МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное АВТОНОМНОЕ образовательное учреждение высшего образования

**Национальный исследовательский ядерный университет «МИФИ»**



к учебно-исследовательской работе студента на тему:

Применение RPC как класса технологий для организации сетевого взаимодействия компонентов распределенной системы

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Группа | |  | |  | | |
| Студент | |  | |  | Белозёров Александр Александрович | |
|  | | (подпись) | | (ФИО) | | |
| Руководитель | |  | |  | Макаров Дмитрий Сергеевич | |
|  | | (подпись) | | (ФИО) | | |
| Научный консультант | |  | |  | Саманчук Владимир Никифорович | |
|  | | (подпись) | | (ФИО) | | |
| Оценка руководителя |  | | Оценка консультанта | | |  |
|  | (0-5 баллов) | |  | | | (0-5 баллов) |
|  |  | |  | | |  |
| Оценка за оформление |  | | Оценка за структуру | | |  |
|  | (0-5 баллов) | |  | | | (0-5 баллов) |

**Москва 2017**

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

**«Национальный исследовательский ядерный университет «МИФИ»**

Студенту гр. Б14-501 Белозёрову Александру Александровичу

(группа) (фио)

**ТЕМА УИР**

|  |
| --- |
| Применение RPC как класса технологий для организации сетевого взаимодействия компонентов распределенной системы |

**ЗАДАНИЕ**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| №  п/п | Содержание работы | Форма отчетности | Срок исполне­ния | Отметка руководителя о выполнении |
|  | **Аналитическая часть** |  |  |  |
| 1.1 | Изучение существующих методик сравнения реализаций класса технологий | Аналитический отчет | 23.02.17 |  |
| 1.2 | Сравнительный анализ реализаций RPC | Аналитический отчет | 09.03.17 |  |
| 1.3 | Выявление и обоснование критериев применимости реализаций RPC | Аналитический отчет | 23.03.17 |  |
| 1.4 | *Оформление расширенного содержания пояснительной записки (РСПЗ)* | Текст РСПЗ | 27.03.17 |  |
|  | **Теоретическая часть** |  |  |  |
| 2.1 | Разработка методик сравнения реализаций RPC по критериям применимости | Текст ПЗ | 06.04.17 |  |
| 2.2 | Разработка методик проведения испытаний | Текст ПЗ | 13.04.17 |  |
| 2.3 | Разработка методика оценки результатов проведения испытаний | Текст ПЗ | 20.04.17 |  |
|  | **Инженерно-технологическая часть** |  |  |  |
| 3.1 | Реализация разработанных методик и получение результатов сравнения реализаций RPC на основе критериев применимости | Исполняемые файлы, исходный код | 27.04.17 |  |
| 3.2 | Выводы на основе результатов сравнительных испытаний | Текст ПЗ | 11.05.17 |  |
|  | *Оформление пояснительной записки (ПЗ) и иллюстративного материала для доклада*. | Текст ПЗ, презентация | 15.05.17 |  |

**ЛИТЕРАТУРА**

|  |  |  |
| --- | --- | --- |
|  | Nicolai M. Josuttis "The C++ Standard Library: A Tutorial and Reference (2nd Edition)".-US, Addison\_Wesley, 2012 | |
|  | Matthias Kalle Dalheimer "Programming with Qt, 2nd Edition. Writing Portable GUI applications on Unix and Win32". -US, O'Reilly Media, 2002 | |
|  | GRPC documentation [Электронный ресурс] // grpc-community webpage:  сайт. – URL:http://www.grpc.io/docs/ | |
|  | The ZeroMQ Reference Manual etc. [Электронный ресурс] //zeroMQ manuals webpage:  сайт. - URL:http://zeromq.org/intro:read-the-manual | |
|  | Thrift Tools [Электронный ресурс] // apache-community webpage:  сайт. – URL:https://thrift.apache.org/ | |
| 6. | Apache Avro™ 1.8.1 Documentation [Электронный ресурс] // apache-community webpage: сайт. – URL:http://avro.apache.org/docs/1.8.1/ | |
| Дата выдачи задания:    « » 2017г. | | Руководитель \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_(Макаров Д.С.)  (подпись) (фио)  Консультант \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_(Саманчук В.Н.)  (подпись) (фио)  Студент \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_(Белозёров А.А.)  (подпись) (фио) |

**Отчет о проверке на плагиат**

**Рецензия**

**Оглавление**

**Реферат.**

**Введение.**

**1. Анализ, выявление и обоснование критериев применимости RPC.**

1.1. Выявление и обоснование критериев применимости реализаций RPC

1.2. Анализ реализаций RPC

1.3. Описание существующих методик сравнения реализаций класса технологий

1.4. Вывод

1.5. Цели и задачи УИР

**2. Разработка методики сравнения реализаций PRC по критериям применимости.**

2.1. Методика сравнения по критериям применимости

2.2. Методика проведения испытаний

2.3. Методика оценки результатов проведения испытаний

2.4. Вывод

**3. Результаты сравнения реализаций RPC на основе критериев применимости.**

3.1. Реализация разработанной методики сравнения реализаций RPC по критериям применимости

3.2. Результаты сравнительных испытаний

3.3. Выводы

**Заключение.**

**Список литературы**

# Реферат

Пояснительная записка содержит \_ страниц (из них количество\_листингов\_кода страниц приложений). Количество использованных источников — пока\_числится\_6. Количество приложений — количество\_листингов\_кода.

Ключевые слова: сетевое взаимодействие, промежуточное программное обеспечение, распределенные системы, RPC, критерий применимости реализации RPC.

Целью данной учебно-исследовательской работы является оценка эффективности применения класса RPC для организации сетевого взаимодействия компонентов распределенной системы.

В введении описывается организация сетевого взаимодействия компонентов распределенной системы, перечисляются имеющиеся классы технологий, ориентированных на обеспечение связи между компонентами и объясняется выбор в пользу RPC.

В первом разделе работы представлен анализ реализаций RPC, выбор и обоснование критериев применимости реализаций RPC, а также рассмотрены существующие методики сравнения реализаций класса технологий.

Во втором разделе приведены описания методики сравнения реализации RPC по критериям применимости, методики проведения сравнительных испытаний и методики оценки результатов сравнительных испытаний.

В третьем разделе приведены реализация разработанной методики сравнения реализаций RPC по критериям применимости и результаты сравнительных испытаний.

# Введение

С ростом сложности вычислений и увеличении объема обрабатываемых данных все большее внимание уделяется распределенным вычислениям. Распределенные вычисления позволяют распределить нагрузку на различные программно-аппаратные модули в рамках распределенной системы. Основным этапом в процессе проведения распределенных вычислений является обмен информацией между компонентами распределенной системы.

Для организации сетевого взаимодействия компонентов распределенной системы используется комплекс программного обеспечения, называемый промежуточным программным обеспечением (ППО). ППО предоставляет набор сервисов, обращение к которым позволяет различным приложениям, в общем случае выполняющимся на разных платформах, взаимодействовать между собой. Общеприкладные интерфейсы ППО позволяют реализовать взаимодействие между приложениями, не углубляясь в инфраструктуру и детали реализации гетерогенной сети, а последующие изменения в структуре и составе такой сети не потребуют изменений в приложениях (при условии, что эти изменения не затрагивают общие прикладные интерфейсы ППО)

Существуют различные классы ППО, позволяющие организовать сетевое взаимодействие компонентов распределенной системы. Универсальным подходом при взаимодействии компонентов распределенных систем является удаленный вызов процедур (RPC).

Целью данной учебно-исследовательской работы является оценка эффективности применения класса RPC для организации сетевого взаимодействия компонентов распределенной системы. Задачи, которые необходимо выполнить для достижения поставленной цели:

1. Описание существующих методик сравнения реализаций класса технологий;
2. Анализ реализаций RPC;
3. Выявление и обоснование критериев применимости реализаций RPC;
4. Разработка методики сравнения реализаций PRC по критериям применимости;
5. Реализация разработанной методики и получение результатов сравнения реализаций RPC на основе критериев применимости.

**Глава 1. Анализ, выявление и обоснование критериев применимости RPC.**

**1.1. Выявление и обоснование критериев применимости реализаций RPC**

Оценка эффективности применения класса RPC для организации сетевого взаимодействия компонентов распределенной системы должна осуществляться на основе методик, разработанных специально для конкретной задачи, и системы объективных критериев.

В данной работе, которая посвящена вопросу применения RPC для решения вышеупомянутой задачи, рассматривают­ся следующие критерии, полагаемые как в достаточной степени независимые от базирующихся на чувствах суждений исследователя характеристики реализаций класса:

1. Скорость сериализации /десериализации данных;
2. Ресурсоемкость процесса;
3. Скорость формирования запроса, совершения удаленный вызов функции и получения из другого адресного пространства результат обработки данного запроса.

Первый критерий относится к компоненту, непосредственно связанному с языком сериализации обьектов или же структур, данный критерий имеет значение, если соответствующая реализация позволяет использовать протокол сериализации независимо от сетевой компонента (например, когда предпочтительная какой-либо иной транспортный протокол). Основным параметром, определяющим степень удовлетворения реализации критерию будет время, затраченное на сериализацию/десериализацию данных;

Второй критерий имеет значение в тех случаях, когда необходимо в рамках решения поставленной задачи налаживания взаимодействия компонент распределенной системы уложиться в некоторые рамки касательно имеющихся вычислительных мощностей и объемов памяти. Основными параметрами, определяющим степень удовлетворения реализации критерию будет загруженность системы при выполнении задач, потребляемая память.

Третий критерий имеет отношение к оценке применимости функциональности какого-то исключительно одного фреймворка для решения поставленной задачи, когда имеется возможность использовать готовые решения по эффективному взаимодействию сетевого и транспортного протоколов и, соответственно, оценивается как эффективность работы каждого из протоколов в отдельности, так и в некоторой степени — эфективность их совместной работы. Основным параметром, определяющим степень удовлетворения реализации критерию будет время, затраченное на описанный запрос-ответ.

Помимо всего прочего, имеют место быть и частично объективные, частично субъективные характеристики, из которых, однако, тоже можно извлечь пользу, такие как:

1. Удобство пользования предоставляемыми инструментами
2. Возможность выбора конкретного языка программирования для использования этих самых инструментов

Несмотря на кажущуюся относительность возможных рассуждений, основывающихся на данных критериях, первый критерий, касающийся наличия документации к реализации, ее грамотного составления и полноты, сложность кода, который необходимо написать для использования той или иной части функциональности фреймворка — хоть и трудноизмерим, но возможен при сравнительном анализе реализаций и имеет большой вес; второй же критерий может определять возможность использования реализации в той или иной сфере, которые подразумевают разную эффективность функционирования конечного продукта в зависимости от языка — что связано опять же с удобством использования языка конкретной группой программистов, возможность его применения на конкретном аппаратном обеспечении, если язык машинно-ориентирован и т. п.

**1.2. Анализ реализаций RPC**

В данной работе рассматривались следующие реализации класса RPC:

1. gRPC —  высокопроизводительный фреймворк, разработанный компанией Google для вызов удаленных процедур, работающий поверх HTTP/2.

gRPC прост в использовании, подходит для создания распределенных систем (микросервисов) и API. Имеет встроенную поддержку для балансировки нагрузки, трассировки, аутентификации и проверки жизнеспособности сервисов. Имеется возможность создавать клиентские библиотеки для работы с бэкендом на 10 языках: C, C++, Java, Go, Node.js, Python, Ruby, Objective-C, PHP, C#.

Принципы:

Первый принцип, сформулированный компанией, звучит буквально как «Сервисы, а не объекты, сообщения, а не ссылки» - что говорит об ориентированности на написание микросервисов. Второй — широта применимости и простота — означает поддержку всех популярные платформы для разработки и тот факт, что фреймворк должен нормально работать на маломощных устройствах. Многоуровневость — подразумевает то, что наличествуют уровни, в достаточной степени независимые и, как следствие, изменения в формате данных не должны ломать вызовы в приложении. Что касается гибкости протокола — речь идет о концептуальной возможности замены формата сообщений (JSON, XML, Thrift, FlatBuffers) и подключаемом механизме сжатия сообщений.

1. Архитектура:

#### Protocol Buffers:

Protobuf - это формат сериализации, по умолчанию используемый для передачи данных между клиентом и сервером. Потребляет меньше ресурсов, так как использует строгую типизацию полей и бинарный формат для передачи структурированных данных. Время выполнения процесса сериализации/десериализации значительно меньше, как и размер сообщений, в отличии от JSON/XML. Для написания protobuf файлов используется язык описания интерфейсов (IDL). Так, чтобы описать структуру данных сообщения, нужно добавить message, имя структуры, внутри нее — тип, название и номер поля. Номера полей очень важны для обратной совместимости, поэтому не стоит менять их последовательность при добавлении или удалении полей. Старые номера можно резервировать.

#### Типы RPC:

Унарный (Unary RPC). Синхронный запрос клиента, который блокируется, пока не будет получен ответ от сервера.

Серверный стрим (Server streaming RPC), при подключении клиента сервер открывает стрим и начинает отправлять сообщения.

Клиентский стрим (Client streaming RPC). То же самое, что и серверный, только клиент начинает стримить сообщения на сервер.

Двунаправленный стрим (Bidirectional streaming). Клиент инициализирует соединение, создаются два стрима. Сервер может отправить изначальные данные при подключении или отвечать на каждый запрос клиента по типу “пинг-понга”.

1. Перехватчики (Interceptors):

Выступают в роли ППО внутри ППО, служат для перехвата вызовов и встраивания необходимых условий, что подойдет для решения задач по сбору метрик, логированию запросов, аутентификаций, etc.

#### Балансировка нагрузки (Load Balancing):

Выполняется на стороне клиента. Клиент использует простой “round- robin” алгоритм для передачи запросов по списку полученному от LB сервера.

2. Преимущества:

* 1. HTTP/2 в качестве транспорта: сжатие данных, возможность инициации событий с сервера, контроля трафика, повторного использования одного cокета для нескольких параллельных запросов.
  2. Статические пути — никаких структур типа «сервис/коллекция/ресурс/запрос? параметр=значение», только «сервис», а содержимое можно описывать в терминах нужной модели и её событий.
  3. Никакого привязывания методов к HTTP-методам, никакого привязывания возвращаемых значений к HTTP-статусам.
  4. SSL/TLS, OAuth 2.0, возможность аутентификации через сервисы Google или же встраивание своей (например, двухфакторной)
  5. Поддержка gRPC в публичных API от Google, что уже работает для некоторых сервисов.

2. Apache Thrift  — язык описания интерфейсов, который используется для определения и создания служб на различных языках программирования, являющийся фреймворком к удалённому вызову процедур. Используется компанией Facebook в качестве масштабируемого кросс-языкового сервиса по разработке, имеет реализации на языках C#, C++, Cappuccino, Cocoa, Delphi, Erlang, Go, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, Smalltalk и JavaScript. С апреля 2007 года разрабатывается компанией Apache Software Foundation как open-source проект.

Принципы:

В состав Thrift входит готовый программный конвейер для работы над клиентской и серверной сставляющими, состоящий из шести уровней. Верхний уровень – сгенерированный код описания Thrift. Из него службы генерируют код клиента и сервера, и созданная таким образом структура данных возвращается в виде сгенерированного кода. Уровни протокола и транспортировки являются частью runtime-библиотеки. Имеется возможность выбора служб и изменения протоколов сериализации и транспортировки без перекомпиляции кода. Кроме клиентской части в состав Thrift входит серверная инфраструктура для объединения протоколов сериализации и транспортировки в blocking, non-blocking и multi-threaded серверах. Что же касается уровня ввода/вывода данных, то его основа по-разному реализована для различных языков программирования.

1. Архитектура:

Поддерживаемые протоколы:

* + 1. TBinaryProtocol — Простой, неоптимизированный для экономии пространства, двоичный формат данных
    2. TCompactProtocol — Более эффективный и компактный (как правило) формат бинарного представления данных
    3. TDebugProtocol — Формат текста, понятный человеку и предназначенный для облегчения процесса отладки сервиса
    4. TDenseProtocol — предназначен для извлечения мета-информации из переданных данных, аналогичен TCompactProtocol
    5. TJSONProtocol — в качестве кодировщика данных испольуется JSON
    6. TSimpleJSONProtocol — Предназначенный только для записи протокол, который также использует JSON, в связи с чем могущий подойти, например, для парсинга на скриптовых языках

Поддерживаемые транспортировщики:

* + 1. TFileTransport — Транспорт, использующий файлы для ввода/вывода.
    2. TFramedTransport — Транспорт для non-blocking серверов, отправляющий данные во фреймах, предваряя их информацией о длине передаваемых данных.
    3. TMemoryTransport — Транспорта. Использующий для ввода/вывод память.
    4. TSocket — Транспорт, использующий blocking socket ввода/вывода.
    5. TZlibTransport — Транспорт, выполняющий сжатие данных при помощи zlib и исполльзующийся в сочетании с другим

Поддерживаемые серверы:

* + - 1. TNonblockingServer — multi-threaded сервер, использующий non-blocking ввод/вывод иTFramedTransport.
      2. TSimpleServer — single-threaded, использующий std blocking ввод/вывод, находящий свое применение при тестировании
      3. TThreadPoolServer — multi-threaded сервер, использующий std blocking ввод/вывод.

1. Преимущества:
   1. Кросс-языковая сериализация с более низкими накладными расходами, в отличие от таких альтернатив как SOAP, за счет использования двоичного формата.
   2. Простая и чистая библиотека., не нуждающаяся во фреймворке для кода и не использующая XML-конфигурацию.
   3. Языковые привязки ощущаются естественными. Пример: в Java для ArrayList<String>. C++ используется std::vector<std::string>.
   4. Формат связи уровня приложений и формат связи уровня сериализации строго разделены — могут быть изменены независимо друг от друга.
   5. Встроенные типы сериализации включают в себя: двоичный, компактный двоичный и дружественный к HTTP.
   6. Мягкие версии протокола: не требуется такой централизованный и явный механизм как major-version/minor-version, слабосвязанные группы могут свободно переходить в RPC вызовы.
   7. Независим от архитектуры или от нестандартного ПО, нет несовместимых лицензий на ПО.
2. Относительно Protobuf:
   1. Документация хуже → хуже дела с доступными рабочими примерами, etc.
   2. API более запутанное.
   3. Большее количество языков, поддерживаемых “из коробки”.
   4. Больше поддерживаемых сложных структур (таких как Map, Set)
   5. Наличие механизма улавливания исключений
   6. Включает в себя RPC имплементацию, в то время как у Protobuf она есть только будучи в составе gRPC

3. Apache Avro – фреймворк для удаленного вызов процедур и сериализации данных, разработанный в проекте Apache Hadoop. Avro использует формат JSON для объявления структур данных и сериализует данные в компактном двоичном формате. В настоящее время есть поддержка таких языков, как Java, C, C ++, C #, Python и Ruby. Его основное применение - в Apache Hadoop, где он может предоставить как формат сериализации для постоянных данных, так и формат для обмена данными между узлами Hadoop и клиентскими программами для служб Hadoop.

Принипы:

Avro базируется на схемах: при считывании данных схема остается доступной, что позволяет записывать информацию без каких-либо накладных расходов, делая процесс сериализации быстрее, а продукт - компактнее. Это также облегчает использование динамических, скриптовых языков программирования, поскольку данные вместе со схемой полностью себя самоописывают, а генерация кода не требуется для чтения или записи файлов данных и использования или реализации протоколов RPC. Если программа, считывающая данные, ожидает другую схему данных, это не будет проблемой – так как и клиент, и сервер имеют полные схемы, описывающие ожидаемые при получении данные, либо же могут обмениваться в специальном режиме ими, и соответствия между одинаковыми именами полями, отсутствующими полями, дополнительными полями и т.п. легко описываются.

1. Архитектура:

1. Схема:

Данные Avro всегда сериализуются по соответствующей им схеме, а файлы, в которых хранятся данные Avro, всегда должны также включать схему для этих данных в одном файле. Системы удаленного вызова процедур (ARC) на основе Avro также должны гарантировать, что у удаленных получателей данных есть копия схемы, используемой для записи этих данных. Поскольку схема, используемая для записи данных, всегда доступна, когда данные читаются, то сами данные Avro не помечены информацией о типе. Для анализа данных требуется схема.

1. Сериализация:

Как правило, сериализация и десериализация выполняется с исполльзованием принципа обхода в глубину, слева направо, и сериализуя типовые примитывы по мере их появления. В Avro определено два алгоритмасериализации: двоичный и JSON. В большинстве случаев кодирование в бинарном виде предпочтительнее, так как позицинируется как более быстрое и дающее более компактные сообщения на выходе, но, например для отладки и веб-приложений кодировка JSON иногда может быть удобным решением.

1. Транспорт

1. Система которая поддерживает передачу сообщений-запросов, представляющих из себя “голые” последовательности байтов, и получение соответствующих ответных сообщений, что является, собственно говоря, основой принципа рассматриваемого класса.

 2. При отправке сервером ответа клиенту механизм выдачи такого ответа будет сильно зависеть от используемого в качестве сетевого протокола транспорта. Например, в HTTP этот механизм – неявный, так как HTTP напрямую поддерживает запросы и ответы. Но транспорт, который мультиплексирует многие потоки клиентов через один сокет, должен будет помечать сообщения с уникальными идентификаторами.

3. В тех случаях, когда HTTP используется как транспорт, каждый обмен сообщениями Avro является парой HTTP-запрос / ответ. Все сообщения протокола Avro должны иметь один URL-адрес на HTTP-сервере. Другие протоколы также могут использовать этот URL. Как регулярные сообщения, так и сообщения об ошибках Avro должны использовать коду ответа 200 (OK). Кодировка разбитой на куски информации может использоваться для запросов и ответов, но независимо от запроса и ответа Avro - это весь контент HTTP-запроса и ответа. Тип содержимого HTTP-запроса и ответы должны быть указаны как «avro / binary». Запросы должны выполняться с использованием метода POST.

4. Имеется поддержка фреймов (разбития на кадры) сообщений, что необходимо, когда входные данные больше допустимого размера буфера. Важно отметить, что, маленькие объекты при “кадрировании” затем добавляются в буферы, тогда как более крупные объекты записываются как собственные буферы. Когда же при считывании осуществляется попытка прочитать большой объект, среда, в которой это происходит, может передать весь буфер напрямую, без необходимости его копирования.

5. Отдельно стоит упомянуть имеющийся механизм “рукопожатий” (Handshake), смысл которого заключается в том, чтобы убедиться, что клиент и сервер имеют определение протокола друг друга, чтобы клиент мог правильно десериализовать ответы, а сервер, соответственно, мог правильно десериализовать запросы. Как клиенты, так и серверы должны хранить кэш недавно просмотренных протоколов, поэтому в большинстве случаев “рукопожатие” будет завершено без дополнительных обменов, и при дальнейшем обмене сообщениями не потребуется

2. Относительно gRPC(Protobuf) и Apache Thrift (а также – преимущества Avro):

В целом все аналогично, но вот важные отличия, заключающиеся в следующих аспектах:

1. Динамическая типизация: Avro не требует создания кода – данные всегда сопровождаются схемой, которая позволяет полностью обрабатывать эти данные без генерации кода, статических типов данных и т.д. Это облегчает создание общих систем обработки данных и языков.
2. “Untagged” данные: поскольку при считывании данных схема никуда не исчезает и находится в зоне досягаемости, значительно меньше информации о типе данных необходимо кодировать, что, соответственно, приводит к меньшему размеру продукта сериализации.
3. Отсутствие вручную назначаемых id полей: при изменении схемы при обработке данных всегда присутствуют как старые, так и новые схемы, поэтому различия могут быть отображены символически –при помощи имен полей.
4. Реализация RPC отсутствует в некоторых языках, где присутствует реализация протокола сериализации, что может учинять трудности при разработке приложений на конкретной платформе.

**1.3. Описание существующих методик сравнения реализаций класса технологий**

Сравнение реализаций класса технологии базируется на присовенных каждому продукту оценках, выносимых на основе критериев применимости, по принципу вынесения которых и можно дифференцировать методики сравнения.

Так, для обработки полученных данных о программных продуктах используется подход экспертного оценивания — процедура получения оценки проблемы на основе мнения специалиста/ов с целью последующего принятия решения.

Существует две группы экспертных оценок:

1. Индивидуальные оценки основаны на использовании мнения отдельных экспертов, независимых друг от друга.
2. Коллективные оценки основаны на использовании коллективного мнения экспертов.

Совместное мнение обладает большей точностью, чем индивидуальное мнение каждого из специалистов. Данный метод применяют для получения количественных оценок качественных характеристик и свойств. Например, оценка нескольких технических проектов по их степени соответствия заданному критерию, во время соревнования оценка судьями выступления фигуриста.

Известны следующие методы экспертных оценок:

1. Метод ассоциаций. Основан на изучении схожего по свойствам объекта с другим объектом.
2. Метод парных (бинарных) сравнений. Основан на сопоставлении экспертом альтернативных вариантов, из которых надо выбрать наиболее предпочтительные.
3. Метод векторов предпочтений. Эксперт анализирует весь набор альтернативных вариантов и выбирает наиболее предпочтительные.
4. Метод фокальных объектов. Основан на перенесении признаков случайно отобранных аналогов на исследуемый объект.
5. Индивидуальный экспертный опрос. Опрос в форме интервью или в виде анализа экспертных оценок. Означает беседу заказчика с экспертом, в ходе которой заказчик ставит перед экспертом вопросы, ответы на которые значимы для достижения программных целей. Анализ экспертных оценок предполагает индивидуальное заполнение экспертом разработанного заказчиком формуляра, по результатам которого производится всесторонний анализ проблемной ситуации и выявляются возможные пути её решения. Свои соображения эксперт выносит в виде отдельного документа.
6. Метод средней точки. Формулируются два альтернативных варианта решения, один из которых менее предпочтителен. После этого эксперту необходимо подобрать третий альтернативный вариант, оценка которого расположена между значений первой и второй альтернативы.
7. Метод мозгового штурма. Оперативный метод решения проблемы на основе стимулирования творческой активности, при котором участникам обсуждения предлагают высказывать как можно большее количество вариантов решения, в том числе самых фантастичных. Затем из общего числа высказанных идей отбирают наиболее удачные, которые могут быть использованы на практике.
8. Иерархический аналитический процесс – мощный метод сопоставительного анализа и ранжирования объектов, характеризующихся наборами критериев и показателей, количественных и качественных. В литературе этот метод называют также методом анализа иерархий. Метод применяется для многих задач, в том числе и для сравнительного анализа обьектов (многокритериального ранжирования).

**1.4. Вывод**

По результатам аналитической части учебно-исследовательской работы:

1. Были изучены и описаны вышеупомянутые релизации RPC. Таким образом, был проведен их сравнительный анализ.
2. Были сформулированны и обоснованны критерии применимости реализаци RPC. Это позволит сделать выводы об эффективности класса при применении одной из существующих или разработанной самостоятельно методики оценки реализаций класса технологий для их сравнения между собой.
3. Были описаны эти самые методики сравнения (оценки).

**1.5. Цели и задачи УИР**

Целью учебно-исследовательской работы является оценка эффективности применения RPC как класса технологий для организации сетевого взаимодействия компонентов распределенной системы.

Для этого необходимо разработать методики сравнения реализаций RPC по критериям применимости, методики проведения испытаний и методики оценки результатов проведения этих испытаний, реализовать разработанные методики и получить результаты сравнения реализаций RPC на основе критериев применимости, после чего сделать выводы на основе результатов сравнительных испытаний.

**Глава 2. Разработка методики сравнения реализаций PRC по критериям применимости**

**2.1. Методика сравнения по критериям применимости**

Суть метода - выполнение аддитивной сверки продуктов по заранее согласованным критериям оценки с коэффициентами значимости каждого критерия. Оценки будут опираться на критерии применимости и, таким образом, сравнение реализаций RPC будет происходить на основе этих критериев.

Шаг 1. Потребуется провести необходимые испытания реализаций класса RPC, чтобы получить данные по замерам параметров, имеющих непосредственно отношение к критерям применимости.

Шаг 2. После завершения записи всех необходимых наблюдений необходимо разработать методику сравнения проведения испытаний.

Шаг 3. Аккумулированные вследствие проведения испытаний данные замеров следует обработать согласно разработанной методике оценки результатов испытаний.

Шаг 4. На основе полученных оценок и графических представлений данных при помощи метода попарных сравнений обозначить для каждой из реализаций класса RPC ту распределеную систему, характеризуемую соответстующими критериям применимости параметрами, для налаживания и работы которой она будет наиболее эффективна, и вынести вердикт об эффективности применения данного класса технологий для решения поставленной задачи в целом.

**2.2. Методика проведения испытаний**

Шаг 1. Формирование единого пакета (написание скриптов для генерации) разнородных данных, которые в дальнейшем будут использоваться для проведения каждого из испытаний.

Выборка должна быть дифференцирема по обьему и типу данных, файлы сгенерированны случайным образом и сохраняются в неизменном виде на протяжении всех испытательных мероприятий.

Шаг 2. Разворачивание фреймворков на машине, на которой будут проводиться испытания.

Для того, чтобы перейти к следующим этапам испытаний, необходимо при помощи тестов, предоставленных разработчиками, удостовериться в корректности установки и работы рассматриваемых реализаций.

Шаг 3(опционально). Создание тестовых клиент — серверных сетей, для того чтобы удостовериться в корректности работы сервисов, необходимых для тестирования самолично, тем самым заодно утвердив собственное понимание принципов работы с API реализаций класса технологий.

Шаг 4. Написание программ, реализующих процессы сериализации/десериализации данных при помощи инструментов, предоставляемых фреймворками и обмена сообщениями с использованием как протоколов, предоставляемых фреймворками, так и с использованием стандартного API для создания ориентиров.

Шаг 5. Написание скриптов, которые согласно намеченному плану позволят запустить реализованные сервисы и собрать необходимые для дальнейшей оценки и сравнительного анализа данные о времени работы тех или иных процессов и затратам памяти.

**2.3.** **Методика оценки результатов проведения испытаний**

Шаг 1. Формирование таблицу критериев оценки.

Таблица критериев оценки должна быть достаточно полной, чтобы отразить наиболее существенные для поставленной задачи характеристики реализации. В то же время таблица не должна быть излишне перегруженной, чтобы значимые критерии не были маскированы несущественным "шумом".

Шаг 2. Определятся методика оценки выполнения критерия таким образом, чтобы "обезразмерить" исходные показатели. Данная процедура обычно называется шкалированием. Для этого результат оценки переводится в один из двух вариантов: равен 0 в случае невыполнения критерия и равен 1 в случае, если критерий оценки выполнен.

Шаг 3. Для каждого критерия выставляются коэффициенты значимости критерия для оценки реализации, в зависимости от исходных параметров распределенной системы, которую необходимо реализовать с помощью данного программного продукта. Коэффициенты распределяются на отрезке от 0 до 1.

Шаг 4. Производится расчет аддитивной суммы интегральной оценки для каждого сравниваемого продукта по следующей формуле:

О = ∑ni=1(Zi\*Ki), где

O - интегральная оценка реализации;

п - количество критериев сравнения заявленных продуктов согласно таблице критериев;

Zi - значение выполнения критерия;

Ki - коэффициент значимости критерия сравнения.

Шаг 5. Значения интегральных оценок для каждого сравниваемого продукта ранжируются по убыванию.

Таким образом, получается таблица оценки реализации, которая используется для принятия решения в результате сравнения реализаций RPC между собой по отнесению каждой реализации к группе инструментов, пригодной для эффективной организации сетевого взаимодействия компонентов распределенной системы с теми или иными требованиями к ресурсоемкости и скорости (исходя из критериев применимости) и оценки эффективности класса технологии для решения данной задачи в целом.

Шаг 6. Наглядное представление оценок

Для наглядности вынесенных вердиктов и осуществления возможности их обсуждения и дополнительного, возможно, более подробного или глубоко анализа, возможности сконцентрироваться на более мелких аспектов, полученные в результате замеров различные совокупности данных следует представить в виде удобных для анализа диаграмм.

**2.4. Вывод**

В результате выполнения теоретической части учебно исследовательской были разработаны и представлены(описаны):

1. Методика сравнения по критериям применимости.

2. Методика проведения испытаний.

3. Методика оценки результатов проведения испытаний.

**Глава 3. Результаты сравнения реализаций RPC на основе критериев применимости.**

*\*в данной главе приводится реализация разработанной методики сравнения реализаций RPC по критериям применимости, а также результаты сравнительных испытаний\**

**3.1. Реализация разработанной методики сравнения реализаций RPC по критериям применимости**

(Было написано то и то, между делом — приложения кода с комментариями)

**3.2. Результаты сравнительных испытаний**

(таблицы и диаграммы с результатами тестов и оценками с пояснениями — пока приложу отдельным файлом рядом с пз, потом встрою)

**3.3. Выводы**

(ссылаясь на табличные данные, скажу, в каком случае какой комбинации протоколов отдается предпочтение, после чего в целом оцениваем эффективность класса RPC)

# Заключение

В ходе данной работы были выполнены следующие задачи:

* 1. Выявлены и обоснованы критерии применимости реализаций RPC;
  2. Разработаны методики сравнения реализаций PRC по критериям применимости:

2.1 Методика сравнения по критериям применимости;

2.2 Методика проведения испытаний;

2.3 Методика оценки результатов проведения испытаний;

* 1. Выполнена реализация разработанной методики сравнения реализаций RPC на основе критериев применимости.

Выполненные задачи позволили достичь поставленной в работе цели, а именно оценить эффективность применения класса RPC для организации сетевого взаимодействия компонентов распределенной системы.

# Список литературы

1. Nicolai M. Josuttis "The C++ Standard Library: A Tutorial and Reference (2nd Edition)".-US, Addison\_Wesley, 2012

2. Matthias Kalle Dalheimer "Programming with Qt, 2nd Edition. Writing Portable GUI applications on Unix and Win32". -US, O'Reilly Media, 2002

3. GRPC documentation [Электронный ресурс] // grpc-community webpage: сайт. – URL:http://www.grpc.io/docs/

4. The ZeroMQ Reference Manual etc. [Электронный ресурс] //zeroMQ manuals webpage: сайт. - URL:http://zeromq.org/intro:read-the-manual

5. Thrift Tools [Электронный ресурс] // apache-community webpage: сайт. – URL:https://thrift.apache.org/

# 6. Apache Avro™ 1.8.1 Documentation [Электронный ресурс] // apache-community webpage: сайт. – URL:http://avro.apache.org/docs/1.8.1/