация о конференции хранится в GridGain
- Нужно знать, когда выходит последний участник, чтобы удалить конференцию

Попытка 1. Объект Conference

```
class Conference {
    int participants;

    void addParticipant(){
        participants++;
    }

    boolean removeParticipant(){
        participants--;
        return participants == 0;
    }
}
```

Попытка 1. Объект Conference

```
class Conference {
    int participants;

    void addParticipant(){
        participants++;
    }

    boolean removeParticipant(){
        participants--;
        return participants == 0;
    }
}
```

Попытка 1. Объект Conference

```
class Conference {
    int participants;

    void addParticipant(){
        participants++;
    }

    boolean removeParticipant(){
        participants--;
        return participants == 0;
    }
}
```

Попытка 1. Объект Conference

```
class Conference {
    int participants;

    void addParticipant(){
        participants++;
    }

    boolean removeParticipant(){
        participants--;
        return participants == 0;
    }
}
```

Попытка 1. ConferenceService

```
interface ConferenceService {
    void addParticipant(String conferenceId);
    void removeParticipant(String conferenceId);
}
```

Попытка 1. ConferenceService

```
interface ConferenceService {
    void addParticipant(String conferenceId);
    void removeParticipant(String conferenceId);
}
```

Попытка 1. ConferenceService

```
interface ConferenceService {
    void addParticipant(String conferenceId);
    void removeParticipant(String conferenceId);
}
```

Попытка 1. Add participant

```
void addParticipant(String conferenceId){
    Conference conference = conferenceCache.get(conferenceId);
    if (conference == null) conference = new Conference();
    conference.addParticipant();
    conferenceCache.put(conferenceId, conference);
}
```

Попытка 1. Add participant

```
void addParticipant(String conferenceId){
    Conference conference = conferenceCache.get(conferenceId);
    if (conference == null) conference = new Conference();
    conference.addParticipant();
    conferenceCache.put(conferenceId, conference);
}
```

Попытка 1. Add participant

```
void addParticipant(String conferenceId){
    Conference conference = conferenceCache.get(conferenceId);
    if (conference == null) conference = new Conference();
    conference.addParticipant();
    conferenceCache.put(conferenceId, conference);
}
```

Попытка 1. Add participant

```
void addParticipant(String conferenceId){
    Conference conference = conferenceCache.get(conferenceId);
    if (conference == null) conference = new Conference();
    conference.addParticipant();
    conferenceCache.put(conferenceId, conference);
}
```

Попытка 1. Add participant

```
void addParticipant(String conferenceId){
    Conference conference = conferenceCache.get(conferenceId);
    if (conference == null) conference = new Conference();
    conference.addParticipant();
    conferenceCache.put(conferenceId, conference);
}
```

Попытка 1. Remove participant

```
void removeParticipant(String conferenceId){
    Conference conference = conferenceCache.get(conferenceId);
    if (conference.removeParticipant()) {
        conferenceCache.remove(conferenceId);
    } else {
        conferenceCache.put(conferenceId, conference);
    }
}
```

Попытка 1. Remove participant

```
public void removeParticipant(String conferenceId){
    Conference conference = conferenceCache.get(conferenceId);
    if (conference.removeParticipant()) {
        conferenceCache.remove(conferenceId);
    } else {
        conferenceCache.put(conferenceId, conference);
    }
}
```

Попытка 1. Remove participant

```
public void removeParticipant(String conferenceId){
    Conference conference = conferenceCache.get(conferenceId);
    if (conference.removeParticipant()) {
        conferenceCache.remove(conferenceId);
    } else {
        conferenceCache.put(conferenceId, conference);
    }
}
```

Попытка 1. Remove participant

```
public void removeParticipant(String conferenceId){
    Conference conference = conferenceCache.get(conferenceId);
    if (conference.removeParticipant()) {
        conferenceCache.remove(conferenceId);
    } else {
        conferenceCache.put(conferenceId, conference);
    }
}
```

Попытка 2. Eliminate race condition - replace

```
void addParticipant(String conferenceId) {
    Conference oldC, newC;
    do {
        oldC = conferenceCache.get(conferenceId);
        if (oldC != null) {
            newC = new Conference(oldC);
        } else {
            newC = new Conference();
        }
        newC.addParticipant();
    } while (!conferenceCache.replace
                (conferenceId, oldC, newC));
}
```

Попытка 2. Eliminate race condition - replace

```
void addParticipant(String conferenceId) {
    Conference oldC, newC;
    do {
        oldC = conferenceCache.get(conferenceId);
        if (oldC != null) {
            newC = new Conference(oldC);
        } else {
            newC = new Conference();
        }
        newC.addParticipant();
    } while (!conferenceCache.replace
                (conferenceId, oldC, newC));
}
```

Попытка 2. Eliminate race condition - replace

```
void addParticipant(String conferenceId) {
    Conference oldC, newC;
    do {
        oldC = conferenceCache.get(conferenceId);
        if (oldC != null) {
            newC = new Conference(oldC);
        } else {
            newC = new Conference();
        }
        newC.addParticipant();
    } while (!conferenceCache.replace
                (conferenceId, oldC, newC));
}
```

Попытка 2. Eliminate race condition - replace

```
void addParticipant(String conferenceId) {
    Conference oldC, newC;
    do {
        oldC = conferenceCache.get(conferenceId);
        if (oldC != null) {
            newC = new Conference(oldC);
        } else {
            newC = new Conference();
        }
        newC.addParticipant();
    } while (!conferenceCache.replace
                (conferenceId, oldC, newC));
}
```

Попытка 2. Eliminate race condition - replace

```
void addParticipant(String conferenceId) {
    Conference oldC, newC;
    do {
        oldC = conferenceCache.get(conferenceId);
        if (oldC != null) {
            newC = new Conference(oldC);
        } else {
            newC = new Conference();
        }
        newC.addParticipant();
    } while (!conferenceCache.replace
                (conferenceId, oldC, newC));
}
```

Попытка 2. Eliminate race condition - replace

```
void addParticipant(String conferenceId) {
    Conference oldC, newC;
    do {
        oldC = conferenceCache.get(conferenceId);
        if (oldC != null) {
            newC = new Conference(oldC);
        } else {
            newC = new Conference();
        }
        newC.addParticipant();
    } while (!conferenceCache.replace
                (conferenceId, oldC, newC));
}
```

Попытка 2. Eliminate race condition - replace

```
public void removeParticipant(String conferenceId) {
    Conference oldC, newC;
    do {
        oldC = conferenceCache.get(conferenceId);
        newC = new Conference(oldC);
        if (newC.removeParticipant()) {
            newC = null;
        }
    } while (!conferenceCache.replace
                (conferenceId, oldC, newC));
}
```

Попытка 3. Eliminate RT - EntryProcessor

```
interface EntryProcessor<K, V, R> {
    R process(MutableEntry<K, V> entry);
}

interface MutableEntry<K, V> {
    V getValue();
    void setValue(V newValue);
    …
}
```

Попытка 3. Eliminate RT - EntryProcessor

```
interface EntryProcessor<K, V, R> {
    R process(MutableEntry<K, V> entry);
}

interface MutableEntry<K, V> {
    V getValue();
    void setValue(V newValue);
    …
}
```

Попытка 3. AddParticipantProcessor

```
class AddParticipantProcessor implements EntryProcessor{

    public Conference process(MutableEntry entry){
        Conference conference = entry.getValue();
        if (conference == null)
          conference = new Conference();
        conference.addParticipant();
        entry.setValue(conference);
        return conference;
    }
}
```

Попытка 3. AddParticipantProcessor

```
class AddParticipantProcessor implements EntryProcessor{

    public Conference process(MutableEntry entry){
        Conference conference = entry.getValue();
        if (conference == null)
          conference = new Conference();
        conference.addParticipant();
        entry.setValue(conference);
        return conference;
    }
}
```

Попытка 3. AddParticipantProcessor

```
class AddParticipantProcessor implements EntryProcessor{

    public Conference process(MutableEntry entry){
        Conference conference = entry.getValue();
        if (conference == null)
          conference = new Conference();
        conference.addParticipant();
        entry.setValue(conference);
        return conference;
    }
}
```

Попытка 3. AddParticipantProcessor

```
class AddParticipantProcessor implements EntryProcessor{

    public Conference process(MutableEntry entry){
        Conference conference = entry.getValue();
        if (conference == null)
          conference = new Conference();
        conference.addParticipant();
        entry.setValue(conference);
        return conference;
    }
}
```

Попытка 3. ConferenceService

```
class ConferenceService {
    Cache<String, Conference> conferenceCache;

    void addParticipant(String conferenceId){
        conferenceCache.invoke(conferenceId,
          new AddParticipantProcessor());
    }

    void removeParticipant(String conferenceId){
        conferenceCache.invoke(conferenceId,
          new RemoveParticipantProcessor());
    }
}
```

Попытка 3. ConferenceService

```
class ConferenceService {
    Cache<String, Conference> conferenceCache;

    void addParticipant(String conferenceId){
        conferenceCache.invoke(conferenceId,
          new AddParticipantProcessor());
    }

    void removeParticipant(String conferenceId){
        conferenceCache.invoke(conferenceId,
          new RemoveParticipantProcessor());
    }
}
```

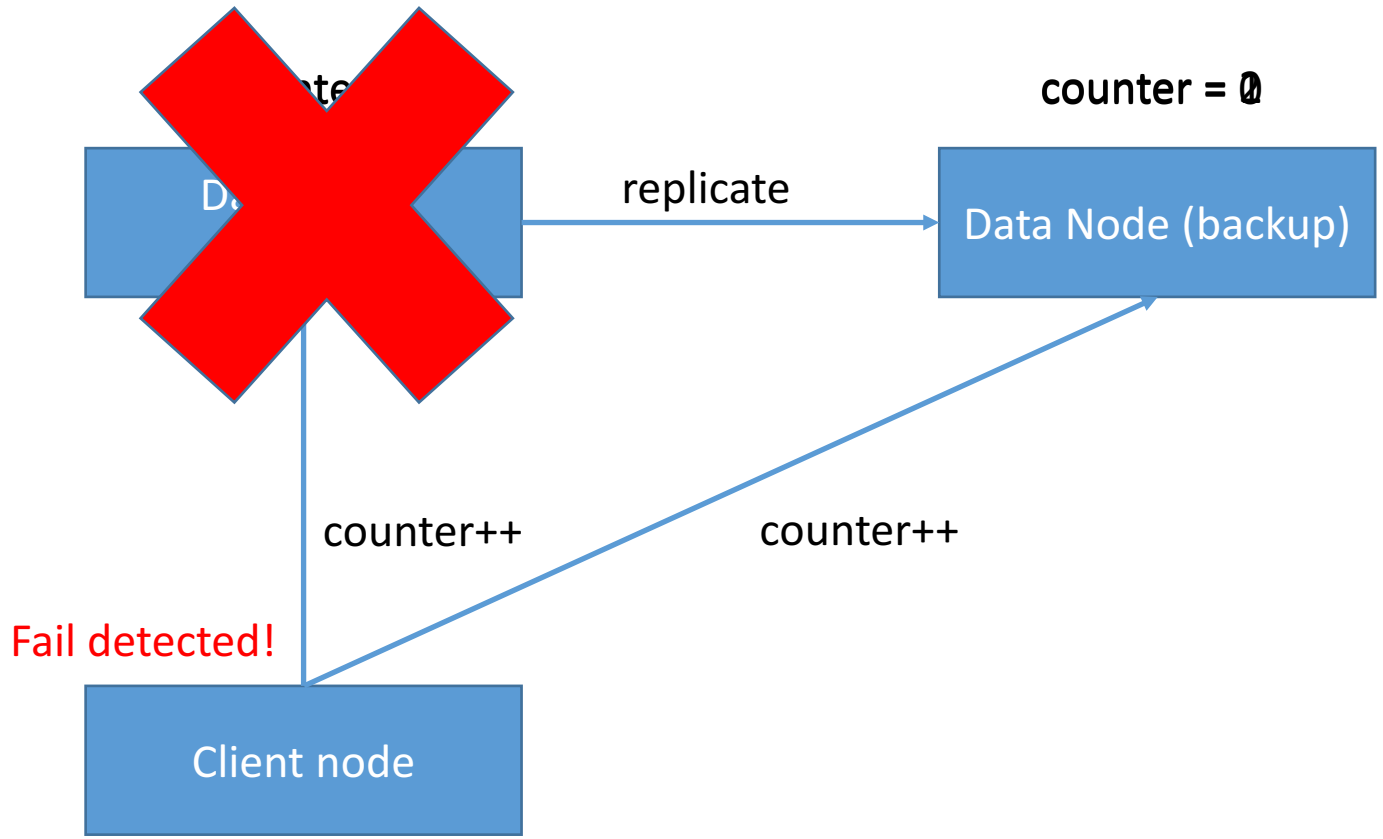
Попытка 3. ConferenceService

```
class ConferenceService {
    Cache<String, Conference> conferenceCache;

    void addParticipant(String conferenceId){
        conferenceCache.invoke(conferenceId,
          new AddParticipantProcessor());
    }

    void removeParticipant(String conferenceId){
        conferenceCache.invoke(conferenceId,
          new RemoveParticipantProcessor());
    }
}
```

Попытка 3. Проблема



Попытка 4. Идемпотентность

```
class Conference {
    Set<String> participants = new HashSet<>();

    void addParticipant(String participantId){
        participants.add(participantId);
    }

    boolean removeParticipant(String participantId){
        participants.remove(participantId);
        return participants.isEmpty();
    }
}
```

Попытка 4. Идемпотентность

```
class Conference {
    Set<String> participants = new HashSet<>();

    void addParticipant(String participantId){
        participants.add(participantId);
    }

    boolean removeParticipant(String participantId){
        participants.remove(participantId);
        return participants.isEmpty();
    }
}
```

Попытка 4. Идемпотентность

```
class Conference {
    Set<String> participants = new HashSet<>();

    void addParticipant(String participantId){
        participants.add(participantId);
    }

    boolean removeParticipant(String participantId){
        participants.remove(participantId);
        return participants.isEmpty();
    }
}
```

Попытка 4. EntryProcessors

```
class AddParticipantProcessor implements EntryProcessor {
    String participantId;

    AddParticipantProcessor(String participantId){
        this.participantId = participantId;
    }
    …
}

class RemoveParticipantProcessor implements EntryProcessor
{}
```

Попытка 4. ConferenceService

```
interface ConferenceService {
    void addParticipant(String conferenceId,
                        String participantId);
    void removeParticipant(String conferenceId,
                           String participantId);
}
```

Попытка 5. Уменьшение размера объекта

```
class Conference {
    Map<String, Integer> servers2Participants;

    boolean setParticipantsCount(String serverId,
                                 int count){
        if (count == 0) {
            servers2Participants.remove(serverId);
        } else {
            servers2Participants.put(serverId, count);
        }
        return servers2Participants.isEmpty();
    }
}
```

Попытка 5. Уменьшение размера объекта

```
class Conference {
    Map<String, Integer> servers2Participants;

    boolean setParticipantsCount(String serverId,
                                 int count){
        if (count == 0) {
            servers2Participants.remove(serverId);
        } else {
            servers2Participants.put(serverId, count);
        }
        return servers2Participants.isEmpty();
    }
}
```

Попытка 5. Уменьшение размера объекта

```
class Conference {
    Map<String, Integer> servers2Participants;

    boolean setParticipantsCount(String serverId,
                                 int count){
        if (count == 0) {
            servers2Participants.remove(serverId);
        } else {
            servers2Participants.put(serverId, count);
        }
        return servers2Participants.isEmpty();
    }
}
```

Попытка 5. Уменьшение размера объекта

```
class Conference {
    Map<String, Integer> servers2Participants;

    boolean setParticipantsCount(String serverId,
                                 int count){
        if (count == 0) {
            servers2Participants.remove(serverId);
        } else {
            servers2Participants.put(serverId, count);
        }
        return servers2Participants.isEmpty();
    }
}
```

Попытка 5. Уменьшение размера объекта

```
class Conference {
    Map<String, Integer> servers2Participants;

    boolean setParticipantsCount(String serverId,
                                 int count){
        if (count == 0) {
            servers2Participants.remove(serverId);
        } else {
            servers2Participants.put(serverId, count);
        }
        return servers2Participants.isEmpty();
    }
}
```

Попытка 5. EntryProcessor

```
class SetParticipantsCountProcessor
                implements EntryProcessor {
    String serverId;
    int count;

    …
}
```

Попытка 5. ConferenceService

```
Map conference2Participants = new HashMap<>();

synchronized void addParticipant(String conferenceId,
                    String participantId){
    int participants = conference2Participants.compute
(conferenceId, (key, count) -> count == null ? 1 : count+1);

    conferenceCache.invoke(conferenceId,
new SetParticipantsCountProcessor(serverId, participants));
    }

    …
}
```

Попытка 5. ConferenceService

```
Map conference2Participants = new HashMap<>();

synchronized void addParticipant(String conferenceId,
                                 String participantId){
    int participants = conference2Participants.compute
(conferenceId, (key, count) -> count == null ? 1 : count+1);

    conferenceCache.invoke(conferenceId,
new SetParticipantsCountProcessor(serverId, participants));
    }

    …
}
```

Попытка 5. ConferenceService

```
Map conference2Participants = new HashMap<>();

synchronized void addParticipant(String conferenceId,
                                 String participantId){
    int participants = conference2Participants.compute
(conferenceId, (key, count) -> count == null ? 1 : count+1);

    conferenceCache.invoke(conferenceId,
new SetParticipantsCountProcessor(serverId, participants));
    }

    …
}
```

Попытка 5. ConferenceService

```
Map conference2Participants = new HashMap<>();

synchronized void addParticipant(String conferenceId,
                    String participantId){
    int participants = conference2Participants.compute
(conferenceId, (key, count) -> count == null ? 1 : count+1);

    conferenceCache.invoke(conferenceId,
new SetParticipantsCountProcessor(serverId, participants));
    }

    …
}
```

Попытка 6. Асинхронность

- Получение кэша в асинхронном режиме

```
IgniteCache<String, Conference> conferenceCache =
ignite.cache("conference").withAsync();
```

- Запуск EntryProcessor

```
conferenceCache.invoke(conferenceId, new
SetParticipantsCountProcessor(serverId, participants));
```

- Получение IgniteFuture

```
IgniteFuture<Conference> igniteFuture =
conferenceCache.future();
```

Попытка 6. CompletableFuture

```
static <V> CompletableFuture<V>
toCompletableFuture(IgniteFuture<V> igniteFuture) {
      CompletableFuture<V> future = new CF<V>();
      igniteFuture.listen(fut -> {
            try {
                  V res = fut.get();
                  future.complete(res);
            } catch (Exception e){
                  future.completeExceptionally(e);
            }
      });
      return future;
}
```

Попытка 6. CompletableFuture

```
static <V> CompletableFuture<V>
toCompletableFuture(IgniteFuture<V> igniteFuture) {
      CompletableFuture<V> future = new CF<V>();
      igniteFuture.listen(fut -> {
            try {
                  V res = fut.get();
                  future.complete(res);
            } catch (Exception e){
                  future.completeExceptionally(e);
            }
      });
      return future;
}
```

Попытка 6. CompletableFuture

```
static <V> CompletableFuture<V>
toCompletableFuture(IgniteFuture<V> igniteFuture) {
      CompletableFuture<V> future = new CF<V>();
      igniteFuture.listen(fut -> {
            try {
                  V res = fut.get();
                  future.complete(res);
            } catch (Exception e){
                  future.completeExceptionally(e);
            }
      });
      return future;
}
```

Попытка 6. CompletableFuture

```
static <V> CompletableFuture<V>
toCompletableFuture(IgniteFuture<V> igniteFuture) {
      CompletableFuture<V> future = new CF<V>();
      igniteFuture.listen(fut -> {
            try {
                  V res = fut.get();
                  future.complete(res);
            } catch (Exception e){
                  future.completeExceptionally(e);
            }
      });
      return future;
}
```

Попытка 6. CompletableFuture

```
static <V> CompletableFuture<V>
toCompletableFuture(IgniteFuture<V> igniteFuture) {
      CompletableFuture<V> future = new CF<V>();
      igniteFuture.listen(fut -> {
            try {
                  V res = fut.get();
                  future.complete(res);
            } catch (Exception e){
                  future.completeExceptionally(e);
            }
      });
      return future;
}
```

Попытка 6. CompletableFuture

```
static <V> CompletableFuture<V>
toCompletableFuture(IgniteFuture<V> igniteFuture) {
      CompletableFuture<V> future = new CF<V>();
      igniteFuture.listen(fut -> {
            try {
                  V res = fut.get();
                  future.complete(res);
            } catch (Exception e){
                  future.completeExceptionally(e);
            }
      });
      return future;
}
```

Задача 1. Выводы

- Проблемы с конкурентностью есть так же в распределенных системах.
- GridGain предоставляет API аналогичный CHM.
- Стоит задумываться о количестве сетевых вызовов и использовать EntryProcessor.
- Семантика исполнения EntryProcessor – at least once.
- Данные передаются по сети, поэтому нужно минимизировать размер объектов.
- Используем асинхронность там где это нужно.

Задача №2

Создание конференции

Задача №2. Формулировка задачи

- В конференции участвуют сессии с нескольких управляющих серверов
- Медиа-сервер один
- Первый участник выбирает медиа сервер. Медиа сервер возвращает MediaInfo
- Остальные участники должны знать MediaInfo для подсоединения к конференции

Попытка 1. Conference

```
class Conference {
    MediaInfo mediaInfo;
}
```

Попытка 1. AddParticipantProcessor

```
public MediaInfo process(MutableEntry entry) {
    Conference conference = entry.getValue();
    if (conference == null){
        conference = new Conference();
        entry.setValue(conference);
    }
    return conference.getMediaInfo();
}
```

Попытка 1. AddParticipantProcessor

```
public MediaInfo process(MutableEntry entry) {
    Conference conference = entry.getValue();
    if (conference == null){
        conference = new Conference();
        entry.setValue(conference);
    }
    return conference.getMediaInfo();
}
```

Попытка 1. AddParticipantProcessor

```
public MediaInfo process(MutableEntry entry) {
    Conference conference = entry.getValue();
    if (conference == null){
        conference = new Conference();
        entry.setValue(conference);
    }
    return conference.getMediaInfo();
}
```

Попытка 1. AddParticipantProcessor

```
public MediaInfo process(MutableEntry entry) {
    Conference conference = entry.getValue();
    if (conference == null){
        conference = new Conference();
        entry.setValue(conference);
    }
    return conference.getMediaInfo();
}
```

Попытка 1. SetMediaInfoProcessor

```
MediaInfo mediaInfo;

public SetMediaInfoProcessor(MediaInfo mediaInfo) {
    this.mediaInfo = mediaInfo;
}

@Override
public Void process(MutableEntry entry) {
    entry.getValue().setMediaInfo(mediaInfo);
    return null;
}
```

Попытка 1. ConferenceService

```
void addParticipant(String conferenceId,
                    String participantId){
    MediaInfo mediaInfo =
      conferenceCache.invoke(conferenceId,
        new AddParticipantProcessor());
    if (mediaInfo == null){
      mediaInfo = createConference(conferenceId);
      conferenceCache.invoke(conferenceId,
        new SetMediaInfoProcessor(mediaInfo));
    }
    connectToConference(mediaInfo, participantId);
}
```

Попытка 1. ConferenceService

```
void addParticipant(String conferenceId,
                    String participantId){
    MediaInfo mediaInfo =
      conferenceCache.invoke(conferenceId,
        new AddParticipantProcessor());
    if (mediaInfo == null){
      mediaInfo = createConference(conferenceId);
      conferenceCache.invoke(conferenceId,
        new SetMediaInfoProcessor(mediaInfo));
    }
    connectToConference(mediaInfo, participantId);
}
```

Попытка 1. ConferenceService

```
void addParticipant(String conferenceId,
                    String participantId){
    MediaInfo mediaInfo =
      conferenceCache.invoke(conferenceId,
        new AddParticipantProcessor());
    if (mediaInfo == null){
      mediaInfo = createConference(conferenceId);
      conferenceCache.invoke(conferenceId,
        new SetMediaInfoProcessor(mediaInfo));
    }
    connectToConference(mediaInfo, participantId);
}
```

Попытка 1. ConferenceService

```
void addParticipant(String conferenceId,
                    String participantId){
    MediaInfo mediaInfo =
      conferenceCache.invoke(conferenceId,
        new AddParticipantProcessor());
    if (mediaInfo == null){
      mediaInfo = createConference(conferenceId);
      conferenceCache.invoke(conferenceId,
        new SetMediaInfoProcessor(mediaInfo));
    }
    connectToConference(mediaInfo, participantId);
}
```

Попытка 1. ConferenceService

```
void addParticipant(String conferenceId,
                    String participantId){
    MediaInfo mediaInfo =
      conferenceCache.invoke(conferenceId,
        new AddParticipantProcessor());
    if (mediaInfo == null){
      mediaInfo = createConference(conferenceId);
      conferenceCache.invoke(conferenceId,
        new SetMediaInfoProcessor(mediaInfo));
    }
    connectToConference(mediaInfo, participantId);
}
```

Попытка 1. Проблема

- При одновременном присоединение двух участников, будут созданы две независимые конференции
- Второй участник должен ждать, пока первый участник не проинициализирует конференцию

Попытка 2. Continuous query

- Позволяет следить за изменениями в кэше
- Remote Filter – предикат, запускается рядом с данными
- Local callback – запускается на подписчике, если Remote Filter возвращает true. Получает старое и новое значение записи

Попытка 2. Continuous query failover

- С каждым update’ом ассоциируется updateCounter, который монотонно возрастает
- Клиент запоминает updateCounter для последнего полученного update
- Backup поддерживает буфер последних update’ов
- Когда primary падает, backup высылает клиенту записи из буфера
- С помощью updateCounter клиент выполняет дедубликацию

Попытка 2. Объект Conference

```
class Conference {
    String firstParticipant;
    MediaInfo mediaInfo;
}
```

Попытка 2. EntryProcessor results

```
interface AddResult extends Serializable {}

class FirstParticipant implements AddResult {}

class ConferenceInitializing implements AddResult {}

class ConferenceInitialized implements AddResult {
   MediaInfo mediaInfo;
}
```

Попытка 2. EntryProcessor results

```
interface AddResult extends Serializable {}

class FirstParticipant implements AddResult {}

class ConferenceInitializing implements AddResult {}

class ConferenceInitialized implements AddResult {
   MediaInfo mediaInfo;
}
```

Попытка 2. EntryProcessor results

```
interface AddResult extends Serializable {}

class FirstParticipant implements AddResult {}

class ConferenceInitializing implements AddResult {}

class ConferenceInitialized implements AddResult {
   MediaInfo mediaInfo;
}
```

Попытка 2. EntryProcessor results

```
interface AddResult extends Serializable {}

class FirstParticipant implements AddResult {}

class ConferenceInitializing implements AddResult {}

class ConferenceInitialized implements AddResult {
   MediaInfo mediaInfo;
}
```

Попытка 2. EntryProcessor results

```
interface AddResult extends Serializable {}

class FirstParticipant implements AddResult {}

class ConferenceInitializing implements AddResult {}

class ConferenceInitialized implements AddResult {
   MediaInfo mediaInfo;
}
```

Попытка 2. EntryProcessor

```
AddParticipantResult process(MutableEntry entry) {
    Conference conference = entry.getValue();
    if (conference == null) {
       conference = new Conference(participantId);
       entry.setValue(conference);
       return new FirstParticipant();
    }
    if (conference.getFirstParticipantId()
                  .equals(participantId)){
       return new FirstParticipant();
    }
    …
}
```

Попытка 2. EntryProcessor

```
AddParticipantResult process(MutableEntry entry) {
    Conference conference = entry.getValue();
    if (conference == null) {
       conference = new Conference(participantId);
       entry.setValue(conference);
       return new FirstParticipant();
    }
    if (conference.getFirstParticipantId()
                  .equals(participantId)){
       return new FirstParticipant();
    }
    …
}
```

Попытка 2. EntryProcessor

```
AddParticipantResult process(MutableEntry entry) {
    Conference conference = entry.getValue();
    if (conference == null) {
       conference = new Conference(participantId);
       entry.setValue(conference);
       return new FirstParticipant();
    }
    if (conference.getFirstParticipantId()
                  .equals(participantId)){
       return new FirstParticipant();
    }
    …
}
```

Попытка 2. EntryProcessor

```
AddParticipantResult process(MutableEntry entry) {
    Conference conference = entry.getValue();
    if (conference == null) {
        conference = new Conference(participantId);
        entry.setValue(conference);
        return new FirstParticipant();
    }
    if (conference.getFirstParticipantId()
                  .equals(participantId)){
        return new FirstParticipant();
    }
    …
}
```

Попытка 2. EntryProcessor

```
…
    if (conference.getMediaInfo() != null) {
     return
      new ConferenceInitialized(conference.getMediaInfo());
    }
    return new ConferenceInitializing();
}
```

Попытка 2. EntryProcessor

```
…
    if (conference.getMediaInfo() != null) {
     return
      new ConferenceInitialized(conference.getMediaInfo());
    }
    return new ConferenceInitializing();
}
```

Попытка 2. ConferenceService

```
void addParticipant(String conferenceId, String participantId){
    AddParticipantResult result =
      conferenceCache.invoke(conferenceId,
      new AddParticipantProcessor(participantId));
    if (result instanceof ConferenceInitializing){
        MediaInfo mediaInfo =
          initService.waitForInitialization(conferenceId);
        connectToConference(mediaInfo, participantId);
    }
    else if (result instanceof ConferenceInitialized) {…}
    else {…}
}
```

Попытка 2. ConferenceInitializationService

```
class ConferenceInitializationService
    implements CacheEntryUpdatedListener<String, Conference>
{
    ConferenceInitializationService() {…}

    MediaInfo waitForInitialization(String conferenceId) {…}

    @Override
    void onUpdated(Iterable<CacheEntryEvent> events) {…}
}
```

Попытка 2. ConferenceInitializationService

```
ConferenceInitializationService() {
  ContinuousQuery query = new ContinuousQuery<>();
  query.setLocalListener(this);
  query.setRemoteFilter(event ->
       event.getOldValue() != null
    && event.getValue() != null
    && event.getOldValue().getMediaInfo() == null
    && event.getValue().getMediaInfo() != null
  );
  conferenceCache.query(query);
}
```

Попытка 2. ConferenceInitializationService

```
ConferenceInitializationService() {
  ContinuousQuery query = new ContinuousQuery<>();
  query.setLocalListener(this);
  query.setRemoteFilter(event ->
        event.getOldValue() != null
    && event.getValue() != null
    && event.getOldValue().getMediaInfo() == null
    && event.getValue().getMediaInfo() != null
  );
  conferenceCache.query(query);
}
```

Попытка 2. ConferenceInitializationService

```
ConferenceInitializationService() {
  ContinuousQuery query = new ContinuousQuery<>();
  query.setLocalListener(this);
  query.setRemoteFilter(event ->
        event.getOldValue() != null
    && event.getValue() != null
    && event.getOldValue().getMediaInfo() == null
    && event.getValue().getMediaInfo() != null
  );
  conferenceCache.query(query);
}
```

Попытка 2. ConferenceInitializationService

```
ConferenceInitializationService() {
  ContinuousQuery query = new ContinuousQuery<>();
  query.setLocalListener(this);
  query.setRemoteFilter(event ->
        event.getOldValue() != null
    && event.getValue() != null
    && event.getOldValue().getMediaInfo() == null
    && event.getValue().getMediaInfo() != null
  );
  conferenceCache.query(query);
}
```

Попытка 2. ConferenceInitializationService

```
CHM<String, CF> futures = new CHM<>();

MediaInfo waitForInitialization(String conferenceId) {
    CompletableFuture<MediaInfo> future =
      futures.computeIfAbsent(conferenceId, key ->
      new CompletableFuture<>()
    );
    MediaInfo mediaInfo = future.get();
    futures.remove(conferenceId);
    return mediaInfo;
}
```

Попытка 2. ConferenceInitializationService

```
CHM<String, CF> futures = new CHM<>();

MediaInfo waitForInitialization(String conferenceId) {
    CompletableFuture<MediaInfo> future =
      futures.computeIfAbsent(conferenceId, key ->
      new CompletableFuture<>()
    );
    MediaInfo mediaInfo = future.get();
    futures.remove(conferenceId);
    return mediaInfo;
}
```

Попытка 2. ConferenceInitializationService

```
CHM<String, CF> futures = new CHM<>();

MediaInfo waitForInitialization(String conferenceId) {
    CompletableFuture<MediaInfo> future =
      futures.computeIfAbsent(conferenceId, key ->
      new CompletableFuture<>()
    );
    MediaInfo mediaInfo = future.get();
    futures.remove(conferenceId);
    return mediaInfo;
}
```

Попытка 2. ConferenceInitializationService

```
CHM<String, CF> futures = new CHM<>();

MediaInfo waitForInitialization(String conferenceId) {
    CompletableFuture<MediaInfo> future =
      futures.computeIfAbsent(conferenceId, key ->
      new CompletableFuture<>()
    );
    MediaInfo mediaInfo = future.get();
    futures.remove(conferenceId);
    return mediaInfo;
}
```

Попытка 2. ConferenceInitializationService

```
CHM<String, CF> futures = new CHM<>();

MediaInfo waitForInitialization(String conferenceId) {
    CompletableFuture<MediaInfo> future =
      futures.computeIfAbsent(conferenceId, key ->
      new CompletableFuture<>()
    );
    MediaInfo mediaInfo = future.get();
    futures.remove(conferenceId);
    return mediaInfo;
}
```

Попытка 2. ConferenceInitializationService

```
void onUpdated(Iterable<CacheEntryEvent> events) {
    events.forEach(event -> {
      CompletableFuture<MediaInfo> future =
      futures.computeIfAbsent(event.getKey(), key ->
        new CompletableFuture<>()
      );
      future.complete(event.getValue().getMediaInfo());
    });
}
```

Попытка 2. ConferenceInitializationService

```
void onUpdated(Iterable<CacheEntryEvent> events) {
    events.forEach(event -> {
      CompletableFuture<MediaInfo> future =
      futures.computeIfAbsent(event.getKey(), key ->
        new CompletableFuture<>()
      );
      future.complete(event.getValue().getMediaInfo());
    });
}
```

Попытка 2. ConferenceInitializationService

```
void onUpdated(Iterable<CacheEntryEvent> events) {
    events.forEach(event -> {
      CompletableFuture<MediaInfo> future =
      futures.computeIfAbsent(event.getKey(), key ->
        new CompletableFuture<>()
      );
      future.complete(event.getValue().getMediaInfo());
    });
}
```

Попытка 2. Проблема

- Вне зависимости от того, пытался ли участник с данного клиента подключиться к конференции или нет, local listener сработает

Попытка 3. Объект Conference

```
class Conference {
    Set<String> subscribedServers;
    String firstParticipant;
    MediaInfo mediaInfo;
}
```

Попытка 3. EntryProcessor

```
class AddParticipantProcessor implements EntryProcessor {

    AddParticipantResult process(MutableEntry entry) {
        …
        conference.subscribe(serverId);
        return new ConferenceInitializing();
    }
}
```

Попытка 3. Remote Filter

```
query.setRemoteFilter(event ->
       event.getOldValue() != null
    && event.getValue() != null
    && event.getOldValue().getMediaInfo() == null
    && event.getValue().getMediaInfo() != null
    && event.getOldValue().isSubscribed(serverId) != null
  );
```

Задача 2. Выводы

- Для ожидания наступления определенного события можно использовать Continuous Query.
- Remote Filter должен отсеивать все события, в которых вы не заинтересованы.

Задача №3

Таймеры

Задача №3. Формулировка задачи

- Предоставить HTTP API для создания таймера срабатывающего через фиксированные промежутки времени
- Когда таймер срабатывает, необходимо отправить HTTP запрос на URL, заданный при создании
- Так же должен быть метод для отмены таймера

Попытка 1. TimerService

```
class TimerService {
    ScheduledExecutorService executor;
    Set<String> timers;
    void createTimer(String timerId, int interval, URL c){
        timers.add(timerId);
        executor.schedule(new TimerTask(timerId, interval,
callback), interval, SECONDS);
    }
    void cancelTimer(String timerId){
        timers.remove(timerId);
    }
}
```

Попытка 1. TimerService

```
class TimerService {
    ScheduledExecutorService executor;
    Set<String> timers;
    void createTimer(String timerId, int interval, URL c){
        timers.add(timerId);
        executor.schedule(new TimerTask(timerId, interval,
callback), interval, SECONDS);
    }
    void cancelTimer(String timerId){
        timers.remove(timerId);
    }
}
```

Попытка 1. TimerService

```
class TimerService {
    ScheduledExecutorService executor;
    Set<String> timers;
    void createTimer(String timerId, int interval, URL c){
        timers.add(timerId);
        executor.schedule(new TimerTask(timerId, interval,
callback), interval, SECONDS);
    }
    void cancelTimer(String timerId){
        timers.remove(timerId);
    }
}
```

Попытка 1. TimerService

```
class TimerService {
    ScheduledExecutorService executor;
    Set<String> timers;
    void createTimer(String timerId, int interval, URL c){
        timers.add(timerId);
        executor.schedule(new TimerTask(timerId, interval,
callback), interval, SECONDS);
    }
    void cancelTimer(String timerId){
        timers.remove(timerId);
    }
}
```

Попытка 1. TimerService

```
class TimerService {
    ScheduledExecutorService executor;
    Set<String> timers;
    void createTimer(String timerId, int interval, URL c){
        timers.add(timerId);
        executor.schedule(new TimerTask(timerId, interval,
callback), interval, SECONDS);
    }
    void cancelTimer(String timerId){
        timers.remove(timerId);
    }
}
```

Попытка 1. TimerTask

```
class TimerTask implements Runnable {
    …
    public void run() {
        if (timers.contains(timerId)) {
            notifyClient(callback);
        }
        if (timers.contains(timerId)){
            executor.schedule(this, interval, SECONDS);
        }
    }
}
```

Попытка 1. TimerTask

```
class TimerTask implements Runnable {
    …
    public void run() {
        if (timers.contains(timerId)) {
            notifyClient(callback);
        }
        if (timers.contains(timerId)){
            executor.schedule(this, interval, SECONDS);
        }
    }
}
```

Попытка 1. TimerTask

```
class TimerTask implements Runnable {
    …
    public void run() {
        if (timers.contains(timerId)) {
            notifyClient(callback);
        }
        if (timers.contains(timerId)){
            executor.schedule(this, interval, SECONDS);
        }
    }
}
```

Попытка 1. Проблема

- Если падает сервер, на котором был создан таймер – таймер перестанет срабатывать

Попытка 2. Cluster Singleton

- Создать кэш Timer. timerId -> (interval, nextTrigger, url).
- При создании/удалении таймера добавляется/удаляется запись в кэш.
- Cluster singleton Timer Service, который подписывается через Continuous Query только на добавление данных в кэш.
- При добавлении данных, Timer Service делает schedule на локальном executor.
- Каждый раз при срабатывании таймера обновляется nextTrigger, если таймер существует.
- И принимается решение о reschedule.

Попытка 2. Обработка падения.

- В случае падения сервера, на которой запущен TimerService, GridGain автоматически перезапускает его на другой ноде.
- При старте TimerService обходит кэш и выполняет schedule для всех таймеров из кэша.

Попытка 2. Вопрос

- Создали таймер с id “123”
- Удалили таймер с id “123”
- Повторно создали таймер с id “123”
- Старый таймер с id “123” сработал
- Как ему не попортить данные нового таймера и прекратить работу?

Ответ:

timerId -> (interval, nextTrigger, url, **UUID**)

Попытка 2. Проблема

- Есть высокая доступность, но нет горизонтальной масштабируемости

Попытка 3. Горизонтальная масштабируемость

- Каждый сервер отвечает за свои таймеры.
- Сервер пишет в кэш Timers информацию о таймере и обновляет ее при срабатывании таймера.
- timerId -> (interval, nextTrigger, url, UUID, **ownerId**)
- Сервер подписывается на события о падении других серверов.

Попытка 3. Обработка падения.

- При падении ноды, все остальные ноды узнают об этом и начинают разбор таймеров.
- Задача разбора – максимально быстро и равномерно распределить таймеры по живым нодам.

Задача №3.1

Разбор таймеров

Разбор таймеров. Попытка 1

- После падения сервера, все остальные сервера проходят по кэшу Timers и меняют owner на себя, используя ChangeOwnerProcessor, если ownerId == failedNodeId.

Попытка 1. Проблема

- Необходимо пройти по всему кэшу, что является неэффективным при большом количестве нод.

Разбор таймеров. Попытка 2

- Использовать SQL Query и предварительно выбрать те записи, для которых ownerId == failedNodeId.
- Совершить итерацию по получившемуся списку и применить ChangeOwnerProcessor.
- ChangeOwnerProcessor по-прежнему проверяет ownerId == failedNodeId.

Попытка 2. Проблема

- Сервера могут получать записи в одинаковом порядке и будут наступать друг другу на пятки.

Разбор таймеров. Попытка 3

- После получения списка таймеров упавшего сервера, выполнить shuffle.

Попытка 3. Наблюдение

- При приближении к концу списка уменьшается вероятность успешного выполнения ChangeOwnerProcessor.

Разбор таймеров. Попытка 4

1) Выбрать первые M записей упавшей ноды. Если список непустой, перейти к след шагу.

2) Выполнить shuffle.

3) Выбрать первые K записей из списка. Выполнить для них ChangeOwnerProcessor.

4) Перейти к 1ому шагу.

5) Эмпирическим способом выбрать оптимальные M и K.

Попытка 4. Проблема

- Сервер, которому event пришел немногим раннее других, скорее всего заберет все таймеры.
- Нет механизма backpressure при разборе.

Разбор таймеров. Попытка 5

- Добавить CapacityMonitor

```
public interface CapacityMonitor {
    void acquire(int slots);
    void release(int slots);
}
```

- Задача, выполняющая разбор будет вызывать метод acquire перед вызовом ChangeOwnerProcessor. Если нет доступных слотов, то задача блокируется.
- Добавление слотов осуществляется с помощью метода release. Инициатором добавления может быть таймер, а может быть внешняя система.

Задача 3. Выводы

- Не забываем про высокую доступность.
- Не забываем про горизонтальную масштабируемость.
- Используем описанный метод для разбора любых сущностей.

Итоги

- Локальные системы писать сложно
- Конкурентные системы писать еще сложнее
- Распределенные системы еще сложнее
- In-memory grid немного упростит вам жизнь

Andrey Ershov
andrershov@gmail.com

Audio&Video Conferences
We’re hiring!

Задача №4. Отправка уведомлений stateless клиенту

Задача №5. Объединение конференций из разных ЦОДов

Задача №6. Автоматический state transfer