МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное
учреждение высшего образования
«Ижевский государственный технический университет
имени М. Т. Калашникова»

К защите

Руководитель направления
д.т.н., профессор Лялин В.Е.
«___» _____ 2016 г.

Филиппов Алексей Николаевич

Модели RPC в проектировании олимпиадного сервера

09.04.04 – Программная инженерия
09.04.04-1 – Разработка программно-информационных систем

Диссертация на соискание академической степени магистра

Магистрант
Филиппов А.Н.

Научный руководитель
к.т.н., профессор
Тарасов В.Г.

Руководитель программы
д.т.н., профессор Лялин В.Е.

Ижевск 2016

РЕФЕРАТ

Объем и структура диссертационной работы. Диссертация содержит введение, три главы, заключение и одно приложение, изложенные на 103 страницах машинописного текста. В работу включены 7 рисунков, список литературы из 23 наименований. В приложении представлены основные фрагменты исходного кода реализации разработанного RPC протокола.

В исследовании проведен обзор, сравнение и анализ программных средств, реализующих удалённый вызов процедур. Выделены характерные особенности систем RPC и требования, предъявляемые к ним. Разработан асинхронный протокол RPC на основе системы очередей сообщений RabbitMQ. Создана реализация протокола для языков программирования Go, Python и C#. Произведена оценка производительности полученного протокола на основе эксперимента.

Ключевые слова: распределённые системы, удалённый вызов процедур, модели RPC, очередь сообщений, система проведения олимпиад.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Разработка систем автоматизации образовательных процессов является важным этапом информатизации общества. Это позволяет повысить эффективность образовательного процесса путём снижения нагрузки на сотрудников, исключения ошибок и неточностей, допускаемых человеком. Также это позволяет создавать программы удалённого обучения без непосредственного участия преподавателя.

Автоматизация в сфере образования затрагивает множество аспектов, от электронных табелей успеваемости и расписаний занятий до автоматизированных тестирований учащихся.

Неотъемлемой частью образовательного процесса является практическая работа. Она характеризуется решением определённого класса задач в пределах изучаемого курса. Существуют группы задач, решения для которых могут быть проверены в автоматическом режиме, к примеру из курсов программной инженерии. Существуют системы, автоматизирующие проверку решений для таких задач, а также проведения лабораторных и практических работ, соревнований и олимпиад.

С ростом количества пользователей таких систем возрастают и системные требования. Одним из способов решения проблем производительности является распределение нагрузки между отдельными серверами проверяющей системы. Это позволяет добиться горизонтальной масштабируемости, то есть возможности увеличения мощности системы путём увеличения количества узлов сети. Такая система уже будет являться распределённой [1]. Для них характерно распределение функций и ресурсов между множеством узлов. Такие системы часто реализуют избыточность ресурсов, что позволяет им оставаться работоспособными даже при выходе части узлов из строя.

Одной из проблем, которые необходимо решить при создании такой системы, это координация работы узлов системы и передача результатов вычислений между ними. Для решения этой проблемы может применяться концепция RPC – remote procedure call. Эта концепция позволяет организовать процесс передачи данных в виде запрос-ответ. Различные типы запросов и передаваемых данных создают необходимость использования различных технологий RPC.

Целью работы является анализ различных моделей использования RPC и разработка технологии RPC, которая требуется олимпиадной системе, но не имеет аналогов, удовлетворяющих требованиям.

Для достижения цели необходимо решить следующие **задачи**:

– исследование моделей использования RPC в олимпиадной системе BACS, а также

требований, предъявляемых к RPC в рамках каждой модели;

– исследование существующих технологий RPC и анализ их соответствия полученным моделям;
– разработка протокола RPC для олимпиадной системы;
– реализация протокола для языков программирования Go, Python и C#.

Объектом исследования являются распределённые системы.

Предметом исследования являются межпроцессные взаимодействия в распределённых системах посредством RPC.

На защиту выносятся результаты разработки и исследования моделей RPC, а также результаты практической реализации технологии RPC.

Научная новизна работы состоит в разработке моделей RPC.

Представленные в диссертации модели описывают качественные особенности и требования, предъявляемые к RPC. Это вносит ясность в работу инженера при выборе определённой технологии, предоставляя понятный для него алгоритм действий.

Практическая ценность работы. На базе полученных моделей разработана технология RPC, которая позволяет организовать асинхронную и надёжную передачу данных с учётом распределения нагрузки и горизонтальной масштабируемости. Данная технология может быть использована для организации вычислений ряда распределённых систем, в том числе BACS, рассмотренной в диссертации.

Реализация и внедрение результатов работы. Разработанная технология внедрена в систему проведения соревнований по спортивному программированию BACS, разрабатываемую силами студентов ИжГТУ, и является одним из средств обеспечения отказоустойчивости и масштабирования системы.

Публикации. В ходе работы над диссертацией было подано 2 патентные заявки:

– Библиотека программных функций для поддержки сетевого взаимодействия между программами в системе «BACS». Принята 10.12.2015.
– Программный модуль тестирования корректности работы интернет-сайта системы «BACS». Принята 18.05.2016.

Объём и структура диссертационной работы. Диссертация содержит введение, 3 главы и заключение, изложенные на 103 с. машинописного текста, а также 1 приложения. В работу включены 7 рис., список литературы из 23 наименований. В приложении представлены основные фрагменты исходного кода реализации разработанного RPC протокола.

анализ межпроцессных взаимодействий в олимпиадной системе BACS, обзор существующих реализаций, анализ требований и постановка цели и задач работы

В первой главе анализируются межпроцессные взаимодействия, делается вывод о целесообразности применения RPC. Делается обзор существующих реализаций RPC. Обосновывается выбор gRPC в качестве синхронного RPC для запросов с ограниченным временем работы, обосновывается необходимость создания асинхронного RPC для запросов с неограниченным временем работы. Формулируются цели и задачи работы.

Во второй главе подробно рассматриваются связи между удалёнными компонентами олимпиадной системы BACS и требования к ним. Разрабатывается протокол асинхронного RPC на основе системы очередей сообщений.

В третьей главе рассматривается процесс разработки RPC на основе брокера сообщений RabbitMQ. Оценивается производительность реализованного RPC.

В заключении диссертационной работы сформулированы основные выводы и результаты выполненных исследований и намечены возможные перспективные направления их развития.

В приложении представлены фрагменты исходного кода реализации протокола для языков Go, Python и C#.

1. АНАЛИТИЧЕСКИЙ ОБЗОР МЕЖПРОЦЕССНЫХ ВЗАИМОДЕЙСТВИЙ И ФОРМИРОВАНИЕ ТРЕБОВАНИЙ К НИМ

Целью данной главы является анализ межпроцессных взаимодействий в олимпиадной системе BACS, обзор существующих реализаций, анализ требований и постановка цели и задач работы.

1.1. Олимпиадная система BACS

Олимпиадная система BACS это распределённая система проведения соревнований и лабораторных работ среди учащихся по дисциплинам, связанным с программированием. Система состоит из нескольких компонент, связанных между собой.

В олимпиадной системе можно выделить два типа связей:

– Запрос существующих данных, пример запросов между Web-интерфейсом и архивом задач приведён на рисунке 1.1. При штатной работе системы обработка запроса занимает строго ограниченное время. В случае сбоя запрос может зависнуть. Необходимо прекращать обработку данного запроса, информируя вызывающую сторону об ошибке. Такие запросы являются синхронными.
– Запрос на совершение ресурсоёмкой операции. Такие запросы происходят между Web-интерфейсом и сервером тестирования решений: тестирование пользовательского решения. Длительность тестирования решения заранее неизвестна, потому следует обеспечить надёжность асинхронной обработки запроса. При этом ответ Web-интерфейсу отправляется сервером тестирования самостоятельно.

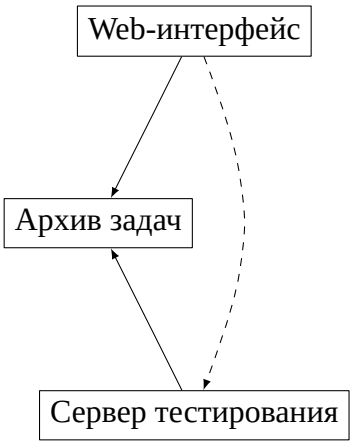

Рис. 1.1: Связи между компонентами BACS

Такие связи являются примером межпроцессного взаимодействия.

1.2. Межпроцессное взаимодействие

Межпроцессное взаимодействие IPC (inter-process communication) – это обмен данными между отдельными процессами. Процессы могут быть запущены как в одном адресном пространстве, так и на удалённых компьютерах. Обеспечиваются такие взаимодействия как посредством ядра операционной системы, так и при помощи механизмов пользовательского пространства (к примеру внешних программных модулей).

Среди механизмов IPC выделяют механизмы обмена сообщениями, синхронизации, разделения памяти и удалённых вызовов (RPC – remote procedure call). Для распределённых систем актуальны механизмы, независимые от ядра операционной системы и позволяющие реализовывать связь между узлами, запущенными под управлением различных операционных систем. К таким относятся механизмы обмена сообщениями и удалённых вызовов.

1.2.1. Обмен сообщениями

Обмен сообщениями – это форма связи, используемая в параллельных вычислениях, реализуемая путём посылки пакетов информации получателям, которые могут содержать команды и уведомления, а также данные для обработки.

Обычно обмен сообщениями реализуется асинхронно, сообщение может быть доставлено неопределённому кругу получателей, в том числе независимо от отправителя, если используется программа-посредник – брокер. Наличие брокера с одной стороны приводит к централизации системы, с другой позволяет упростить архитектуру. Брокер хранит сообщения до момента доставки, уведомляет об ошибках, реализует маршрутизацию сообщений.

1.2.2. RPC

Удалённый вызов процедур – это класс технологий, позволяющих производить вызов процедур одной компьютерной программы из адресного пространства другой. Как правило реализации RPC позволяют абстрагироваться от деталей сетевого взаимодействия, фактически стирая разницу между вызовом удалённой и локальной процедуры.

Технологии RPC удобно применять для организации вычислений в распределённых системах. В языках программирования высокого уровня команда есть вызов процедуры. В распределённых системах таким образом команда есть удалённый вызов процедуры. RPC позволяет передавать структурированные данные небольшого объёма (десятки мегабайт). Так как используются типы данных языков программирования высокого уровня, происходит прозрач-

ная интеграция различных частей системы.

Любая технология RPC состоит из двух частей: протокола RPC и реализации протокола для конкретных языков программирования. Следует обратить внимание, что для большинства языков программирования следует создавать отдельную реализацию (в некоторых случаях обёртку к существующей), так как в каждом языке есть свои характерные особенности стиля работы, что позволяет сделать реализацию максимально приближенной к языку и понятной пользователям.

Протокол RPC с другой стороны является общим, он един для всех реализаций. Это позволяет объединять различные узлы распределённой системы, реализованные на различных языках программирования.

Протокол RPC обычно является двухуровневым. Уровень представления данных определяет методы кодирования информации, такие как байтовое представление чисел и других простых типов данных, массивов, строк, структур. Транспортный уровень определяет механизмы передачи уже закодированных байт по сети. Часто используется какой-либо существующий протокол в качестве транспортного.

1.3. Требования к технологии RPC

Среди требований, предъявляемых к RPC, выделяют:

- переносимость данных;
- надёжность;
- производительность;
- персистентность;
- диагностика неисправностей.

1.3.1. Переносимость данных

Разные части системы могут быть запущенны на различных платформах. Это включает в себя как различные архитектуры процессора, операционные системы, так и языки программирования и интерпретаторы. Представление данных на различных платформах может существенно отличаться, к примеру, различным порядком байт, выравниванием, размером указателя, представлением строк и массивов. Простое копирование представления данных из оперативной памяти не позволит взаимодействовать различающимся частям системы. Необходим механизм сериализации в платформо-независимое представление для последующей передачи в другую часть системы.

Для системы BACS этот критерий является обязательным, так как одним из основных качеств системы является модульность. Это означает заменяемость компонент без изменения интерфейса, а значит возможность замены к примеру языка программирования в отдельных компонентах.

1.3.2. Надёжность и производительность

Отдельные компоненты могут быть запущены на различных серверах. Сбои при работе сети, технические работы на сервере (даже кратковременные), моменты пиковой нагрузки с временным отказом от обработки новых запросов, – всё это может приводить к недоступности или медленной работе отдельных узлов системы. При запросах к этим узлам остальные узлы должны адекватно реагировать на их недоступность, корректно определяя это в приемлемое время, повторяя запрос при необходимости.

1.3.3. Персистентность

В асинхронных связях данные являются более ценными, чем в синхронных. Это связано со стоимостью их получения: обработка асинхронного запроса может занимать существенно больше времени, потому желательно избегать повторения запросов без крайней на то необходимости. Эта проблема решается при помощи персистентного хранения запросов и ответов, что даёт возможность получения ответа на запрос, даже если обработчик уже прекратил свою работу, или отправки запроса при отсутствии свободных обработчиков.

Ни одна из рассмотренных RPC технологий не реализует это требование. Для его реализации необходим промежуточный сервис – брокер.

1.3.4. Диагностика неисправностей

При обработке запросов могут возникать ошибки. Важно как можно более полно передавать информацию об этих ошибках для последующей диагностики системными администраторами или разработчиками системы. В информацию об ошибке может входить стек вызовов, время вызова, данные запроса, идентификаторы вызывающей и принимающей сторон.

1.4. Обзор существующих технологий RPC

1.4.1. XML-RPC

Использует XML для представления данных, HTTP в качестве транспортного протокола.

Пример запроса:

```
<?xml version=”1.0”?>
<methodCall>
  <methodName>examples.getStateName</methodName>
  <params>
    <param>
     <value><i4>41</i4></value>
    </param>
  </params>
</methodCall>
```

Пример ответа:

```
<?xml version=”1.0”?>
<methodResponse>
  <params>
    <param>
      <value><string>South Dakota</string></value>
    </param>
  </params>
</methodResponse>
```

Сильными сторонами технологии являются исключительная простота и распространённость среди языков программирования. Недостатками являются низкая производительность и избыточность представления данных (недостаток XML). Последнее влечёт за собой чрезмерную нагрузку на сеть при значительных объёмах передаваемых данных. В случае же если преобладают бинарные данные, они кодируются в base64, специальной кодировке для представления бинарных данных в текстовом виде. Особенностью данной кодировки является увеличение размера передаваемой информации на 33% [13].

1.4.2. JSON-RPC

Технология аналогична XML-RPC, за исключением использования JSON вместо XML. Удобна для реализации в Web ввиду распространённости поддержки JSON в браузерах. Сильные и слабые стороны те же.

1.4.3. D-BUS

Система для организации RPC в пределах одной операционной системы. Оперирует концепцией сервиса: каждое серверное приложение при запуске регистрирует сервис с заранее известным именем. Клиенты будут обращаться к сервису при помощи данного имени.

Сообщения в D-Bus бывают четырёх видов: вызовы методов, результаты вызовов методов, сигналы (широковещательные сообщения) и ошибки.

Используется собственный бинарный протокол представления данных. В качестве транспортного протокола используется как правило доменный сокет UNIX, но может быть применён и TCP.

Сильными сторонами являются высокая скорость работы и широкая поддержка в языках программирования. Так как технология ориентирована на работу в пределах одного компьютера, для распределённых систем не подходит.

1.4.4. Java RMI

Java remote method invocation – программный интерфейс вызова удалённых методов в языке Java.

```
import java.rmi.Remote;
import java.rmi.RemoteException;

public interface RmiServerIntf extends Remote {
    public String getMessage() throws RemoteException;
}
```

Для кодирования данных используется встроенная в язык технология сериализации. В качестве транспортного протокола используется TCP с реализованным поверх него собственным бинарным протоколом.

Сильными сторонами являются высокая скорость работы и бесшовная интеграция с

кодом на языке Java. Недостатком – поддержка только платформы Java. Последнее не позволяет создавать по-настоящему кросс-платформенные интерфейсы в распределённых приложениях, если только для разработки не используется исключительно Java.

1.4.5. SOAP

Simple Object Access Protocol – простой протокол доступа к объектам. Является расширением XML-RPC. Он является более гибким, но также добавляет и свои недостатки к XML-RPC: существуют несовместимые реализации протокола. Технология, как и XML-RPC, снижает скорость работы за счёт передачи данных в форме XML.

1.4.6. Apache Thrift

Это язык описания интерфейсов для определения и создания служб на различных языках программирования. Является фреймворком RPC. Содержит генератор кода для таких языков как C#, C++, Cappuccino, Cocoa, Delphi, Erlang, Go, Haskell, Java, OCaml, Perl, PHP, Python, Ruby и Smalltalk.

Поддерживает множество форматов передачи и кодирования данных, среди них бинарные и текстовые, в том числе отладочный (легко читаемый человеком), JSON.

Поддерживает различные транспортные протоколы, в том числе сетевые.

Сильными сторонами являются гибкость, распространённость реализаций для различных языков программирования и совместимость между ними, высокая скорость работы. Основным недостатком выделяют низкое качество документации. Кроме этого технология сериализации данных неотделима от фреймворка, потому её использование для сохранения данных например на жёсткий диск может быть затруднительно.

Технология широко применяется компанией Facebook для построения распределённых систем.

1.4.7. gRPC

gRPC – высокопроизводительный фреймворк для построения сервисов и клиентов. Разрабатывается компанией Google, находится в открытом доступе.

Технология использует Google Protocol Buffers в качестве технологии представления данных. Эта технология широко распространена среди различных языков программирования.

В качестве протокола передачи данных используется HTTP/2 – современная версия

протокола HTTP. Она отличается поддержкой параллельной передачи данных в рамках одного соединения.

gRPC имеет реализации для C, C++, Java, Go, Node.js, Python, Ruby, Objective-C, PHP и C#. Отличается высокой скоростью работы и совместимостью между реализациями. К недостаткам относят сложность анализа закодированных данных. Также следует иметь ввиду что в настоящее время многие реализации являются новыми и не получили достаточной стабильности работы.

1.5. Постановка цели и задач работы

Среди рассмотренных технологий RPC лучше всего удовлетворяют требованиям синхронных связей олимпиадной системы gRPC и Apache Thrift. Обе технологии имеют реализацию для большинства популярных платформ и языков программирования. Предпочтение отдано gRPC по причине лучшей модульности: возможность использовать одинаковый формат данных для хранения и передачи. gRPC использует Google Protocol Buffers, который удобно применять при сохранении данных на жёсткий диск.

Существует второй тип связи – асинхронный. Для асинхронной связи важна персистентность, то есть возможность продолжить обработку сообщения даже в случае временной недоступности отправителя или получателя. Эта возможность обычно реализуется при помощи промежуточного сервера – брокера сообщений. Ни одна из рассмотренных технологий не удовлетворяет требованию персистентности, таким образом необходимо разработать данную технологию.

Разработка ведётся в два этапа:

1) разработка протокола, независимого от платформы (операционной системы или языка программирования);
2) разработка клиентов для различных платформ, для олимпиадной системы BACS важны Go, Python и C#.

1.6. Выводы

В олимпиадной системе BACS есть различные виды связей между удалёнными компонентами. Для реализации синхронных связей применимы существующие технологии, выбор остановлен на gRPC. В случае же с асинхронной связью показано, что нет подходящей реализации RPC, потому ставится задача разработки асинхронной технологии RPC на основе брокера сообщений.

2. РАЗРАБОТКА МОДЕЛЕЙ ИСПОЛЬЗОВАНИЯ RPC

В главе подробно рассматриваются связи между удалёнными компонентами олимпиадной системы BACS и требования к ним. Разрабатывается протокол асинхронного RPC на основе системы очередей сообщений.

2.1. Синхронные RPC

Архив задач является одним из важнейших компонентов олимпиадной системы. Он содержит все задачи, которые используются для проведения соревнований. Без задач работа системы не имеет смысла.

Задача представляет из себя набор файлов в определённом формате, Архив предоставляет унифицированный интерфейс доступа к задачам вне зависимости от формата. Различным частям олимпиадной системы важны разные компоненты задачи, в частности Web-интерфейсу необходимо запрашивать метаинформацию: имя задачи, параметры тестирования, авторов. Подобная информация имеет ряд характерных особенностей:

- размер сильно ограничен, обычно до 1 мегабайта;
- присутствует у каждой задачи вне зависимости от формата;
- в случае обновления необходимо незамедлительно обновлять данные Web-интерфейса, выдавать пользователю устаревшие данные недопустимо.

2.1.1. Интерфейс Архива задач

Интерфейс Архива задач состоит из двух частей. Первая часть это интерфейс администратора. Он позволяет добавлять, изменять и скачивать задачи. Вторая часть это запросы, которые обычно инициируются Web-интерфейсом системы: получение списка всех задач, подробной и краткой информации об определённых задачах. В обоих случаях важна производительность каждого запроса. Если же ответ на запрос не может быть доставлен, то необходимости повторять попытку нет, так как скорее всего данные устарели, по этой причине персистентность не является требованием.

2.1.2. Обоснование gRPC

При добавлении задачи в архив передаётся большое количество бинарных данных. Поэтому их кодирование в текстовых представлениях не является эффективным, XML-RPC или

JSON-RPC применять не следует. Выбор был остановлен на gRPC. Эта реализация RPC позволяет эффективно организовать передачу данных, а также их хранение без изменения представления при помощи Google Protocol Buffers.

2.2. Асинхронные RPC

В отличие от запросов к Архиву задач, при тестировании решения особенности работы существенно отличаются.

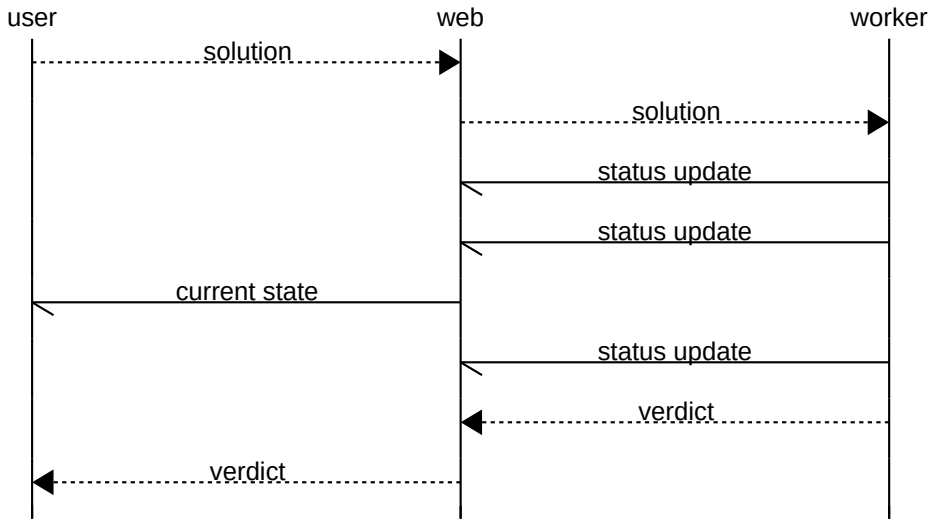

Рис. 2.1: Решение в системе BACS

На рисунке 2.1 представлен пример поведения пользователя и системы при обработке решения. В представленном сценарии пользователь отправляет в систему решение, которое передаётся Web-интерфейсом на сервер проверки. Время тестирования решения неизвестно, потому полезно информировать пользователя о текущем состоянии проверки. При обновлении страницы пользователь увидит что происходит с его решением в данный момент времени.

Необходимо иметь ввиду, что пользователей может быть неопределённое количество, а значит множество решений может находиться на проверке в один момент времени. Помимо этого, проверка решения может занимать неопределённое время, обычно до нескольких минут. Выделять на каждого пользователя отдельный процесс обработки решения в Web-интерфейсе нецелесообразно, потому применение синхронных RPC не представляется возможным. Необходима реализация RPC, которая будет контролировать передачу данных в фоне – асинхронно, при этом допуская временную недоступность узлов (Web-интерфейса или сервера проверки), чтобы ресурсоёмкие операции не приходилось повторять заново, то есть имела свойство персистентности.

В ходе анализа показано, что ни одна из рассмотренных реализаций RPC не решает проблему персистентности. В данной работе рассматривается реализация асинхронного персистентного RPC протокола.

2.3. Разработка протокола

2.3.1. Требования к протоколу

Для олимпиадной системы BACS необходимо разработать протокол передачи сообщений, который будет обладать следующими свойствами:

- надёжность и персистентность: если запрос не был обработан, в том числе по причине отказа принимающей стороны, а также если ответ на запрос временно не может быть доставлен, протокол должен повторять попытки;
- асинхронность: в один и тот же момент времени может обрабатываться неопределённое количество запросов, ответы на них возвращаются по мере выполнения запросов (но сохранение порядка неважно);
- переносимость: протокол должен иметь возможность быть реализованным для различных платформ и языков программирования;
- балансировка нагрузки: протокол должен поддерживать дубликацию сервисов;
- диагностика неисправностей: протокол должен содержать механизм для обработки ошибок.

Набор требований к RPC позволяет в качестве основы для реализации использовать систему очередей сообщений. Очередь сообщений позволяет передавать сообщения между различными компонентами распределённой системы, беря на себя заботу о сохранности данных и их доставки.

2.3.2. Выбор очереди сообщений

При выборе очереди сообщений важно учитывать популярность и стандартизованность реализации. При анализе были рассмотрен следующие очереди сообщений.

Apache ActiveMQ – это брокер сообщений с открытым исходным кодом, реализующий Java Message Service. Обеспечивает кластеризацию и хранение сообщений в различных базах данных, кэширование, ведение журналов.

Apache Kafka – это распределённый брокер сообщений, спроектированный для высокой масштабируемости.

RabbitMQ – это платформа, реализующая обмен сообщениями между компонентами системы на основе протокола AMQP [15]. Имеет множество библиотек на различных платформах и языках программирования для доступа к брокеру сообщений: C++, Java, .NET, Perl, Python, Ruby, PHP, Go. Поддерживает горизонтальное масштабирование.

Выбор был остановлен на RabbitMQ по причине большей распространённости [16], использования протокола AMQP, а значит лучшей гарантии поддержки или простоты замены в будущем брокера сообщений.

2.3.3. Модель RPC

Протокол RPC строится на основе системы очередей сообщений. Каждый RPC два основных сообщения: запрос и ответ, а также опциональные статусные сообщения – краткие описания текущего состояния.

Протокол RPC делится на два уровня: представление данных и транспортный. На уровне представления данных передаётся служебная информация, такая как информация о получателе и отправителе, пользовательский запрос или ответ, информация об успешности выполнения запроса или ошибки. Транспортный уровень строится на основе системы очередей сообщений.

В рамках данного протокола как отправитель запроса, так и его получатель являются клиентами по отношению к брокеру сообщений. Тем не менее для удобства именования рассматривается весь протокол в целом, потому отправитель запроса называется клиентом, а получатель – сервисом.

2.3.3.1. Особенности RabbitMQ

Платформа RabbitMQ обладает рядом особенностей, которые необходимо учитывать при использовании в качестве транспортного уровня. Правильный выбор параметров очередей и самих сообщений позволяет добиться высокой надёжности и эффективности.

Для каждого клиента выделяется две постоянные очереди для получения результатов и сообщений о критических ошибках. Вторая очередь необходима для тех случаев, когда передача сообщения об ошибке невозможна посредством первой очереди. К примеру, если сервис не смог раскодировать запрос из-за ошибки кодирования или несоответствия формата. В таком случае проблема может быть связана с несоответствием версий сервиса и клиента, и информация должна быть передана максимально надёжным способом – в виде текста без использования протоколозависимого представления. Подобного рода критические ошибки не могут быть обработаны протоколом автоматически и предполагается вмешательство извне для их устране-

ния, к примеру системным администратором.

Особенность постоянных очередей в RabbitMQ является их сохранность при перезапуске брокера сообщений или его клиента. Сообщение, будучи сохранённым в очередь, должно быть передано для обработки при первой возможности. Временная недоступность сети, клиента или самого брокера не должно приводить к потере сообщения. Каждая из постоянных очередей и сообщений в них помечаются параметром durable=true, который гарантирует их хранение в постоянной памяти компьютера. При чтении сообщений из этих очередей используется режим с подтверждением обработки сообщений noAck=false. После того, как сообщение было обработано и, к примеру, сохранено в базе данных клиента, брокер должен получить подтверждение. Только после этого сообщение будет удалено из очереди навсегда. Если же подтверждение не получено, то после отключения клиента от брокера сообщение помещается в очередь заново. Это позволяет не беспокоиться о сохранности сообщений в случае аварийного завершения клиента.

Помимо двух постоянных очередей клиент создаёт временную очередь статусов. Статус – это специальное сообщение, которое содержит информацию о текущем состоянии выполнения запроса. Характер использования очереди статусов кардинально отличается от использования постоянных очередей. Здесь в первую очередь важна не сохранность сообщений, а скорость их доставки. Статус не влияет на логику работы системы и быстро устаревает. В первую очередь он необходим для пользователя, который хочет иметь возможность наблюдать состояние выполнения запроса. Можно заметить, что хранить больше одного состояния смысла нет. Также устаревшие состояния не имеют смысла, потому очередь, как и все сообщения в ней, помечаются параметром durable=false, а также для очереди устанавливается параметр autoDelete=true, который указывает, что брокер будет удалять очередь при отключении клиента. Попытка доставить сообщение в несуществующую очередь ведёт к его удалению, что и требуется для временных сообщений.

2.3.4. Модель протокола

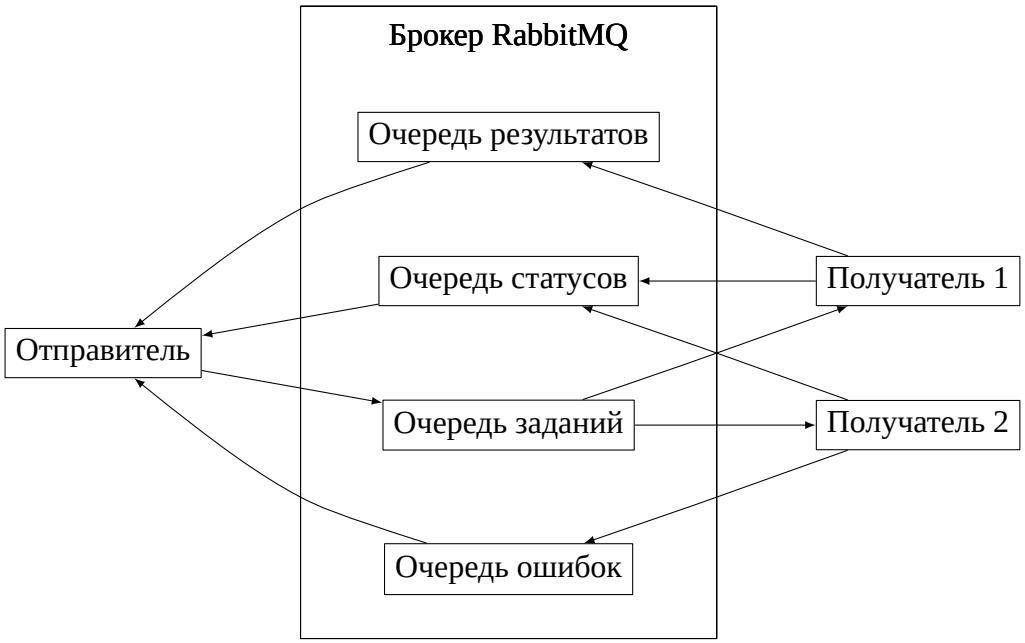

Рис. 2.2: Модель протокола

Протокол предусматривает наличие нескольких сервисов в системе одновременно. Каждый сервис имеет уникальное имя, но при этом сервис может быть продублирован для обеспечения надёжность в случае сбоя одной или нескольких его копий. При сбое необходимо перенаправить все необработанные сообщения на работающие копии сервиса. Эта задача решается при помощи использования одной постоянной очереди на сервис. Она создаётся при подключении первой из копий и используется всеми последующими копиями. При чтении из очереди сообщения распределяются равномерно между сервисами. Аналогично клиенту, постоянная очередь и все сообщения, помещаемые в неё помечаются параметром durable=true, а для чтения сообщений используется режим noAck=false, то есть необходимо подтверждение обработки сообщения. Так как результатом обработки сообщения является отправка ответа на него, подтверждать обработку следует только после помещения ответа в очередь результатов или ошибок. Следует заметить, что возможен сценарий, при котором сообщение может быть обработано несколько раз, ответ отправлен несколько раз, но подтверждение обработки запроса не отправлено. Поэтому задачей клиента является корректная обработка множественных ответов на один и тот же запрос. Эта проблема является распространённой в распределённых системах и корректная обработка множественных ответов или использование идемпотентных запросов, то есть таких, выполнение которых повторно не приводит к изменению состояния, является

повсеместной практикой [1].

2.3.4.1. Содержание сообщений

В рамках платформы RabbitMQ сообщение это массив байт с дополнительной метаинформацией, содержащей параметры хранения сообщения. Так как в протоколе требуется передача дополнительной информации, к примеру, имена очередей клиента для ответа, необходим метод кодирования информации в массив байт. Этот процесс называется сериализацией. Сериализацией в программировании называют процесс перевода какой-либо структуры данных в последовательность битов. Процесс восстановления закодированного сообщения из последовательности битов называется десериализацией.

Для сериализации и десериализации выбрана библиотека Google Protocol Buffers. Выбор связан с её использованием совместно с gRPC в других частях системы. Google Protocol Buffers – включает в себя язык описания структур данных, которые могут быть сериализованы и десериализованы.

Для удобства изменения структуры разделены на два уровня. Первый уровень это независимое от RabbitMQ представление данных самих запросов и ответов, второй же уровень включает вспомогательную информацию, специфичную для RabbitMQ.

Запрос описывается структурой Task (задание), которая содержит название сервиса worker, имя модуля этого сервиса package, который отвечает за обработку конкретного типа запроса, а также сам запрос в поле data.

Статус (структура Status) содержит целочисленный код состояния code, используемый для кодирования типа состояния, краткое описание типа состояния reason, а также вспомогательная информация data.

Результат (структура Result) содержит тип результата в поле status, его краткое описание в поле reason, результат работы в поле data если выполнение запроса завершилось без ошибок (status=OK), и журнал выполнения запроса в поле log.

Виды состояний результатов:

- OK – выполнение задания прошло успешно;
- ERROR – произошла неизвестная ошибка при выполнении задания;
- PROTO_ERROR – ошибка связана с обработкой Google Protocol Buffers;
- WORKER_ERROR – ошибка связана с сервисом;
- UNKNOWN_WORKER – сервис неизвестен;
- INCOMPATIBLE_PACKAGE – модуль сервиса несовместим;

– PACKAGE_ERROR – ошибка модуля сервиса;
– PACKAGE_NOT_FOUND – модуль не найден;
– PACKAGE_BUILD_ERROR – ошибка инициализации модуля;
– EXECUTION_ERROR – ошибка выполнения модуля;
– EXECUTION_TIMEOUT – превышено время выполнения модуля..

```
syntax = ”proto3”;

package bunsan.broker;
option go_package = ”broker”;

message Task {
  /* required */ string worker = 1;
  /* required */ string package = 2;
  bytes data = 3;
}

message Status {
  int32 code = 1;
  string reason = 2;
  bytes data = 3;
}

message Result {
  enum Status {
    OK = 0;
    ERROR = 1;

    // proto errors
    PROTO_ERROR = 100;

    // worker errors
    WORKER_ERROR = 200;
    UNKNOWN_WORKER = 201;
    INCOMPATIBLE_PACKAGE = 202;

    // package errors
    PACKAGE_ERROR = 300;
    PACKAGE_NOT_FOUND = 301;
    PACKAGE_BUILD_ERROR = 302;
```

```
35
    // execution errors
37  EXECUTION_ERROR = 400;
    EXECUTION_TIMEOUT = 401;
39 }
  Status status = 1;
41 string reason = 2;
  bytes data = 3;  // if status == OK
43 bytes log = 4;
}
```

”Структуры данных”

Для передачи описанных выше структур данных используются дополнительные структуры-обёртки, содержащие служебную информацию для маршрутизации в RabbitMQ.

Для структуры-задания добавляются идентификатор сообщения, дополнительные ограничения на сервис, а также имена очередей для ответа.

Для структур статуса и результата добавляется только идентификатор сообщения.

```
syntax = ”proto3”;
2
package bunsan.broker.rabbit;
4 option go_package = ”rabbit”;

6 import ”bunsan/broker/connection.proto”;
  import ”bunsan/broker/protocol.proto”;
8
  message RabbitTask {
10  string identifier = 1;
    Task task = 2;
12  Constraints constraints = 3;
    string status_queue = 4;
14  string result_queue = 5;
    string error_queue = 6;
16 }

18 message RabbitStatus {
    string identifier = 1;
20  Status status = 2;
  }
22
```

```
message RabbitResult {
  string identifier = 1;
  Result result = 2;
}
```

”Структуры данных для RabbitMQ”

2.4. Выводы

1) Проанализированы синхронные RPC, применяемые при взаимодействии Web-интерфейса и Архива задач, построена их модель, обоснован выбор gRPC для реализации представленной модели.
2) Проанализированы асинхронные RPC, применяемые при взаимодействии Web-интерфейс и системы тестирования, построена их модель, предложен протокол для реализации представленной модели, обоснован выбор Google Protocol Buffers и RabbitMQ для реализации предложенного протокола.

3. РАЗРАБОТКА RPC НА ОСНОВЕ БРОКЕРА СООБЩЕНИЙ

В данной главе рассматривается процесс разработки RPC на основе брокера сообщений RabbitMQ. Оценивается производительность реализованного RPC.

3.1. Реализация RPC

3.1.1. Общая архитектура

При реализации RPC для конкретных платформ важно придерживаться стиля программирования, принятого в рамках самой платформы. Тем не менее можно выделить ряд особенностей, связанных с брокером.

Важным достоинством протокола AMQP [15] является его популярность, и как следствие, существует существенное количество его реализаций. В работе представлена реализация протокола RPC для трёх платформ Go, Python и C#, но в случае реализации протокола для других платформ с императивными языками программирования рекомендуется придерживаться аналогичной архитектуры.

За основу реализации берётся существующая библиотека AMQP, а также библиотека Google Protocol Buffers. Используя эти две библиотеки строится логика работы с подключениями к брокеру, обработка и пересылка сообщений, см. рисунок 3.1.

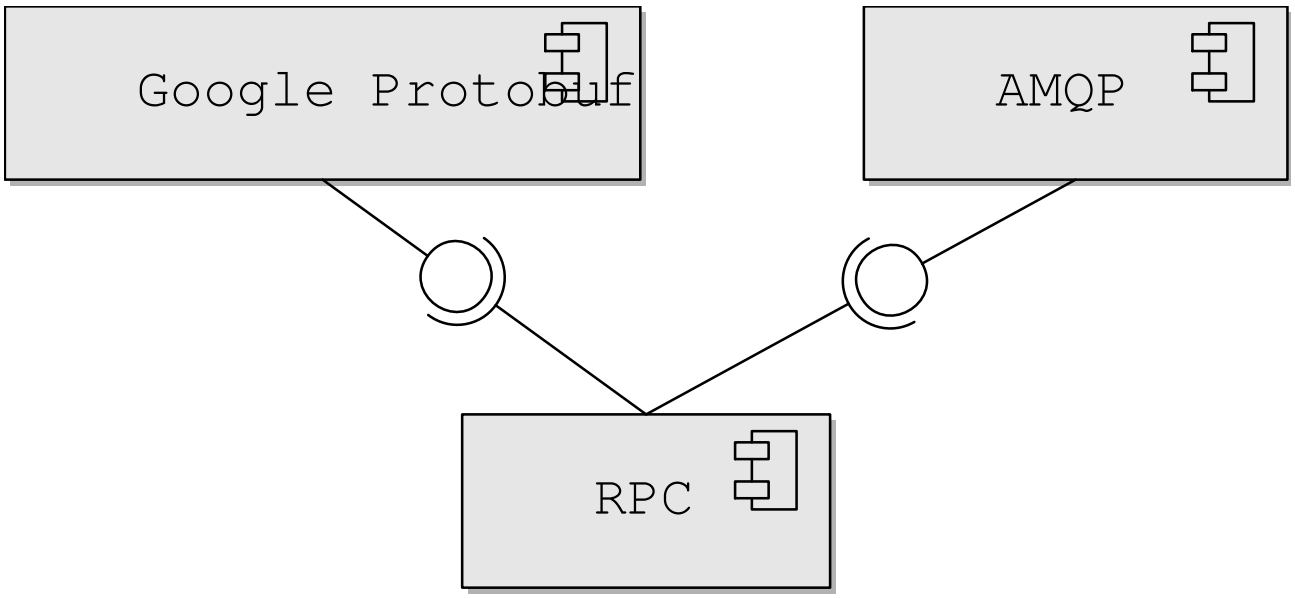

Рис. 3.1: Устройство библиотек

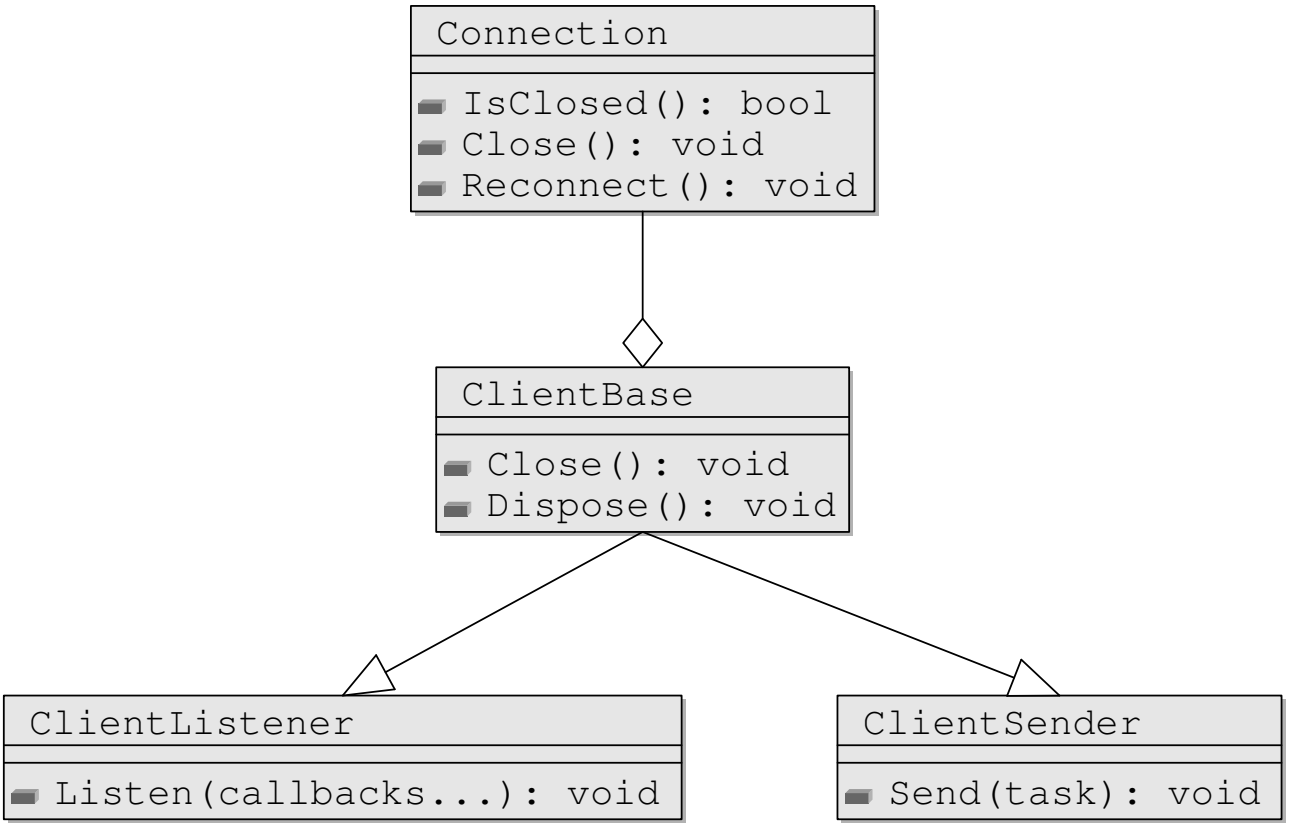

Рис. 3.2: Устройство библиотек

При разработке библиотек выделяется общий компонент – менеджер подключения. Он отвечает за установку и восстановление соединения с брокером. Такая прослойка позволяет упростить логику работы вышестоящих объектов, На основе менеджера подключения создаются классы отправителя и получателя сообщений, см. рис. 3.2.

3.1.2. Реализация на языке Go

В качестве библиотеки AMQP используется streadway/amqp [18]. Также используется официальная реализация Google Protocol Buffers [19].

Исходный код делится на 2 пакета:

– **service** – обработка запросов и ответов;

– **worker** – запуск модулей.

3.1.3. Реализация на языке Python

В качестве библиотеки AMQP используется pika [20]. Также используется официальная реализация Google Protocol Buffers [21].

Исходный код делится на 2 пакета:

– **service** – обработка запросов и ответов;
– **worker** – запуск модулей.

3.1.4. Реализация на языке C#

В качестве библиотеки AMQP используется RabbitMQ .NET client [22].

Также используется неофициальная реализация Google Protocol Buffers [23]. Выбор обусловлен идиоматической реализацией создания структур данных, что упрощает создание пользовательского интерфейса библиотеки.

3.2. Применение разработанных программных решений

Разработанный RPC успешно применяется в олимпиадной системе BACS. Он используется в сервисе тестирования решений для передачи пользовательских решений в кластер серверов для обработки. Каждое решение тестируется при помощи специальных модулей, создаваемых автоматически для каждой имеющейся в системе задачи. Модули генерируются Архивом задач на основе данных самих задач и помещаются в репозиторий. Сервисы BACS скачивают модули из репозитория при помощи библиотеки bunsan::pm [17] и применяют их для тестирования задач. Общая схема работы системы представлена на рисунке 3.3.

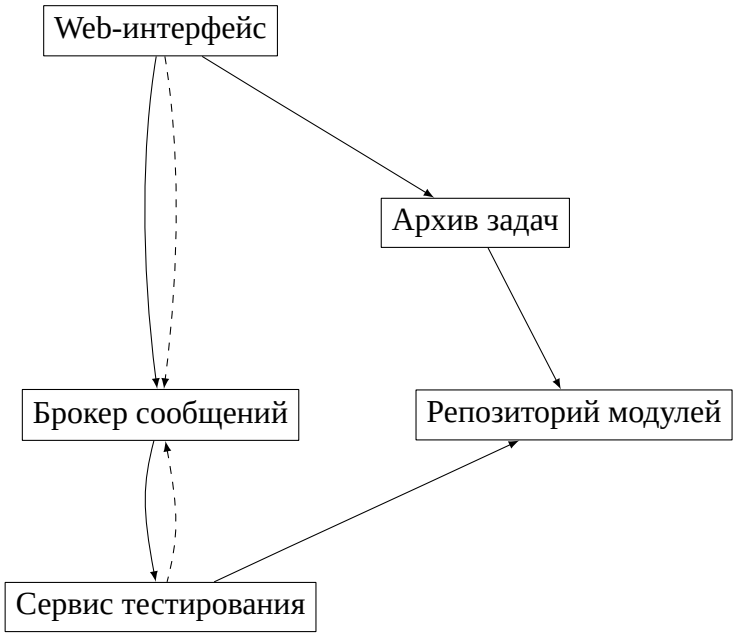

Рис. 3.3: Сервисы системы BACS

3.2.1. Производительность сервиса тестирования решений

Основным показателем эффективности сервиса тестирования решений является его производительность. Она измеряется в среднем количестве проверенных решений в секунду. Для оценки производительности был проведён эксперимент. Целью эксперимента является оценка производительности разработанного RPC, для этого была использована классическая олимпиадная задача ”A+B”, состоящая в сложении двух чисел. Время тестирования решений по этой задаче пренебрежимо мало, меньше 100мс, далее будет показано, что этим можно пренебречь.

В ходе эксперимента было сделано 10000 запросов. Для каждого запроса было измерено время между отправкой запроса и до получения ответа на него. На рисунке 3.4 приведена схема распределения времени выполнения запроса. Среднее время выполнения одного запроса составляет $\mu = \frac{\sum_{i=1}^{N} t_i}{N} = 1.975$ секунд, среднеквадратичное отклонение $\sigma = \sqrt{\frac{\sum_{i=1}^{N} (t_i - \mu)^2}{N}} = 0.229$ секунд. Полученная схема соответствует нормальному распределению с параметрами $\mu = 1.975$ секунд и $\sigma = 0.229$ секунд. Исходя из полученных данных время проверки решения 0.1 секунды это 5% от среднего времени работы, а среднее отклонение 12%, а значит временем проверки решения можно пренебречь.

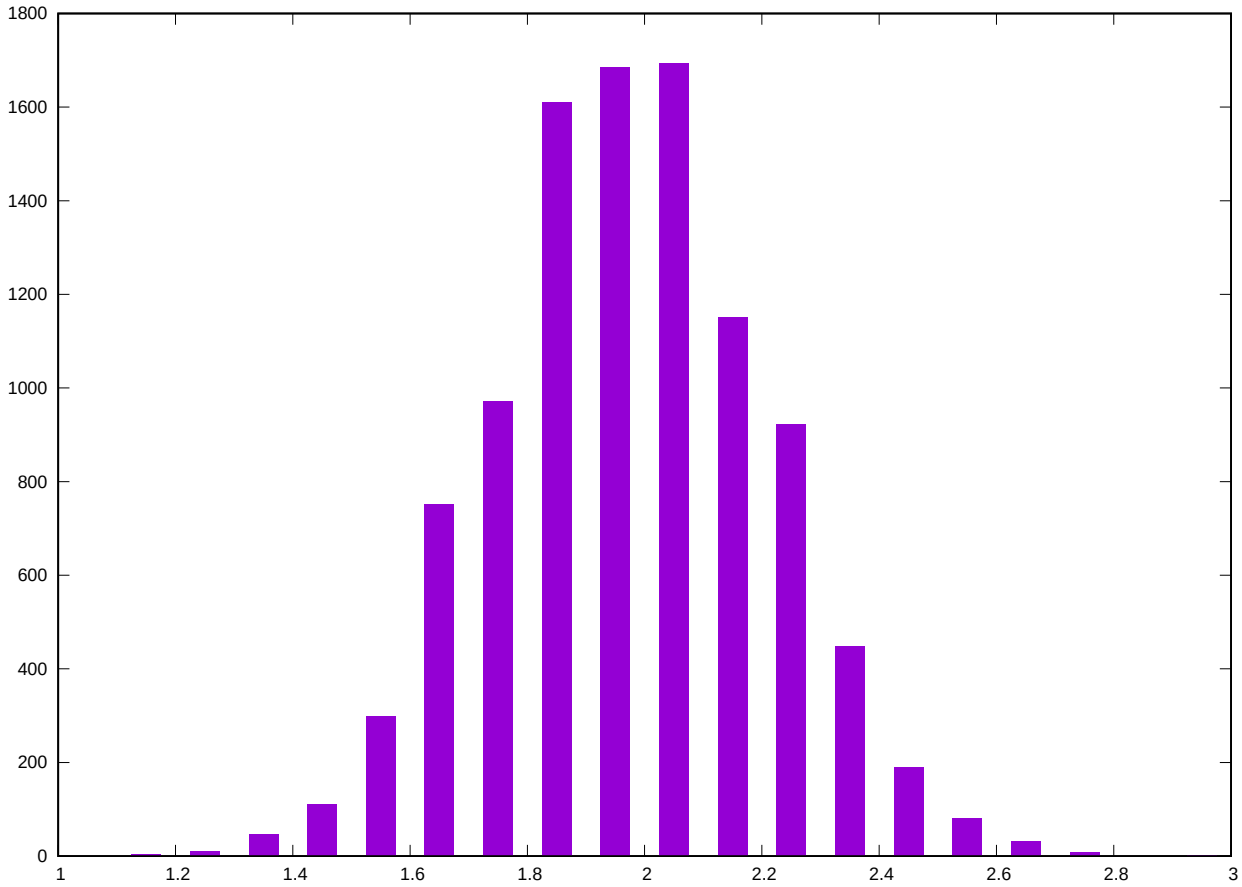

Рис. 3.4: Производительность

3.3. Выводы

– Представлена архитектура реализации созданного во второй главе протокола RPC.
– Показаны особенности реализации протокола для конкретных платформ: Go, Python и C#.
– Представлены результаты оценки производительности RPC. Получены численные результаты, произведён их анализ.

4. ЗАКЛЮЧЕНИЕ

В ходе проделанной работы была достигнута поставленная цель и выполнены следующие задачи:

1) Исследованы модели использования RPC в олимпиадной системе BACS, выделены предъявляемые к RPC требования в рамках каждой модели;
2) исследованы существующие технологии RPC, сделаны выводы о том, что для синхронных RPC с ограниченным временем работы запроса следует использовать gRPC, а для асинхронных RPC без ограничений на время обработки запроса обоснована необходимость создания собственной реализации;
3) разработан асинхронный протокол RPC на основе системы очередей сообщений RabbitMQ;
4) протокол реализован для языков программирования Go, Python и C#, представлена оценка эффективности протокола на основе эксперимента.

На основе реализованного RPC могут создаваться надёжные и масштабируемые распределённые системы.

Список литературы

[1] Таненбаум Э. С. Распределенные системы. Принципы и парадигмы / Эндрю Таненбаум, М. ван Стеен − СПб. : Питер, 2003. − 880 с.

[2] RabbitMQ - Messaging that just works [Online] // RabbitMQ - Messaging that just works. URL: https://www.rabbitmq.com/

[3] grpc / grpc.io [Online] // grpc / grpc.io. URL: http://www.grpc.io/

[4] Таненбаум Э. С. Компьютерные сети / Таненбаум Э. С., Уэзеролл Д. − СПб. : Питер, 2016. − 960 с.

[5] Ejudge [Сайт]. 2006. URL: http://ejudge.ru/ (Дата обращения: 16.06.2014).

[6] ACM Server [Сайт].2008. URL: http://acm-server.ru/ (Дата обращения: 16.04.2016).

[7] Bacs 2.0 2006. [Сайт] URL: http://bacs.cs.istu.ru/ (Дата обращения: 16.04.2016).

[8] XMLRPC 2011, [Сайт] URL: http://xmlrpc.scripting.com/default.html (Дата обращения: 16.04.2016).

[9] ACM ICPC 2014, [Сайт] URL: http://icpc.baylor.edu/ (Дата обращения: 16.04.2016).

[10] Мирзаянов М. Р. Интерактивные задачи: алгоритм тестирования, [Сайт] URL: http://codeforces.ru/blog/entry/5152 (Дата обращения: 16.06.2014).

[11] Michael Kerrisk. Namespaces in operation. 2014 [Сайт] URL: http://lwn.net/Articles/531114/ (Дата обращения: 16.04.2016).

[12] Paul Menage. CGROUPS 2004. [Сайт] URL: https://www.kernel.org/doc/Documentation/cgroups/cgr (Дата обращения: 16.04.2016).

[13] The Base16, Base32, and Base64 Data Encodings. [Сайт] URL : https://tools.ietf.org/html/rfc4648 (Дата обращения: 16.04.2016).

[14] PYTHON MESSAGING: ACTIVEMQ AND RABBITMQ. [Сайт] URL: http://sensatic.net/activemq/python-messaging-activemq-and-rabbitmq.html (Дата обращения: 16.04.2016).

[15] Advanced Message Queuing Protocol. [Сайт] URL: http://www.amqp.org/ (Дата обращения: 16.04.2016).

[16] Evaluating persistent, replicated message queues. [Сайт] https://softwaremill.com/mqperf/ (Дата обращения: 16.04.2016).

[17] Distributed package manager. [Сайт] URL: github.com/bunsanorg/pm/ (Дата обращения: 16.04.2016).

[18] Go client for AMQP 0.9.1. [Сайт] URL: https://github.com/streadway/amqp (Дата обращения: 16.04.2016).

[19] Go support for Google’s protocol buffers. [Сайт] URL: https://github.com/golang/protobuf (Дата обращения: 16.04.2016).

[20] Introduction to Pika. [Сайт] URL: https://pika.readthedocs.io/en/0.10.0/ (Дата обращения: 16.04.2016).

[21] Protocol Buffers - Google’s data interchange format. [Сайт] URL: https://github.com/google/protobuf (Дата обращения: 16.04.2016).

[22] RabbitMQ .NET client. [Сайт] URL: https://github.com/rabbitmq/rabbitmq-dotnet-client (Дата обращения: 16.04.2016).

[23] Protocol Buffers library for idiomatic .NET. [Сайт] URL: https://github.com/mgravell/protobuf-net (Дата обращения: 16.04.2016).

ПРИЛОЖЕНИЕ 1

ИСХОДНЫЙ КОД

1.1. Реализация на языке программирования Go

```
package protocol

type EventReader interface {
  ReadEvent() (Event, error)
}
```

bunsan/broker/worker/protocol/reader.go

```
package text

import (
  ”bufio”
  ”encoding/base64”
  ”fmt”
  ”io”
  ”strings”
)

type rawEventReader struct {
  reader *bufio.Reader
}

func NewRawEventReader(r io.Reader) *rawEventReader {
  return &rawEventReader{bufio.NewReader(r)}
}

func (r *rawEventReader) ReadRawEvent() (RawEvent, error) {
  line, err := r.reader.ReadString(’\n’)
  if err != nil {
    return RawEvent{}, err
  }
  line = strings.Trim(line, ” ”)
  tokens := strings.Split(line, ” ”)
  if len(tokens) != 2 {
    return RawEvent{}, fmt.Errorf(”invalid number of tokens: ”+
      ”expected 2, got %d”, len(tokens))
```

```
  }
  data, err := base64.StdEncoding.DecodeString(tokens[1])
  if err != nil {
    return RawEvent{}, err
  }
  return RawEvent{
    Name: tokens[0],
    Data: data,
  }, nil
}
```

bunsan/broker/worker/protocol/text/raw_reader.go

```
package text

type RawEvent struct {
  Name string
  Data []byte
}
```

bunsan/broker/worker/protocol/text/raw_event.go

```
package text

import (
  ”fmt”
  ”io”

  ”github.com/bunsanorg/broker/go/bunsan/broker”
  ”github.com/bunsanorg/broker/go/bunsan/broker/worker/protocol”
  ”github.com/golang/protobuf/proto”
)

type eventReader struct {
  reader *rawEventReader
}

func NewEventReader(r io.Reader) protocol.EventReader {
  return &eventReader{NewRawEventReader(r)}
}

func (r *eventReader) ReadEvent() (protocol.Event, error) {
```

```
  var status broker.Status
  var result broker.Result
  re, err := r.reader.ReadRawEvent()
  if err != nil {
    return protocol.Event{}, err
  }
  switch re.Name {
  case "status":
    err = proto.Unmarshal(re.Data, &status)
    if err != nil {
      return protocol.Event{}, err
    }
    return protocol.Event{&protocol.Event_Status{&status}}, nil
  case "result":
    err = proto.Unmarshal(re.Data, &result)
    if err != nil {
      return protocol.Event{}, err
    }
    return protocol.Event{&protocol.Event_Result{&result}}, nil
  default:
    return protocol.Event{}, fmt.Errorf("unknown event name %q", re.Name)
  }
}
```

bunsan/broker/worker/protocol/text/reader.go

```
package text

import (
  "bufio"
  "encoding/base64"
  "io"
)

type rawEventWriter struct {
  writer *bufio.Writer
}

func NewRawEventWriter(r io.Writer) *rawEventWriter {
  return &rawEventWriter{bufio.NewWriter(r)}
}
```

```
func (w *rawEventWriter) WriteRawEvent(event RawEvent) error {
  _, err := w.writer.WriteString(event.Name)
  if err != nil {
    return err
  }
  err = w.writer.WriteByte(' ')
  if err != nil {
    return err
  }
  _, err = w.writer.WriteString(
    base64.StdEncoding.EncodeToString(event.Data))
  if err != nil {
    return err
  }
  err = w.writer.WriteByte('\n')
  if err != nil {
    return err
  }
  return w.writer.Flush()
}
```

bunsan/broker/worker/protocol/text/raw_writer.go

```
package text

import (
  "fmt"
  "io"

  "github.com/bunsanorg/broker/go/bunsan/broker/worker/protocol"
  "github.com/golang/protobuf/proto"
)

type eventWriter struct {
  writer *rawEventWriter
}

func NewEventWriter(w io.Writer) protocol.EventWriter {
  return &eventWriter{NewRawEventWriter(w)}
}
```

```
18
func (w *eventWriter) WriteEvent(event protocol.Event) error {
20   var revent RawEvent
     var err error
22   switch ev := event.Kind.(type) {
     case *protocol.Event_Status:
24     revent.Data, err = proto.Marshal(ev.Status)
       revent.Name = ”status”
26   case *protocol.Event_Result:
       revent.Data, err = proto.Marshal(ev.Result)
28     revent.Name = ”result”
     default:
30     return fmt.Errorf(”invalid event oneof”)
     }
32   if err != nil {
       return err
34   }
     return w.writer.WriteRawEvent(revent)
36 }
```

bunsan/broker/worker/protocol/text/writer.go

```
   package protocol
2
   type EventWriter interface {
4    WriteEvent(event Event) error
   }
```

bunsan/broker/worker/protocol/writer.go

```
1  // Code generated by protoc-gen-go.
   // source: bunsan/broker/worker/protocol/event.proto
3  // DO NOT EDIT!

5  /*
   Package protocol is a generated protocol buffer package.
7
   It is generated from these files:
9    bunsan/broker/worker/protocol/event.proto

11 It has these top-level messages:
     Event
```

```
*/
package protocol

import proto "github.com/golang/protobuf/proto"
import fmt "fmt"
import math "math"
import bunsan_broker "github.com/bunsanorg/broker/go/bunsan/broker"

// Reference imports to suppress errors if they are not otherwise used.
var _ = proto.Marshal
var _ = fmt.Errorf
var _ = math.Inf

type Event struct {
  // Types that are valid to be assigned to Kind:
  //  *Event_Status
  //  *Event_Result
  Kind isEvent_Kind `protobuf_oneof:"Kind"`
}

func (m *Event) Reset()         { *m = Event{} }
func (m *Event) String() string { return proto.CompactTextString(m) }
func (*Event) ProtoMessage()    {}

type isEvent_Kind interface {
  isEvent_Kind()
}

type Event_Status struct {
  Status *bunsan_broker.Status `protobuf:"bytes,1,opt,name=status,oneof"`
}
type Event_Result struct {
  Result *bunsan_broker.Result `protobuf:"bytes,2,opt,name=result,oneof"`
}

func (*Event_Status) isEvent_Kind() {}
func (*Event_Result) isEvent_Kind() {}

func (m *Event) GetKind() isEvent_Kind {
  if m != nil {
```

```
53      return m.Kind
    }
55    return nil
}
57
func (m *Event) GetStatus() *bunsan_broker.Status {
59    if x, ok := m.GetKind().(*Event_Status); ok {
        return x.Status
61    }
    return nil
63}

65func (m *Event) GetResult() *bunsan_broker.Result {
    if x, ok := m.GetKind().(*Event_Result); ok {
67      return x.Result
    }
69    return nil
}
71
// XXX_OneofFuncs is for the internal use of the proto package.
73func (*Event) XXX_OneofFuncs() (func(msg proto.Message, b *proto.Buffer) error,
      func(msg proto.Message, tag, wire int, b *proto.Buffer) (bool, error), []
      interface{}) {
    return _Event_OneofMarshaler, _Event_OneofUnmarshaler, []interface{}{
75      (*Event_Status)(nil),
      (*Event_Result)(nil),
77    }
}
79
func _Event_OneofMarshaler(msg proto.Message, b *proto.Buffer) error {
81    m := msg.(*Event)
    // Kind
83    switch x := m.Kind.(type) {
    case *Event_Status:
85      b.EncodeVarint(1<<3 | proto.WireBytes)
      if err := b.EncodeMessage(x.Status); err != nil {
87        return err
      }
89    case *Event_Result:
      b.EncodeVarint(2<<3 | proto.WireBytes)
```

```
    if err := b.EncodeMessage(x.Result); err != nil {
      return err
    }
  case nil:
  default:
    return fmt.Errorf("Event.Kind has unexpected type %T", x)
  }
  return nil
}

func _Event_OneofUnmarshaler(msg proto.Message, tag, wire int, b *proto.Buffer)
   (bool, error) {
  m := msg.(*Event)
  switch tag {
  case 1: // Kind.status
    if wire != proto.WireBytes {
      return true, proto.ErrInternalBadWireType
    }
    msg := new(bunsan_broker.Status)
    err := b.DecodeMessage(msg)
    m.Kind = &Event_Status{msg}
    return true, err
  case 2: // Kind.result
    if wire != proto.WireBytes {
      return true, proto.ErrInternalBadWireType
    }
    msg := new(bunsan_broker.Result)
    err := b.DecodeMessage(msg)
    m.Kind = &Event_Result{msg}
    return true, err
  default:
    return false, nil
  }
}
```

bunsan/broker/worker/protocol/event.pb.go

```
package worker

import (
  "log"
```

```
  "runtime"
  "sync"

  "github.com/bunsanorg/broker/go/bunsan/broker/service"
)

type WorkerPool interface {
  // Add new worker, can be called any time.
  Add(worker Worker) error
  // Process requests from channel.
  DoAll(requests <-chan service.Request) error
  // Finish all consumed requests, do not process new requests.
  Cancel() error
  // Abort everything now, drop consumed requests.
  Abort() error
}

type requestsChannelWaiter struct {
  requestsReady *sync.Cond
  requests      <-chan service.Request
}

func NewRequestsChannelWaiter() *requestsChannelWaiter {
  return &requestsChannelWaiter{
    requestsReady: sync.NewCond(&sync.Mutex{}),
    requests:      nil,
  }
}

func (r *requestsChannelWaiter) Get() <-chan service.Request {
  r.requestsReady.L.Lock()
  for r.requests == nil {
    r.requestsReady.Wait()
  }
  r.requestsReady.L.Unlock()
  return r.requests
}

func (r *requestsChannelWaiter) Set(requests <-chan service.Request) {
  r.requestsReady.L.Lock()
```

```
  if r.requests != nil {
    panic("\"requests\" is set twice")
  }
  r.requests = requests
  r.requestsReady.L.Unlock()
  r.requestsReady.Broadcast()
}

type workerPool struct {
  workersWaitGroup sync.WaitGroup
  workers          []Worker
  requests         *requestsChannelWaiter
  canceler         chan struct{}
  closeFirstWorker sync.Once
  firstWorker      chan struct{}
}

func NewWorkerPool() WorkerPool {
  return &workerPool{
    workers:     make([]Worker, 0, runtime.NumCPU()),
    requests:    NewRequestsChannelWaiter(),
    canceler:    make(chan struct{}),
    firstWorker: make(chan struct{}),
  }
}

func (p *workerPool) do(worker Worker, request service.Request) {
  err := worker.Do(request)
  if err != nil {
    log.Print(err)
  }
}

func (p *workerPool) run(worker Worker) {
  requests := p.requests.Get()
  for run := true; run; {
    select {
    case <-p.canceler:
      run = false
    default:
```

```
      select {
      case request, ok := <—requests:
        if !ok {
          run = false
          continue
        }
        p.do(worker, request)
      case <—p.canceler:
        run = false
      }
    }
  }
  p.workersWaitGroup.Done()
}

func (p *workerPool) Add(worker Worker) error {
  select {
  case <—p.canceler:
    panic(”Called Add() after Cancel()”)
  default:
  }
  p.workers = append(p.workers, worker)
  p.workersWaitGroup.Add(1)
  go p.run(worker)
  p.closeFirstWorker.Do(func() { close(p.firstWorker) })
  return nil
}

func (p *workerPool) DoAll(requests <—chan service.Request) error {
  p.requests.Set(requests)
  <—p.firstWorker
  p.workersWaitGroup.Wait()
  return nil
}

func (p *workerPool) Cancel() error {
  close(p.canceler)
  p.closeFirstWorker.Do(func() { close(p.firstWorker) })
  return nil
}
```

```

func (p *workerPool) Abort() error {
  err := p.Cancel()
  if err != nil {
    return err
  }
  for _, worker := range p.workers {
    werr := worker.Abort()
    if werr != nil {
      // FIXME returns some error
      err = werr
    }
  }
  return err
}
```

bunsan/broker/worker/worker_pool.go

```
package driver

//go:generate bunsan-mockgen -gofile=$GOFILE

import (
  "fmt"
  "sync"

  "github.com/bunsanorg/broker/go/bunsan/broker"
)

var (
  driversMu sync.Mutex
  drivers   = make(map[string]Driver)
)

type Task struct {
  BrokerTask       *broker.Task
  WorkingDirectory string
  StatusWriter     chan<- broker.Status
}

func (task Task) ReadDriverName() (string, error) {
```

```
  if task.BrokerTask.Worker == "" {
    return "", fmt.Errorf("empty driver name")
  }
  return task.BrokerTask.Worker, nil
}

type Driver interface {
  // Execute prepared task.
  // Run does not close task.StatusWriter.
  Run(task Task) (broker.Result, error)
}

func Register(name string, driver Driver) {
  driversMu.Lock()
  defer driversMu.Unlock()
  if driver == nil {
    panic("worker: Register driver is nil")
  }
  if _, dup := drivers[name]; dup {
    panic("worker: Register called twice for driver " + name)
  }
  drivers[name] = driver
}

func Run(task Task) (broker.Result, error) {
  driverName, err := task.ReadDriverName()
  if err != nil {
    return broker.Result{}, err
  }
  driversMu.Lock()
  driver, ok := drivers[driverName]
  driversMu.Unlock()
  if !ok {
    return broker.Result{}, fmt.Errorf(
      "worker: unknown driver %q", driverName)
  }
  return driver.Run(task)
}
```

bunsan/broker/worker/driver/driver.go

```
package driverutil

//go:generate bunsan-mockgen -gofile=$GOFILE

import (
  "fmt"
  "io"
  "os/exec"
)

type StdoutParser func(stdout io.Reader) error

type StdoutRunner interface {
  Run(cmd *exec.Cmd, p StdoutParser) error
}

type RunnerError struct {
  ExecErr   error
  ParserErr error
}

func (e *RunnerError) Error() string {
  return fmt.Sprintf("parser error: %v; exec error: %v",
    e.ParserErr, e.ExecErr)
}

func NewStdoutRunner() StdoutRunner {
  return &stdoutRunner{}
}

type stdoutRunner struct{}

func (r *stdoutRunner) Run(cmd *exec.Cmd, p StdoutParser) (err error) {
  stdout, err := cmd.StdoutPipe()
  if err != nil {
    return err
  }
  if err := cmd.Start(); err != nil {
    return err
  }
```

```
41    defer func() {
        stdout.Close()
43      werr := cmd.Wait()
        if err == nil {
45        err = werr
        } else if werr != nil {
47        err = &RunnerError{
            ExecErr:   werr,
49          ParserErr: err,
          }
51      }
    }()
53  return p(stdout)
}
```

bunsan/broker/worker/driver/driverutil/runner.go

```
  package driver
2
  import (
4   "bytes"
    "io"
6   "log"
    "os/exec"
8   "path"

10  "github.com/bunsanorg/broker/go/bunsan/broker"
    "github.com/bunsanorg/broker/go/bunsan/broker/worker/driver/driverutil"
12  "github.com/bunsanorg/broker/go/bunsan/broker/worker/protocol"
    "github.com/bunsanorg/broker/go/bunsan/broker/worker/protocol/text"
14 )

16 // FIXME should not be placed here
  func init() {
18  Register("bacs_single", &TextDriver{
      Executable: path.Join("bin", "bacs_system_single_worker"),
20    Runner:     driverutil.NewStdoutRunner(),
    })
22 }

24 type TextDriver struct {
```

```
  Executable string
  Runner     driverutil.StdoutRunner
}

func (d *TextDriver) Run(task Task) (broker.Result, error) {
  var wlog bytes.Buffer
  var result broker.Result
  executable := path.Join(task.WorkingDirectory, d.Executable)
  cmd := exec.Command(executable)
  cmd.Dir = task.WorkingDirectory
  cmd.Stdin = bytes.NewReader(task.BrokerTask.Data)
  cmd.Stderr = &wlog
  err := d.Runner.Run(cmd, func(stdout io.Reader) error {
    reader := text.NewEventReader(stdout)
    event, err := reader.ReadEvent()
    for ; err == nil; event, err = reader.ReadEvent() {
      switch ev := event.Kind.(type) {
      case *protocol.Event_Status:
        task.StatusWriter <- *ev.Status
      case *protocol.Event_Result:
        result = *ev.Result
      default:
        log.Fatal("Unknown event type (oneof extended?): %v", event)
      }
    }
    if err != io.EOF {
      return err
    }
    return nil
  })
  if err != nil {
    if exitErr, ok := err.(*exec.ExitError); ok {
      err = nil
      result.Status = broker.Result_EXECUTION_ERROR
      result.Reason = exitErr.Error()
    }
  }
  result.Log = wlog.Bytes()
  return result, err
}
```

bunsan/broker/worker/driver/text_driver.go

```
package worker

//go:generate bunsan-mockgen -gofile=$GOFILE

import (
  "errors"
  "fmt"
  "log"

  "github.com/bunsanorg/broker/go/bunsan/broker"
  "github.com/bunsanorg/broker/go/bunsan/broker/service"
  "github.com/bunsanorg/broker/go/bunsan/broker/worker/driver"
  "github.com/bunsanorg/pm/go/bunsan/pm"
)

var ErrAborted = errors.New("worker was aborted")
var ErrCantAbort = errors.New("worker can't be aborted: not supported")

const (
  numberOfBufferedStatuses = 100
)

type Worker interface {
  Do(request service.Request) error
  Abort() error
}

type worker struct {
  repository pm.Repository
  dir        string
}

func NewWorker(repository pm.Repository, dir string) Worker {
  return &worker{repository, dir}
}

func (w *worker) do(request service.Request) (broker.Result, error) {
  err := w.repository.Extract(request.Task().Package, w.dir)
```

```
  if err != nil {
    return broker.Result{
      Status: broker.Result_PACKAGE_ERROR,
      Reason: err.Error(),
    }, nil
  }
  statuses := make(chan broker.Status, numberOfBufferedStatuses)
  done := make(chan struct{})
  go func() {
    for status := range statuses {
      err := request.WriteStatus(status)
      if err != nil {
        log.Printf("Unable to write status: %v", err)
      }
    }
    close(done)
  }()
  defer func() {
    close(statuses)
    <-done
  }()
  return driver.Run(driver.Task{
    BrokerTask:       request.Task(),
    WorkingDirectory: w.dir,
    StatusWriter:     statuses,
  })
}

func (w *worker) Do(request service.Request) (err error) {
  defer func() {
    if r := recover(); r != nil {
      switch t := r.(type) {
      case error:
        err = t
      case string:
        err = errors.New(t)
      default:
        err = fmt.Errorf("%v", t)
      }
    }
```

```
    if err != nil {
      log.Printf("Unable to finish request: %v", err)
      request.WriteError(err)
      request.Nack()
    } else {
      request.Ack()
    }
  }()
  result, err := w.do(request)
  if err != nil {
    return
  }
  err = request.WriteResult(result)
  return
}

func (w *worker) Abort() error {
  return ErrCantAbort
}
```

bunsan/broker/worker/worker.go

```
// Code generated by protoc-gen-go.
// source: bunsan/broker/rabbit/connection.proto
// DO NOT EDIT!

/*
Package rabbit is a generated protocol buffer package.

It is generated from these files:
  bunsan/broker/rabbit/connection.proto

It has these top-level messages:
  RabbitTask
  RabbitStatus
  RabbitResult
*/
package rabbit

import proto "github.com/golang/protobuf/proto"
import fmt "fmt"
```

```
import math ”math”
import bunsan_broker ”github.com/bunsanorg/broker/go/bunsan/broker”
import bunsan_broker1 ”github.com/bunsanorg/broker/go/bunsan/broker”

// Reference imports to suppress errors if they are not otherwise used.
var _ = proto.Marshal
var _ = fmt.Errorf
var _ = math.Inf

type RabbitTask struct {
  Identifier  string                     ‘protobuf:”bytes,1,opt,name=identifier”
      json:”identifier,omitempty”‘
  Task        *bunsan_broker1.Task       ‘protobuf:”bytes,2,opt,name=task” json:
     ”task,omitempty”‘
  Constraints *bunsan_broker.Constraints ‘protobuf:”bytes,3,opt,name=constraints
     ” json:”constraints,omitempty”‘
  StatusQueue string                     ‘protobuf:”bytes,4,opt,name=
     status_queue” json:”status_queue,omitempty”‘
  ResultQueue string                     ‘protobuf:”bytes,5,opt,name=
     result_queue” json:”result_queue,omitempty”‘
  ErrorQueue  string                     ‘protobuf:”bytes,6,opt,name=error_queue
     ” json:”error_queue,omitempty”‘
}

func (m *RabbitTask) Reset()         { *m = RabbitTask{} }
func (m *RabbitTask) String() string { return proto.CompactTextString(m) }
func (*RabbitTask) ProtoMessage()    {}

func (m *RabbitTask) GetTask() *bunsan_broker1.Task {
  if m != nil {
    return m.Task
  }
  return nil
}

func (m *RabbitTask) GetConstraints() *bunsan_broker.Constraints {
  if m != nil {
    return m.Constraints
  }
  return nil
```

```
}

type RabbitStatus struct {
  Identifier string                ‘protobuf:”bytes,1,opt,name=identifier” json
      :”identifier,omitempty”‘
  Status     *bunsan_broker1.Status ‘protobuf:”bytes,2,opt,name=status” json:”
      status,omitempty”‘
}

func (m *RabbitStatus) Reset()         { *m = RabbitStatus{} }
func (m *RabbitStatus) String() string { return proto.CompactTextString(m) }
func (*RabbitStatus) ProtoMessage()    {}

func (m *RabbitStatus) GetStatus() *bunsan_broker1.Status {
  if m != nil {
    return m.Status
  }
  return nil
}

type RabbitResult struct {
  Identifier string                ‘protobuf:”bytes,1,opt,name=identifier” json
      :”identifier,omitempty”‘
  Result     *bunsan_broker1.Result ‘protobuf:”bytes,2,opt,name=result” json:”
      result,omitempty”‘
}

func (m *RabbitResult) Reset()         { *m = RabbitResult{} }
func (m *RabbitResult) String() string { return proto.CompactTextString(m) }
func (*RabbitResult) ProtoMessage()    {}

func (m *RabbitResult) GetResult() *bunsan_broker1.Result {
  if m != nil {
    return m.Result
  }
  return nil
}
```

bunsan/broker/rabbit/connection.pb.go

```
// Code generated by protoc-gen-go.
```

```
// source: bunsan/broker/connection.proto
// DO NOT EDIT!

/*
Package broker is a generated protocol buffer package.

It is generated from these files:
  bunsan/broker/connection.proto
  bunsan/broker/protocol.proto

It has these top-level messages:
  Credentials
  ConnectionParameters
  Constraints
  Task
  Status
  Result
*/
package broker

import proto "github.com/golang/protobuf/proto"
import fmt "fmt"
import math "math"

// Reference imports to suppress errors if they are not otherwise used.
var _ = proto.Marshal
var _ = fmt.Errorf
var _ = math.Inf

type Credentials struct {
  Method      string `protobuf:"bytes,1,opt,name=method" json:"method,omitempty"
      `
  Username    string `protobuf:"bytes,2,opt,name=username" json:"username,
      omitempty"`
  Password    string `protobuf:"bytes,3,opt,name=password" json:"password,
      omitempty"`
  Certificate []byte `protobuf:"bytes,4,opt,name=certificate,proto3" json:"
      certificate,omitempty"`
  Key         []byte `protobuf:"bytes,5,opt,name=key,proto3" json:"key,omitempty
      "`
```

```
}

func (m *Credentials) Reset()         { *m = Credentials{} }
func (m *Credentials) String() string { return proto.CompactTextString(m) }
func (*Credentials) ProtoMessage()    {}

type ConnectionParameters struct {
  // required
  Identifier  string       `protobuf:"bytes,1,opt,name=identifier" json:"
      identifier,omitempty"`
  Credentials *Credentials `protobuf:"bytes,2,opt,name=credentials" json:"
      credentials,omitempty"`
  Host        string       `protobuf:"bytes,3,opt,name=host" json:"host,
      omitempty"`
  Port        int32        `protobuf:"varint,4,opt,name=port" json:"port,
      omitempty"`
  VirtualHost string       `protobuf:"bytes,5,opt,name=virtual_host" json:"
      virtual_host,omitempty"`
}

func (m *ConnectionParameters) Reset()         { *m = ConnectionParameters{} }
func (m *ConnectionParameters) String() string { return proto.CompactTextString(
    m) }
func (*ConnectionParameters) ProtoMessage()    {}

func (m *ConnectionParameters) GetCredentials() *Credentials {
  if m != nil {
    return m.Credentials
  }
  return nil
}

type Constraints struct {
  Resource []string `protobuf:"bytes,1,rep,name=resource" json:"resource,
      omitempty"`
}

func (m *Constraints) Reset()         { *m = Constraints{} }
func (m *Constraints) String() string { return proto.CompactTextString(m) }
func (*Constraints) ProtoMessage()    {}
```

bunsan/broker/connection.pb.go

```
// Code generated by protoc-gen-go.
// source: bunsan/broker/protocol.proto
// DO NOT EDIT!

package broker

import proto "github.com/golang/protobuf/proto"
import fmt "fmt"
import math "math"

// Reference imports to suppress errors if they are not otherwise used.
var _ = proto.Marshal
var _ = fmt.Errorf
var _ = math.Inf

type Result_Status int32

const (
  Result_OK    Result_Status = 0
  Result_ERROR Result_Status = 1
  // proto errors
  Result_PROTO_ERROR Result_Status = 100
  // worker errors
  Result_WORKER_ERROR         Result_Status = 200
  Result_UNKNOWN_WORKER       Result_Status = 201
  Result_INCOMPATIBLE_PACKAGE Result_Status = 202
  // package errors
  Result_PACKAGE_ERROR       Result_Status = 300
  Result_PACKAGE_NOT_FOUND   Result_Status = 301
  Result_PACKAGE_BUILD_ERROR Result_Status = 302
  // execution errors
  Result_EXECUTION_ERROR   Result_Status = 400
  Result_EXECUTION_TIMEOUT Result_Status = 401
)

var Result_Status_name = map[int32]string{
  0:   "OK",
  1:   "ERROR",
```

```
  100: ”PROTO_ERROR”,
  200: ”WORKER_ERROR”,
  201: ”UNKNOWN_WORKER”,
  202: ”INCOMPATIBLE_PACKAGE”,
  300: ”PACKAGE_ERROR”,
  301: ”PACKAGE_NOT_FOUND”,
  302: ”PACKAGE_BUILD_ERROR”,
  400: ”EXECUTION_ERROR”,
  401: ”EXECUTION_TIMEOUT”,
}
var Result_Status_value = map[string]int32{
  ”OK”:                   0,
  ”ERROR”:                1,
  ”PROTO_ERROR”:          100,
  ”WORKER_ERROR”:         200,
  ”UNKNOWN_WORKER”:       201,
  ”INCOMPATIBLE_PACKAGE”: 202,
  ”PACKAGE_ERROR”:        300,
  ”PACKAGE_NOT_FOUND”:    301,
  ”PACKAGE_BUILD_ERROR”:  302,
  ”EXECUTION_ERROR”:      400,
  ”EXECUTION_TIMEOUT”:    401,
}

func (x Result_Status) String() string {
  return proto.EnumName(Result_Status_name, int32(x))
}

type Task struct {
  // required
  Worker string ‘protobuf:”bytes,1,opt,name=worker” json:”worker,omitempty”‘
  // required
  Package string ‘protobuf:”bytes,2,opt,name=package” json:”package,omitempty”‘
  Data    []byte ‘protobuf:”bytes,3,opt,name=data,proto3” json:”data,omitempty”‘
}

func (m *Task) Reset()         { *m = Task{} }
func (m *Task) String() string { return proto.CompactTextString(m) }
func (*Task) ProtoMessage()    {}
```

```
type Status struct {
  Code   int32  ‘protobuf:”varint,1,opt,name=code” json:”code,omitempty”‘
  Reason string ‘protobuf:”bytes,2,opt,name=reason” json:”reason,omitempty”‘
  Data   []byte ‘protobuf:”bytes,3,opt,name=data,proto3” json:”data,omitempty”‘
}

func (m *Status) Reset()         { *m = Status{} }
func (m *Status) String() string { return proto.CompactTextString(m) }
func (*Status) ProtoMessage()    {}

type Result struct {
  Status Result_Status ‘protobuf:”varint,1,opt,name=status,enum=bunsan.broker.
     Result_Status” json:”status,omitempty”‘
  Reason string        ‘protobuf:”bytes,2,opt,name=reason” json:”reason,
     omitempty”‘
  Data   []byte        ‘protobuf:”bytes,3,opt,name=data,proto3” json:”data,
     omitempty”‘
  Log    []byte        ‘protobuf:”bytes,4,opt,name=log,proto3” json:”log,
     omitempty”‘
}

func (m *Result) Reset()         { *m = Result{} }
func (m *Result) String() string { return proto.CompactTextString(m) }
func (*Result) ProtoMessage()    {}

func init() {
  proto.RegisterEnum(”bunsan.broker.Result_Status”, Result_Status_name,
     Result_Status_value)
}
```

bunsan/broker/protocol.pb.go

```
package main

import (
  ”flag”
  ”io/ioutil”
  ”log”
  ”os”
  ”os/signal”
  ”strings”
```

```
  "syscall"

  "github.com/bunsanorg/broker/go/bunsan/broker"
  "github.com/bunsanorg/broker/go/bunsan/broker/service"
  "github.com/bunsanorg/broker/go/bunsan/broker/worker"
  "github.com/bunsanorg/pm/go/bunsan/pm"
)

var url = flag.String("url", "", "Example: amqp://guest:guest@localhost/")
var jobs = flag.Int("jobs", 1, "Number of parallel jobs")
var constraints = flag.String("constraints", "", ",-separated list")
var repositoryConfig = flag.String("repository-config", "",
  "bunsan::pm configuration file")
var tmpdir = flag.String("tmpdir", "", "Temporary directory for workers")

func abortOnSignal(
  reader service.RequestReader,
  workerPool worker.WorkerPool) {

  sigCh := make(chan os.Signal, 1)
  signal.Notify(sigCh, syscall.SIGINT, syscall.SIGTERM)
  go func() {
    sig := <-sigCh
    log.Printf("Received signal %q, aborting", sig)
    reader.Close()
    workerPool.Abort()
  }()
}

// scope
func run() error {
  constraints := broker.Constraints{
    Resource: strings.Split(*constraints, ","),
  }

  repository, err := pm.NewRepository(*repositoryConfig)
  if err != nil {
    return err
  }
  defer repository.Close()
```

```

  workerPool := worker.NewWorkerPool()
  for i := 0; i < *jobs; i++ {
    dir, err := ioutil.TempDir(*tmpdir, "")
    if err != nil {
      return err
    }
    defer os.RemoveAll(dir)
    worker := worker.NewWorker(repository, dir)
    workerPool.Add(worker)
  }

  reader := service.NewRequestReader(*url)
  defer reader.Close()
  abortOnSignal(reader, workerPool)
  tasks, err := reader.Read(constraints, *jobs)
  if err != nil {
    return err
  }
  log.Print("Processing tasks...")
  return workerPool.DoAll(tasks)
}

func main() {
  flag.Parse()
  if *url == "" {
    log.Fatal("Must set -url")
  }
  if *constraints == "" {
    log.Fatal("Must set -constraints")
  }
  if *repositoryConfig == "" {
    log.Fatal("Must set -repository-config")
  }

  err := run()
  if err != nil {
    log.Fatal(err)
  }
}
```

bunsan/broker/bunsan-broker/main.go

```
package service

import (
  "github.com/bunsanorg/broker/go/bunsan/broker"
  "github.com/bunsanorg/broker/go/bunsan/broker/rabbit"
  "github.com/streadway/amqp"
)

type StatusWriter interface {
  WriteStatus(status broker.Status) error
}

type ResultWriter interface {
  WriteResult(result broker.Result) error
}

type ErrorWriter interface {
  WriteError(err error) error
}

type rabbitStatusWriter struct {
  correlationId string
  writer        ProtoWriter
}

func NewStatusWriter(
  channel *amqp.Channel, queue, correlationId string) StatusWriter {

  return &rabbitStatusWriter{
    correlationId: correlationId,
    writer: &bytesProtoWriter{
      &rabbitBytesWriter{
        Channel:       channel,
        DeliveryMode:  amqp.Transient,
        Destination:   queue,
        CorrelationId: correlationId,
      },
    },
```

```
  }
}

func (w *rabbitStatusWriter) WriteStatus(status broker.Status) error {
  return w.writer.WriteProto(&rabbit.RabbitStatus{
    Identifier: w.correlationId,
    Status:     &status,
  })
}

type rabbitResultWriter struct {
  correlationId string
  writer        ProtoWriter
}

func NewResultWriter(
  channel *amqp.Channel, queue, correlationId string) ResultWriter {

  return &rabbitResultWriter{
    correlationId: correlationId,
    writer: &bytesProtoWriter{
      &rabbitBytesWriter{
        Channel:       channel,
        DeliveryMode:  amqp.Persistent,
        Destination:   queue,
        CorrelationId: correlationId,
      },
    },
  }
}

func (w *rabbitResultWriter) WriteResult(result broker.Result) error {
  return w.writer.WriteProto(&rabbit.RabbitResult{
    Identifier: w.correlationId,
    Result:     &result,
  })
}

type rabbitErrorWriter struct {
  correlationId string
```

```
  writer        BytesWriter
}

func NewErrorWriter(
  channel *amqp.Channel, queue, correlationId string) ErrorWriter {

  return &rabbitErrorWriter{
    correlationId: correlationId,
    writer: &rabbitBytesWriter{
      Channel:       channel,
      DeliveryMode:  amqp.Persistent,
      Destination:   queue,
      CorrelationId: correlationId,
    },
  }
}

func (w *rabbitErrorWriter) WriteError(err error) error {
  return w.writer.WriteBytes([]byte(err.Error()))
}
```

bunsan/broker/service/response_writer.go

```
package service

import (
  ”log”
  ”sync”
  ”time”

  ”github.com/bunsanorg/broker/go/bunsan/broker”
  ”github.com/streadway/amqp”
)

const (
  retryTimeout = time.Second
)

type taskReader func(connection *amqp.Connection) error

type RequestReader interface {
```

```
  Read(constraints broker.Constraints, jobs int) (<-chan Request, error)
  Close() error
}

type rabbitRequestReader struct {
  url       string
  aborter   chan struct{}
  abortOnce sync.Once
  tasks     chan Request
}

func NewRequestReader(url string) RequestReader {
  return &rabbitRequestReader{
    url:     url,
    aborter: make(chan struct{}, 1),
  }
}

func (r *rabbitRequestReader) maintainConnection(reader taskReader) error {
  working := true
  errCh := make(chan error, 1)
  for working {
    log.Printf("Dialing %q broker...", r.url)
    conn, err := amqp.Dial(r.url)
    if err != nil {
      log.Printf("Unable to connect to broker %q: %v", r.url, err)
      log.Printf("Retrying in %v", retryTimeout)
      select {
      case <-r.aborter:
        log.Printf("RequestReader is canceled")
        working = false
        close(r.tasks)
      case <-time.After(retryTimeout):
      }
      continue
    }
    log.Printf("Connected %q broker", r.url)
    go func() { errCh <- reader(conn) }()
    select {
    case <-r.aborter:
```

```
      log.Printf(”RequestReader is canceled”)
60    working = false
      conn.Close()
62    err = <—errCh
      if err != nil {
64      log.Print(err)
      }
66    close(r.tasks)
    case err = <—errCh:
68    if err != nil {
        log.Print(err)
70    } else {
        log.Print(”Disconnected from %q broker”, r.url)
72    }
      conn.Close()
74  }
  }
76  return nil
}
78
func (r *rabbitRequestReader) Read(
80  constraints broker.Constraints, jobs int) (<—chan Request, error) {

82  r.tasks = make(chan Request, jobs)
    go r.maintainConnection(func(connection *amqp.Connection) error {
84    var consumers sync.WaitGroup
      channel, err := connection.Channel()
86    if err != nil {
        return err
88    }
      defer channel.Close()
90    err = channel.Qos(jobs, 0, false)
      if err != nil {
92      return err
      }
94    for _, resource := range constraints.Resource {
        queue, err := channel.QueueDeclare(
96        resource, // name
          true,     // durable
98        false,    // delete when unused
```

```
        false,    // exclusive
100     false,    // no-wait
        nil,      // arguments
102   )
      deliveries, err := channel.Consume(
104     queue.Name, // queue
        "",         // consumer
106     false,      // auto-ack
        false,      // exclusive
108     false,      // no-local
        false,      // no-wait
110     nil,        // args
      )
112   if err != nil {
        return err
114   }
      consumers.Add(1)
116   go func() {
        for delivery := range deliveries {
118       request, err := NewRequest(channel, &delivery)
          if err != nil {
120         // note: error is reported by NewRequest()
            log.Printf("Unable to read request: %v", err)
122         continue
          }
124       log.Printf("Received request %q", delivery.CorrelationId)
          r.tasks <- request
126     }
        consumers.Done()
128   }()
    }
130 consumers.Wait()
    return nil
132 })
  return r.tasks, nil
134 }

136 func (r *rabbitRequestReader) Close() error {
  r.abortOnce.Do(func() {
138   close(r.aborter)
```

```
  })
  return nil
}
```

bunsan/broker/service/request_reader.go

```
package service

//go:generate bunsan-mockgen -gofile=$GOFILE

import (
  "fmt"
  "log"

  "github.com/bunsanorg/broker/go/bunsan/broker"
  "github.com/bunsanorg/broker/go/bunsan/broker/rabbit"
  "github.com/golang/protobuf/proto"
  "github.com/streadway/amqp"
)

type Request interface {
  StatusWriter
  ResultWriter
  ErrorWriter
  Task() *broker.Task
  Ack() error
  Nack() error
}

type rabbitRequest struct {
  task         rabbit.RabbitTask
  statusWriter StatusWriter
  resultWriter ResultWriter
  errorWriter  ErrorWriter
  delivery     *amqp.Delivery
}

// Create new request object, return and send error if it is not possible.
func NewRequest(
  channel *amqp.Channel, delivery *amqp.Delivery) (Request, error) {

```

```
  r := &rabbitRequest{
37    delivery: delivery,
    errorWriter: NewErrorWriter(
39      channel, delivery.ReplyTo, delivery.CorrelationId),
  }
41  err := proto.Unmarshal(delivery.Body, &r.task)
  if err != nil {
43    r.errorWriter.WriteError(err)
    return nil, err
45  }
  if delivery.CorrelationId != r.task.Identifier {
47    err = fmt.Errorf(
      ”invalid task: correlation id %q does not match task id %q”,
49      delivery.CorrelationId, r.task.Identifier)
    r.errorWriter.WriteError(err)
51    return nil, err
  }
53  r.statusWriter = NewStatusWriter(
    channel, r.task.StatusQueue, r.task.Identifier)
55  r.resultWriter = NewResultWriter(
    channel, r.task.ResultQueue, r.task.Identifier)
57  return r, nil
}
59
func (r *rabbitRequest) Task() *broker.Task {
61  return r.task.Task
}
63
func (r *rabbitRequest) WriteStatus(status broker.Status) error {
65  return r.statusWriter.WriteStatus(status)
}
67
func (r *rabbitRequest) WriteResult(result broker.Result) error {
69  return r.resultWriter.WriteResult(result)
}
71
func (r *rabbitRequest) WriteError(err error) error {
73  return r.errorWriter.WriteError(err)
}
75
```

```
func (r *rabbitRequest) Ack() error {
  log.Printf(”Acknowledging request %q”, r.delivery.CorrelationId)
  return r.delivery.Ack(false)
}

func (r *rabbitRequest) Nack() error {
  log.Printf(”Negatively acknowledging request %q”,
    r.delivery.CorrelationId)
  return r.delivery.Nack(false, false)
}
```

bunsan/broker/service/request.go

```
package service

import (
  ”github.com/golang/protobuf/proto”
)

type ProtoWriter interface {
  WriteProto(message proto.Message) error
}

type bytesProtoWriter struct {
  bytesWriter BytesWriter
}

func (w *bytesProtoWriter) WriteProto(message proto.Message) error {
  data, err := proto.Marshal(message)
  if err != nil {
    return err
  }
  return w.bytesWriter.WriteBytes(data)
}
```

bunsan/broker/service/proto_writer.go

```
package service

import (
  ”github.com/streadway/amqp”
)
```

```
type BytesWriter interface {
  WriteBytes(data []byte) error
}

type rabbitBytesWriter struct {
  Channel       *amqp.Channel
  Destination   string
  DeliveryMode  uint8
  CorrelationId string
}

func (w *rabbitBytesWriter) WriteBytes(data []byte) error {
  return w.Channel.Publish(
    "",            // exchange
    w.Destination, // routing key
    false,         // mandatory
    false,         // immediate
    amqp.Publishing{
      Body:          data,
      DeliveryMode:  w.DeliveryMode,
      CorrelationId: w.CorrelationId,
    })
}
```

bunsan/broker/service/bytes_writer.go

1.2. Реализация на языке программирования Python

```
import argparse
import logging
import logging.config

from bunsan.broker import connection_pb2
from bunsan.broker.service import consumer
from bunsan.broker.worker import worker
from google.protobuf import text_format


def main():
```

```
    parser = argparse.ArgumentParser(description='Server')
    parser.add_argument('--logging', required=True,
                        help='Logging configuration file')
    parser.add_argument('--connection', required=True,
                        help='Proto-encoded ConnectionParameters')
    parser.add_argument('--constraints', required=True,
                        help='Proto-encoded Constraints')
    parser.add_argument('--jobs', type=int, default=1,
                        help='Number of jobs to run in parallel')
    parser.add_argument('--repository-config', required=True,
                        help='Configuration for bunsan::pm::repository')
    parser.add_argument('--tmpdir', default='/tmp',
                        help='Temporary directory for task execution')
    args = parser.parse_args()
    logging.config.fileConfig(args.logging)
    _logger = logging.getLogger(__name__)
    connection_parameters = connection_pb2.ConnectionParameters()
    constraints = connection_pb2.Constraints()
    text_format.Parse(args.connection, connection_parameters)
    text_format.Parse(args.constraints, constraints)
    _logger.info('Creating consumer')
    with consumer.Consumer(connection_parameters=connection_parameters,
                           constraints=constraints) as cns:
        _logger.info('Creating worker')
        with worker.Worker(jobs=args.jobs,
                           tmpdir=args.tmpdir,
                           repository_config=args.repository_config) as wrk:
            cns.listen_and_wait(wrk.callback)


if __name__ == '__main__':
    main()
```

worker/cli.py

```
from concurrent import futures
import importlib
import logging
import tempfile
import traceback
```

```
import bunsan.pm

from bunsan.broker.worker import sender
from bunsan.broker import protocol_pb2
import bunsan.broker.worker.imp

_THREADS_PER_JOB = 3


class WorkerError(RuntimeError):
    pass


class UnknownWorkerError(WorkerError):
    pass


class Worker(object):

    def __init__(self, jobs, tmpdir, repository_config):
        self._logger = logging.getLogger(__name__)
        self._tmpdir = tmpdir
        self._repository = bunsan.pm.Repository(repository_config)
        self._executor = futures.ThreadPoolExecutor(
            max_workers=_THREADS_PER_JOB * jobs)

    def _extract(self, status_sender, tmpdir, package):
        status_sender.send('EXTRACTING')
        self._logger.debug('Extracting to %s...', tmpdir)
        self._repository.extract(package, tmpdir)
        status_sender.send('EXTRACTED')

    def _get_actual_worker(self, worker):
        if worker.find('.') != -1:
            raise UnknownWorkerError('Worker name should not contain "."')
        imp = importlib.import_module('bunsan.broker.worker.imp.' + worker)
        return imp.Worker(executor=self._executor)

    def _execute(self, status_sender, worker, tmpdir, data):
        self._logger.debug('Running')
```

```
47          return worker(status_sender=status_sender,
                          root=tmpdir,
49                        data=data)

51      def callback(self, task, send_status):
            self._logger.debug('task = %s', task)
53          result = protocol_pb2.Result()
            status_sender = sender.StatusSender(send_status)
55          try:
                with tempfile.TemporaryDirectory(dir=self._tmpdir) as tmpdir:
57                  self._extract(status_sender=status_sender,
                                  tmpdir=tmpdir,
59                                package=task.package)
                    try:
61                      actual_worker = self._get_actual_worker(task.worker)
                    except Exception as e:
63                      raise UnknownWorkerError() from e
                    result = self._execute(status_sender=status_sender,
65                                         worker=actual_worker,
                                           tmpdir=tmpdir,
67                                         data=task.data)
                status_sender.send('DONE')
69          except UnknownWorkerError:
                self._logger.exception('')
71              result.status = protocol_pb2.Result.UNKNOWN_WORKER
                result.reason = traceback.format_exc()
73          except WorkerError:
                self._logger.exception('')
75              result.status = protocol_pb2.Result.WORKER_ERROR
                result.reason = traceback.format_exc()
77          except bunsan.pm.Error:
                self._logger.exception('')
79              result.status = protocol_pb2.Result.PACKAGE_ERROR
                result.reason = traceback.format_exc()
81          except Exception:
                self._logger.exception('')
83              result.status = protocol_pb2.Result.ERROR
                result.reason = traceback.format_exc()
85          return result
```

```
    def cleanup(self):
        self._executor.shutdown()

    def __enter__(self):
        return self

    def __exit__(self, type, value, tb):
        self.cleanup()
```

worker/worker.py

```
from bunsan.broker import protocol_pb2


class StatusSender(object):

    def __init__(self, send_status):
        self._send_status = send_status

    def send(self, reason, code=None, data=None):
        status = protocol_pb2.Status()
        status.reason = reason
        if code is not None:
            status.code = code
        if data is not None:
            status.data = data
        self.send_proto(status)

    def send_proto(self, status):
        self._send_status(status)
```

worker/sender.py

```
import base64
import logging
import subprocess

from bunsan.broker import protocol_pb2


_EXECUTABLE = 'bin/bacs_system_single_worker'
_MINUTE = 60
```

```
_TIMEOUT = 10 * _MINUTE
_KiB = 1024
_MiB = 1024 * _KiB
_MAX_LINE_SIZE = 50 * _MiB


class SolutionRunner(object):

    def __init__(self, status_sender, data):
        self._logger = logging.getLogger(__name__)
        self._status_sender = status_sender
        self._data = data

    def write_data(self, stdin):
        stdin.write(self._data)
        stdin.close()

    def read_data(self, stdout):
        status = protocol_pb2.Status()
        result = protocol_pb2.Result()
        for line in stdout:
            tokens = line.split()
            if len(tokens) != 2:
                self._logger.warning(
                    'Invalid number of token: expected 2, found %d',
                    len(tokens)
                )
            else:
                command, data = tuple(tokens)
                data = base64.decodebytes(data)
                if command == b'status':
                    status.ParseFromString(data)
                    self._status_sender.send_proto(status)
                elif command == b'result':
                    result.ParseFromString(data)
                else:
                    self._logger.warning('Invalid command: %s', command)
        return result

    def read_log(self, stderr):
```

```
        return stderr.read()

class Worker(object):

    def __init__(self, executor):
        self._logger = logging.getLogger(__name__)
        self._executor = executor

    def __call__(self, status_sender, root, data):
        runner = SolutionRunner(status_sender, data)
        with subprocess.Popen(args=[_EXECUTABLE],
                              cwd=root,
                              stdin=subprocess.PIPE,
                              stdout=subprocess.PIPE,
                              stderr=subprocess.PIPE) as proc:
            status_sender.send('EXECUTING')
            self._executor.submit(runner.write_data, proc.stdin)
            result_future = self._executor.submit(runner.read_data, proc.stdout)
            log_future = self._executor.submit(runner.read_log, proc.stderr)
            ret = proc.wait()
            if ret != 0:
                result = protocol_pb2.Result()
                result.status = protocol_pb2.Result.EXECUTION_ERROR
                status_sender.send('FAIL', code=1)
            else:
                result = result_future.result()
                status_sender.send('DONE')
            result.log = log_future.result()
            return result
```

worker/imp/bacs_single.py

worker/imp/__init__.py

worker/__init__.py

```
import logging
import pika

from bunsan.broker.rabbit import connection_pb2
```

```
5
7  class Sender(object):
9      def __init__(self, channel, queue, identifier, durable=False):
           self._logger = logging.getLogger(__name__)
11         self._channel = channel
           self._queue = queue
13         self._identifier = identifier
           self._properties = pika.BasicProperties(
15             correlation_id=identifier,
               delivery_mode=(2 if durable else 1),
17         )
19     def send(self, body):
           if self._queue:
21             self._logger.debug('Sending message to %s', self._queue)
               self._channel.basic_publish(exchange='',
23                                         routing_key=self._queue,
                                           body=body,
25                                         properties=self._properties)
           else:
27             self._logger.debug('No queue to route message')
29     def sendmsg(self, format, *args):
           self.send((format % args).encode('utf8'))
31
33 class ProtoSender(Sender):
35     def send_proto(self, proto):
           self.send(proto.SerializeToString())
37
39 class StatusSender(ProtoSender):
41     def __init__(self, *args, **kwargs):
           kwargs['durable'] = False
43         super(StatusSender, self).__init__(*args, **kwargs)
```

```
    def send_status(self, code, reason=None, data=None):
        status = connection_pb2.RabbitStatus()
        status.identifier = self._identifier
        status.status.code = code
        if reason is not None:
            status.status.reason = reason
        if data is not None:
            status.status.date = data
        self.send_proto(status)


class ResultSender(ProtoSender):

    def __init__(self, *args, **kwargs):
        kwargs['durable'] = True
        super(ResultSender, self).__init__(*args, **kwargs)

    def send_result(self, status, reason=None, data=None):
        result = connection_pb2.RabbitResult()
        result.identifier = self._identifier
        result.result.status = status
        if reason is not None:
            result.result.reason = reason
        if data is not None:
            result.result.data = data
        self.send_proto(result)


class ErrorSender(Sender):

    def __init__(self, channel, properties):
        super(ErrorSender, self).__init__(channel=channel,
                                          queue=properties.reply_to,
                                          identifier=properties.correlation_id,
                                          durable=True)
```

service/sender.py

```
import logging
import pika
import threading
```

```
import time

from bunsan.broker.service import sender
from bunsan.broker.rabbit import connection_pb2


_RETRY_TIME = 5


class Consumer(object):

    def __init__(self, connection_parameters, constraints):
        self._logger = logging.getLogger(__name__)
        connect = dict()
        if connection_parameters.host:
            connect['host'] = connection_parameters.host
        if connection_parameters.port:
            connect['port'] = connection_parameters.port
        if connection_parameters.virtual_host:
            connect['virtual_host'] = connection_parameters.virtual_host
        if connection_parameters.HasField('credentials'):
            connect['credentials'] = pika.credentials.PlainCredentials(
                username=connection_parameters.credentials.username,
                password=connection_parameters.credentials.password)
        self._logger.debug('Opening connection')
        self._connection_parameters = pika.ConnectionParameters(**connect)
        self._connection = None
        self._constraints = constraints
        self._callback = None
        self._thread = None
        self._logger.debug('Created consumer')

    def listen(self, callback):
        """
            Args:
                callback(task, send_status(Status)) -> Result
        """
        self._logger.info('Start asynchronous consuming')
        self._callback = callback
        self._thread = threading.Thread(target=self._start_consuming)
```

```
        self._thread.start()
45
    def listen_and_wait(self, callback):
        """
            Args:
47              callback(task, send_status(Status)) -> Result
        """
        self._logger.info('Start synchronous consuming')
51      self._callback = callback
        self._start_consuming()
53
    def wait(self):
55      self._thread.join()

    def close(self):
57      self._logger.info('Closing connection to RabbitMQ')
        if self._connection is not None:
59          self._connection.close()
        if self._thread is not None:
61          self._thread.join()

    def _connect(self):
63      while self._connection is None:
            try:
65              self._logger.info('Connecting to RabbitMQ')
                self._connection = pika.BlockingConnection(
67                  self._connection_parameters)
                self._logger.info('Connected to RabbitMQ')
69          except pika.exceptions.AMQPConnectionError:
                self._logger.exception(
71                  'Unable to connect to RabbitMQ, retrying')
                time.sleep(_RETRY_TIME)
73
    def _start_consuming(self):
75      while True:
            try:
77              self._connect()
                channel = self._connection.channel()
79              channel.basic_qos(prefetch_count=1)
                self._logger.debug('Start consuming')
81
```

```
                for queue in self._constraints.resource:
85                  self._logger.debug('Consuming queue=%s', queue)
                    channel.queue_declare(queue=queue, durable=True)
87                  channel.basic_consume(queue=queue,
                                          consumer_callback=self._consume)
89              channel.start_consuming()
            except pika.exceptions.AMQPConnectionError:
91              self._logger.exception(
                    'Broken connection to RabbitMQ, retrying')
93              self._connection = None

95      def _consume(self, channel, method, properties, body):
            """Only commit logic, does not throw"""
97          try:
                self._do_consume(channel, method, properties, body)
99              self._logger.info('Acknowledging: %s',
                                  properties.correlation_id)
101             channel.basic_ack(delivery_tag=method.delivery_tag)
            except Exception:
103             self._logger.info('Not acknowledging: %s',
                                  properties.correlation_id)
105             channel.basic_nack(delivery_tag=method.delivery_tag,
                                   requeue=False)
107
        def _do_consume(self, channel, method, properties, body):
109         """Actual consume implementation, may throw exceptions"""
            self._logger.info('Received task: %s', properties.correlation_id)
111         error_sender = sender.ErrorSender(channel, properties)
            rabbit_task = connection_pb2.RabbitTask()
113         try:
                rabbit_task.ParseFromString(body)
115         except Exception as e:
                self._logger.exception('Unable to parse task proto')
117             error_sender.sendmsg('Unable to parse task proto: %s', e)
                raise
119         status_sender = sender.StatusSender(channel,
                                                rabbit_task.status_queue,
121                                             rabbit_task.identifier)
            result_sender = sender.ResultSender(channel,
123                                             rabbit_task.result_queue,
```

```
                                            rabbit_task.identifier)
125         self._logger.debug('Running callback')

127         def send_status(status):
                rabbit_status = connection_pb2.RabbitStatus()
129             rabbit_status.identifier = rabbit_task.identifier
                rabbit_status.status.CopyFrom(status)
131             status_sender.send_proto(rabbit_status)
            try:
133             result = self._callback(task=rabbit_task.task,
                                        send_status=send_status)
135             self._logger.debug('Completed callback')
            except Exception as e:
137             self._logger.exception('Unable to complete callback')
                error_sender.sendmsg('Unable to complete callback: %s', e)
139             raise
            try:
141             rabbit_result = connection_pb2.RabbitResult()
                rabbit_result.identifier = rabbit_task.identifier
143             rabbit_result.result.CopyFrom(result)
                result_sender.send_proto(rabbit_result)
145             self._logger.info('Sent result')
            except Exception as e:
147             self._logger.exception('Unable to serialize result proto')
                error_sender.sendmsg('Unable to serialize result proto: %s', e)
149             raise

151     def __enter__(self):
            return self
153
        def __exit__(self, type, value, tb):
155         self.close()
```

service/consumer.py

service/__init__.py

__init__.py

1.3. Реализация на языке программирования C#

```
using System;
using System.Text;
using System.Threading;
using NDesk.Options;

namespace Bunsan.Broker.ClientExample
{
    class Program
    {
        static int Main(string[] args)
        {
            var connection_parameters = new ConnectionParameters();
            var constraints = new Constraints { Resource = new[] { ”resource” }
                };
            var task = new Task();
            int interval = 5000;
            int number_of_messages = 10;
            Action ensure_credentials = () => { if (connection_parameters.
                Credentials == null) connection_parameters.Credentials = new
                Credentials(); };
            var options = new OptionSet
            {
                {”identifier=”, ””, identifier => connection_parameters.
                    Identifier = identifier},
                {”host=”, ””, host => connection_parameters.Host = host},
                {”port=”, ””, port => connection_parameters.Port = int.Parse(
                    port)},
                {”virtual-host=”, ””, vhost => connection_parameters.VirtualHost
                     = vhost},
                {”username=”, ””, username => { ensure_credentials();
                    connection_parameters.Credentials.UserName = username; } },
                {”password=”, ””, password => { ensure_credentials();
                    connection_parameters.Credentials.Password = password; } },
                {”resource=”, ””, resource => constraints.Resource[0] = resource
                    },
                {”worker=”, ””, worker => task.Worker = worker},
                {”package=”, ””, package => task.Package = package},
                {”data=”, ””, data => task.Data = Encoding.UTF8.GetBytes(data)},
```

```
            {"interval=", "", interval_str => interval = int.Parse(
                interval_str)},
            {"number=", "", number => number_of_messages = int.Parse(number)
                },
        };
        try
        {
            options.Parse(args);
        }
        catch (OptionException e)
        {
            Console.WriteLine("Argument error: {0}", e.Message);
        }
        var listener = new ClientListener(connection_parameters);
        var sender = new ClientSender(connection_parameters);
        listener.Listen((id, status) =>
        {
            Console.WriteLine("Got status: id = {0}, status = [{1}, {2}] {3}
                ",
                              id, status.Code, status.Reason,
                              status.Data != null ? Encoding.ASCII.GetString
                                  (status.Data) : "null");
        }, (id, result) =>
        {
            Console.WriteLine("Got result: id = {0}, result = [{1}, {2}] {3}
                ",
                              id, result.Status, result.Reason,
                              result.Data != null ? Encoding.ASCII.GetString
                                  (result.Data) : "null");
        }, (id, error) =>
        {
            Console.WriteLine("Got error: id = {0}, error = {1}", id, error)
                ;
        });
        for (int id = 0; id < number_of_messages; ++id)
        {
            sender.Send(constraints, id.ToString(), task);
            Thread.Sleep(interval);
        }
        listener.Close();
```

```
            sender.Close();
            return 0;
        }
    }
}
```

Bunsan.Broker.ClientExample/Program.cs

```
using System.Reflection;
using System.Runtime.InteropServices;

// General Information about an assembly is controlled through the following
// set of attributes. Change these attribute values to modify the information
// associated with an assembly.
[assembly: AssemblyTitle(”Bunsan.Broker.ClientExample”)]
[assembly: AssemblyDescription(””)]
[assembly: AssemblyConfiguration(””)]
[assembly: AssemblyCompany(””)]
[assembly: AssemblyProduct(”Bunsan.Broker.ClientExample”)]
[assembly: AssemblyCopyright(”Copyright ©  2015”)]
[assembly: AssemblyTrademark(””)]
[assembly: AssemblyCulture(””)]

// Setting ComVisible to false makes the types in this assembly not visible
// to COM components.  If you need to access a type in this assembly from
// COM, set the ComVisible attribute to true on that type.
[assembly: ComVisible(false)]

// The following GUID is for the ID of the typelib if this project is exposed to
    COM
[assembly: Guid(”69f070dd-14c2-4327-ab1c-c1ea741b9f59”)]

// Version information for an assembly consists of the following four values:
//
//      Major Version
//      Minor Version
//      Build Number
//      Revision
//
// You can specify all the values or you can default the Build and Revision
   Numbers
```

```
// by using the '*' as shown below:
// [assembly: AssemblyVersion("1.0.*")]
[assembly: AssemblyVersion("1.0.0.0")]
[assembly: AssemblyFileVersion("1.0.0.0")]
```

Bunsan.Broker.ClientExample/Properties/AssemblyInfo.cs

```
using ProtoBuf;

namespace Bunsan.Broker.Rabbit
{
    [ProtoContract]
    class RabbitStatus
    {
        [ProtoMember(1, IsRequired = false)]
        public string Identifier { get; set; }

        [ProtoMember(2, IsRequired = false)]
        public Status Status { get; set; }
    }
}
```

Bunsan.Broker/Rabbit/RabbitStatus.cs

```
using ProtoBuf;

namespace Bunsan.Broker.Rabbit
{
    [ProtoContract]
    class RabbitTask
    {
        [ProtoMember(1, IsRequired = false)]
        public string Identifier { get; set; }

        [ProtoMember(2, IsRequired = false)]
        public Task Task { get; set; }

        [ProtoMember(3, IsRequired = false)]
        public Constraints Constraints { get; set; }

        [ProtoMember(4, IsRequired = false)]
        public string StatusQueue { get; set; }
```

```
20          [ProtoMember(5, IsRequired = false)]
            public string ResultQueue { get; set; }
22
            [ProtoMember(6, IsRequired = false)]
24          public string ErrorQueue { get; set; }
        }
26  }
```

Bunsan.Broker/Rabbit/RabbitTask.cs

```
    using ProtoBuf;
2
    namespace Bunsan.Broker.Rabbit
4   {
        [ProtoContract]
6       class RabbitResult
        {
8           [ProtoMember(1, IsRequired = false)]
            public string Identifier { get; set; }
10
            [ProtoMember(2, IsRequired = false)]
12          public Result Result { get; set; }
        }
14  }
```

Bunsan.Broker/Rabbit/RabbitResult.cs

```
    using System;
2   using System.Threading;
    using RabbitMQ.Client;
4   using RabbitMQ.Client.Exceptions;

6   namespace Bunsan.Broker
    {
8       public class Connection : IDisposable
        {
10          private readonly ConnectionFactory connection_factory;
            protected bool running = true;
12          private IConnection connection;
            protected IModel channel;
14
```

```
        protected object Lock { get { return connection_factory; } }
16
        public IModel Channel
18      {
            get
20          {
                lock (Lock)
22              {
                    reconnect();
24                  return channel;
                }
26          }
        }
28
        public bool IsRunning
30      {
            get
32          {
                lock (Lock)
34              {
                    return running;
36              }
            }
38      }

40      public bool IsClosed { get { return !IsRunning; } }

42      public event Action<IModel> OnConnect = delegate { };

44      public Connection(ConnectionParameters parameters)
        {
46          connection_factory = new ConnectionFactory();
            if (parameters.Host != null) connection_factory.HostName =
               parameters.Host;
48          if (parameters.Port != 0) connection_factory.Port = parameters.Port;
            if (parameters.VirtualHost != null) connection_factory.VirtualHost =
                parameters.VirtualHost;
50          if (parameters.Credentials != null)
            {
52              var credentials = parameters.Credentials;
```

```
            if (credentials.UserName != null) connection_factory.UserName =
                credentials.UserName;
            if (credentials.Password != null) connection_factory.Password =
                credentials.Password;
        }
    }

    public void Close()
    {
        lock (Lock)
        {
            running = false;
            if (connection != null && connection.IsOpen)
                connection.Close();
        }
    }

    private void connect()
    {
        connection = connection_factory.CreateConnection();
        channel = connection.CreateModel();
        OnConnect(channel);
    }

    private void reconnect()
    {
        if (!running) return;
        while (connection == null || !connection.IsOpen)
        {
            try
            {
                connect();
            }
            catch (BrokerUnreachableException)
            {
                Thread.Sleep(5000);
            }
        }
    }

```

```
        // Connect if necessary, i.e. not closed and not connected
        public void Reconnect()
        {
            lock (Lock)
            {
                reconnect();
            }
        }

        public void Dispose()
        {
            Close();
        }
    }
}
```

Bunsan.Broker/Connection.cs

```
using ProtoBuf;

namespace Bunsan.Broker
{
    [ProtoContract]
    public class Status
    {
        [ProtoMember(1, IsRequired = false)]
        public int Code { get; set; }

        [ProtoMember(2, IsRequired = false)]
        public string Reason { get; set; }

        [ProtoMember(3, IsRequired = false)]
        public byte[] Data { get; set; }
    }
}
```

Bunsan.Broker/Status.cs

```
using ProtoBuf;

namespace Bunsan.Broker
{
```

```
    [ProtoContract]
    public class Credentials
    {
        [ProtoMember(1, IsRequired = false)]
        public string Method { get; set; }

        [ProtoMember(2, IsRequired = false)]
        public string UserName { get; set; }

        [ProtoMember(3, IsRequired = false)]
        public string Password { get; set; }

        [ProtoMember(4, IsRequired = false)]
        public byte[] Certificate { get; set; }

        [ProtoMember(5, IsRequired = false)]
        public byte[] Key { get; set; }
    }
}
```

Bunsan.Broker/Credentials.cs

```
using System;
using System.IO;
using System.Text;
using System.Threading;
using Bunsan.Broker.Rabbit;
using ProtoBuf;
using RabbitMQ.Client;
using RabbitMQ.Client.Events;

namespace Bunsan.Broker
{
    public class ClientListener : ClientBase
    {
        private delegate void MessageCallback(BasicDeliverEventArgs message);
        private delegate void ReconnectCallback();

        private class Listener : DefaultBasicConsumer
        {
            private readonly MessageCallback message_callback;
```

```
        private readonly ErrorCallback error_callback;
21      private readonly ReconnectCallback reconnect_callback;

23      public Listener(IModel model,
                        MessageCallback message_callback,
25                      ErrorCallback error_callback,
                        ReconnectCallback reconnect_callback)
27          : base(model)
        {
29          this.message_callback = message_callback;
            this.error_callback = error_callback;
31          this.reconnect_callback = reconnect_callback;
        }
33
        private void call_message(object message_object)
35      {
            var message = (BasicDeliverEventArgs)message_object;
37          try
            {
39              message_callback(message);
            }
41          catch (Exception e)
            {
43              try
                {
45                  error_callback(message.BasicProperties.CorrelationId, e.
                        ToString());
                }
47              catch (Exception) { /* ignore */ }
            }
49      }

51      private void call_reconnect(object ignored)
        {
53          reconnect_callback();
        }
55
        public override void HandleBasicDeliver(string consumerTag,
57                                                ulong deliveryTag,
                                                  bool redelivered,
```

```
                                                string exchange,
                                                string routingKey,
                                                IBasicProperties properties,
                                                byte[] body)
            {
                var message = new BasicDeliverEventArgs
                {
                    ConsumerTag = consumerTag,
                    DeliveryTag = deliveryTag,
                    Redelivered = redelivered,
                    Exchange = exchange,
                    RoutingKey = routingKey,
                    BasicProperties = properties,
                    Body = body,
                };
                ThreadPool.QueueUserWorkItem(call_message, message);
            }

            public override void HandleModelShutdown(object model,
               ShutdownEventArgs reason)
            {
                ThreadPool.QueueUserWorkItem(call_reconnect);
            }
        };

        private class ListenerFactory
        {
            private readonly StatusCallback status_callback;
            private readonly ResultCallback result_callback;
            private readonly ErrorCallback error_callback;
            private readonly ReconnectCallback reconnect_callback;

            public ListenerFactory(StatusCallback status_callback,
                                   ResultCallback result_callback,
                                   ErrorCallback error_callback,
                                   ReconnectCallback reconnect_callback)
            {
                this.status_callback = status_callback;
                this.result_callback = result_callback;
                this.error_callback = error_callback;
```

```
            this.reconnect_callback = reconnect_callback;
        }

        public Listener MakeStatusListener(IModel channel)
        {
            return new Listener(channel, message =>
            {
                // we don't care about error handling of Status messages
                using (var stream = new MemoryStream(message.Body))
                {
                    var status = Serializer.Deserialize<RabbitStatus>(stream
                        );
                    // note: message was already acked, exceptions can be
                        ignored
                    status_callback(status.Identifier, status.Status);
                }
            }, error_callback, reconnect_callback);
        }

        public Listener MakeResultListener(IModel channel)
        {
            return new Listener(channel, message =>
            {
                try
                {
                    using (var stream = new MemoryStream(message.Body))
                    {
                        var result = Serializer.Deserialize<RabbitResult>(
                            stream);
                        result_callback(result.Identifier, result.Result);
                    }
                }
                catch (Exception)
                {
                    // message is either invalid or can't be processed by
                        result_callback
                    // exception will cause error_callback to be called
                    // we do not want to requeue message since it may cause
                        infinite try-exception loop
                    channel.BasicNack(message.DeliveryTag, false, false);
```

```
                    throw;
                }
                // commit phase
                channel.BasicAck(message.DeliveryTag, false);
            }, error_callback, reconnect_callback);
        }

        public Listener MakeErrorListener(IModel channel)
        {
            return new Listener(channel, message =>
            {
                error_callback(message.BasicProperties.CorrelationId,
                               Encoding.UTF8.GetString(message.Body));
                // commit phase
                channel.BasicAck(message.DeliveryTag, false);
            }, error_callback, reconnect_callback);
        }
    }

    private ListenerFactory listener_factory;

    public ClientListener(ConnectionParameters parameters)
        : base(parameters)
    {
        connection.OnConnect += rebind;
    }

    ~ClientListener()
    {
        connection.OnConnect -= rebind;
    }

    private void rebind(IModel channel)
    {
        if (listener_factory == null) return;
        var status_listener = listener_factory.MakeStatusListener(channel);
        var result_listener = listener_factory.MakeResultListener(channel);
        var error_listener = listener_factory.MakeErrorListener(channel);
        channel.QueueDeclare(queue: StatusQueue, durable: false, exclusive:
           false, autoDelete: true, arguments: null);
```

```
            channel.QueueDeclare(queue: ResultQueue, durable: true, exclusive:
                false, autoDelete: false, arguments: null);
            channel.QueueDeclare(queue: ErrorQueue, durable: true, exclusive:
                false, autoDelete: false, arguments: null);
            channel.BasicConsume(queue: StatusQueue, noAck: true, consumer:
                status_listener);
            channel.BasicConsume(queue: ResultQueue, noAck: false, consumer:
                result_listener);
            channel.BasicConsume(queue: ErrorQueue, noAck: false, consumer:
                error_listener);
        }

        public void Listen(StatusCallback status_callback,
                           ResultCallback result_callback,
                           ErrorCallback error_callback)
        {
            listener_factory = new ListenerFactory(status_callback,
                result_callback, error_callback, connection.Reconnect);
            // force rebind for existing connection
            rebind(connection.Channel);
        }
    }
}
```

Bunsan.Broker/ClientListener.cs

```
using System;
using System.IO;
using Bunsan.Broker.Rabbit;
using ProtoBuf;
using RabbitMQ.Client.Exceptions;

namespace Bunsan.Broker
{
    public class ClientSender : ClientBase
    {
        public ClientSender(ConnectionParameters parameters) : base(parameters)
            { }

        public void Send(Constraints constraints, string id, Task task)
        {
```

```
            var rabbit_task = new RabbitTask
16          {
                Identifier = id,
18              Task = task,
                Constraints = constraints,
20              ResultQueue = ResultQueue,
                StatusQueue = StatusQueue,
22          };
            byte[] data;
24          using (var stream = new MemoryStream())
            {
26              Serializer.Serialize(stream, rabbit_task);
                data = new byte[stream.Length];
28              stream.Seek(0, SeekOrigin.Begin);
                stream.Read(data, 0, data.Length);
30          }
            if (constraints.Resource.Length != 1)
32              throw new NotImplementedException(”Supports only single resource
                    constraint”);
            for (; ; )
34          {
                try
36              {
                    var properties = connection.Channel.CreateBasicProperties();
38                  properties.ReplyTo = ErrorQueue;
                    properties.CorrelationId = id;
40                  properties.DeliveryMode = 2;  // persistent
                    if (connection.IsClosed) throw new InvalidOperationException
                       (”Already closed”);
42                  connection.Channel.BasicPublish(exchange: ””,
                                                    routingKey: constraints.
                                                       Resource[0],
44                                                  basicProperties: properties,
                                                    body: data);
46                  break;
                }
48              catch (AlreadyClosedException)
                {
50                  // retry
                }
```

```
            }
        }
    }
}
```

Bunsan.Broker/ClientSender.cs

```
using ProtoBuf;

namespace Bunsan.Broker
{
    [ProtoContract]
    public class Task
    {
        [ProtoMember(1, IsRequired = false)]
        public string Worker { get; set; }

        [ProtoMember(2, IsRequired = false)]
        public string Package { get; set; }

        [ProtoMember(3, IsRequired = false)]
        public byte[] Data { get; set; }
    }
}
```

Bunsan.Broker/Task.cs

```
using ProtoBuf;

namespace Bunsan.Broker
{
    [ProtoContract]
    public enum ResultStatus
    {
        [ProtoEnum]
        OK = 0,
        [ProtoEnum]
        ERROR = 1,

        // proto errors
        [ProtoEnum]
        PROTO_ERROR = 100,
```

```
        // worker errors
        [ProtoEnum]
        WORKER_ERROR = 200,
        [ProtoEnum]
        UNKNOWN_WORKER = 201,
        [ProtoEnum]
        INCOMPATIBLE_PACKAGE = 202,

        // package errors
        [ProtoEnum]
        PACKAGE_ERROR = 300,
        [ProtoEnum]
        PACKAGE_NOT_FOUND = 301,
        [ProtoEnum]
        PACKAGE_BUILD_ERROR = 302,

        // execution errors
        [ProtoEnum]
        EXECUTION_ERROR = 400,
        [ProtoEnum]
        EXECUTION_TIMEOUT = 401,
    }

    [ProtoContract]
    public class Result
    {
        [ProtoMember(1, IsRequired = false)]
        public ResultStatus Status { get; set; }

        [ProtoMember(2, IsRequired = false)]
        public string Reason { get; set; }

        // if status == OK
        [ProtoMember(3, IsRequired = false)]
        public byte[] Data { get; set; }

        [ProtoMember(4, IsRequired = false)]
        public byte[] Log { get; set; }
    }
```

```
}
```

Bunsan.Broker/Result.cs

```
using ProtoBuf;

namespace Bunsan.Broker
{
    [ProtoContract]
    public class Constraints
    {
        [ProtoMember(1, IsRequired = false)]
        public string[] Resource { get; set; }
    }
}
```

Bunsan.Broker/Constraints.cs

```
using ProtoBuf;

namespace Bunsan.Broker
{
    [ProtoContract]
    public class ConnectionParameters
    {
        [ProtoMember(1, IsRequired = false)]
        public string Identifier { get; set; }

        [ProtoMember(2, IsRequired = false)]
        public Credentials Credentials { get; set; }

        [ProtoMember(3, IsRequired = false)]
        public string Host { get; set; }

        [ProtoMember(4, IsRequired = false)]
        public int Port { get; set; }

        [ProtoMember(5, IsRequired = false)]
        public string VirtualHost { get; set; }
    }
}
```

Bunsan.Broker/ConnectionParameters.cs

```
using System;

namespace Bunsan.Broker
{
    public abstract class ClientBase : IDisposable
    {
        public string StatusQueue { get; set; }
        public string ResultQueue { get; set; }
        public string ErrorQueue { get; set; }
        protected readonly Connection connection;

        public ClientBase(ConnectionParameters parameters)
        {
            connection = new Connection(parameters);
            if (string.IsNullOrEmpty(parameters.Identifier)) throw new
               ArgumentException("Expected non-empty Identifier");
            StatusQueue = "client." + parameters.Identifier + ".status";
            ResultQueue = "client." + parameters.Identifier + ".result";
            ErrorQueue = "client." + parameters.Identifier + ".error";
        }

        public delegate void StatusCallback(string id, Status status);
        public delegate void ResultCallback(string id, Result result);
        public delegate void ErrorCallback(string id, string error);

        public void Close()
        {
            connection.Close();
        }

        public void Dispose()
        {
            Close();
        }
    }
}
```

Bunsan.Broker/ClientBase.cs

```
using System.Reflection;
using System.Runtime.InteropServices;
```

```
3
// General Information about an assembly is controlled through the following
5 // set of attributes. Change these attribute values to modify the information
// associated with an assembly.
7 [assembly: AssemblyTitle(”Bunsan.Broker”)]
[assembly: AssemblyDescription(””)]
9 [assembly: AssemblyConfiguration(””)]
[assembly: AssemblyCompany(””)]
11 [assembly: AssemblyProduct(”Bunsan.Broker”)]
[assembly: AssemblyCopyright(”Copyright ©  2015”)]
13 [assembly: AssemblyTrademark(””)]
[assembly: AssemblyCulture(””)]
15
// Setting ComVisible to false makes the types in this assembly not visible
17 // to COM components.  If you need to access a type in this assembly from
// COM, set the ComVisible attribute to true on that type.
19 [assembly: ComVisible(false)]

21 // The following GUID is for the ID of the typelib if this project is exposed to
    COM
[assembly: Guid(”a927d6c3-b4d8-4330-ab0b-82dea8bffc42”)]
23
// Version information for an assembly consists of the following four values:
25 //
//      Major Version
27 //      Minor Version
//      Build Number
29 //      Revision
//
31 // You can specify all the values or you can default the Build and Revision
   Numbers
// by using the ’*’ as shown below:
33 // [assembly: AssemblyVersion(”1.0.*”)]
[assembly: AssemblyVersion(”1.0.0.0”)]
35 [assembly: AssemblyFileVersion(”1.0.0.0”)]
```

Bunsan.Broker/Properties/AssemblyInfo.cs