УДК 004.451.2

СРАВНЕНИЕ КЛИЕНТ-СЕРВЕРНЫХ РЕШЕНИЙ ПРИ РАЗРАБОТКЕ МНОГОПОЛЬЗОВАТЕЛЬСКИХ ОНЛАЙН-ИГР НА UNITY

И. Р. Мухаметханов[1] [0000-0002-0562-1434], М. Р. Хафизов[2] [0000-0001-7275-9102],

А. В. Шубин[3] [0000-0002-6203-3268]

[1,2,3]*Институт информационных технологий и интеллектуальных систем Казанского (Приволжского) федерального университета;*

[1]ilnurhamay@mail.ru, [2]mrimsh@mrimsh.ru, [3]shubin.aleksey.kpfu@gmail.com

Аннотация

В работе представлена критика традиционного подхода, используемого для создания многопользовательской игры в системе разработки интерактивных приложений в реальном времени Unity, особенно в случае большого числа одновременных пользователей. В качестве гипотезы предложен альтернативный вариант, не являющийся распространённым, но решающий многие проблемы предыдущего подхода. Проведено сравнение двух клиент-серверных решений при разработке в Unity многопользовательских онлайн-игр, также описаны преимущества обоих подходов для разных случаев. Предложена архитектура разработки игры при помощи более актуального метода: вместо библиотеки Mirror – стандартного инструментария для Unity-разработки – использованы микросервисы, написанные на языке Golang. Приведены весомые доказательства предпочтительности альтернативного подхода, главное преимущество которого – поддержка современной архитектуры, обеспечивающей высокоскоростную связь между микросервисами, что подкреплено тестами при передаче сообщений на разных платформах.

Полученные результаты тестирования подтверждают выдвинутую гипотезу, и можно сделать вывод, что для многопользовательских видеоигр связка Unity вместе с Golang является более эффективной.

Описаны также основные методы отладки многопоточного приложения на Golang в связке с системой разработки игровых приложений Unity и предложен

технологический прием, позволяющий получить быстрый способ передачи данных между клиентом и сервером.

***Ключевые слова**: игровой движок, микросервисы, Unity, Golang, Mirror, видеоигра, мультиплеер.*

ВВЕДЕНИЕ

Играя с другими людьми, игрок получает новые опыт и эмоции. Поэтому для реализации видеоигр очень важно использовать подходящие клиент-серверные решения, тем самым обеспечив игроку комфорт в плане корректной и бесперебойной работы для заданного количества пользователей.

Отсутствие ошибок и корректная работа – одни из важнейших аспектов игры, позволяющие удержать игрока на более длительный промежуток времени, а обеспечивает данные аспекты именно грамотный выбор используемого решения. Обсудим, какие инструменты для этого используют при разработке и в чем их принципиальные различия.

Unity – система разработки интерактивных приложений, называемая среди разработчиков игровым движком[1], является бесплатным, одним из наиболее популярных вариантов для разработки видеоигр, имеющих низкий порог входа, что позволяет начать разработку в кратчайшие сроки.

При создании мультиплеерной[2] видеоигры разработчики имеют довольно широкий выбор сервисов для реализации. Довольно популярным среди других разработчиков [1–3] является использование Photon Unity Networking (PUN)[3]. Данный инструмент при всём его удобстве и популярности имеет большой недостаток – поддержка до 20 пользователей, за большее число пользователей разработчик обязан заплатить за аренду мест, так как все приложения, применяющие технологию PUN, запускаются на серверах Photon Cloud.

[1] Игровой движок (от англ. game engine) – комплексное ПО для разработки и поддержки игровых приложений.

[2] Мультиплеер (от англ. multiplayer) – режим компьютерной игры, когда совместно играет больше одного человека.

[3] Photon Unity Networking (PUN) или Photon Engine – популярный фреймворк для разработки многопользовательских игр. URL: https://www.photonengine.com/en-US/Photon.

Просмотрев другие варианты реализации мультиплеера, мы решили сравнить пару вариантов решений: (1) традиционное решение от Unity в виде библиотеки Mirror[4], использующей архитектуру REST[5], и (2) приложение на языке Golang[6] (или Go), реализуемое через микросервисы[7].

Mirror – это высокоуровневая сетевая библиотека для Unity, совместимая с различными транспортами низкого уровня. Она используется для создания как клиентского, так и серверного приложений и поддерживает архитектуру REST для оптимизации приложений и транспортировки данных. REST – это набор правил для разработки веб-приложений. Приложение является RESTful, если удовлетворяет набору условий.

Golang – компилируемый язык программирования с открытым исходным кодом, ориентированный на избавление от избыточных данных и скорость работы кода, в том числе при помощи многопоточных вычислений. С момента своего первого выпуска в 2009 году язык программирования Golang [4] был хорошо принят сообществами разработчиков программного обеспечения. Основной причиной его успеха стала мощная поддержка разработки на основе библиотек, когда проект, реализуемый на Golang, может быть удобно построен поверх других проектов путем ссылки на них как на библиотеки, что позволяет проекту импортировать и повторно использовать функциональные возможности из другого проекта Golang, просто указав путь импорта.

ДОСТОИНСТВА И НЕДОСТАТКИ ВЫБРАННЫХ ВАРИАНТОВ РЕАЛИЗАЦИИ

По нашему мнению, одна из важнейших проблем Unity и решений, реализованных при помощи этого игрового движка, это отсутствие безопасности отдельных потоков, заключающееся в невозможности проводить сверхсложные вычисле-

[4] Mirror, https://mirror-networking.gitbook.io/docs/general/ccu.

[5] REST (от англ. Representational State Transfer – «передача репрезентативного состояния» или «передача „самоописываемого“ состояния») — архитектурный стиль взаимодействия компонентов распределённого приложения в сети.

[6] Golang – The Go Programming Language, https://go.dev

[7] Микросервис (от англ. microservices) – вариант архитектуры программного обеспечения, состоящей из взаимозаменяемых и независимых модулей.

ния в другом потоке – ведь, чтобы применять результаты таких вычислений на игровой сцене[8], необходимо вернуться в основной поток. При этом Golang имеет большой потенциал для разработки в многопоточном режиме и, соответственно, может задействовать больше мощностей сервера.

Вторая проблема состоит в том, что в Unity присутствует ограничение на количество игроков на одном сервере в 1500 пользователей.

Наконец, ещё одна проблема состоит в том, что библиотека Mirror, традиционно используемая для Unity, имеет недостаток в виде устаревшей системы удалённого вызова процедур (RPC[9]), разработка и практическая имплементация которой были начаты ещё в 1970-х годах.

В свою очередь Golang славится поддержкой gRPC – это архитектура RPC с открытым исходным кодом, разработанная Google для обеспечения *высокоскоростной* связи между микросервисами. Она позволяет интегрировать сервисы, написанные на разных языках, используя высокоэффективный формат обмена сообщениями через буферы протокола.

Существует много популярной литературы и научных исследований, посвященных сравнению производительности веб-серверов, но редко обсуждается их сочетание, хотя Unity и Golang – популярные платформы, которые широко используются в качестве бэкендов веб- и мобильных приложений.

Итак, оценим реализацию мультиплеера на Unity при помощи как традиционного решения в виде библиотеки Mirror, так и через реализацию микросервисов на языке Golang. Выделим конкретные достоинства и недостатки каждого метода – для определения более подходящего в случае разработки многопользовательской видеоигры.

[8] Игровая сцена – общеупотребительный термин в разработке игр и приложений компьютерной графики – внутриигровое пространство с 3D и 2D объектами и другими данными, структурированными в определенной иерархии, которые необходимы для организации взаимодействий игрового процесса.

[9] RPC (от англ. remote procedure call) – Удалённый вызов процедур.

Характеристики gRPC

Достоинства gRPC:

1. Использование http/2 в качестве транспортного протокола:
   - асинхронная обработка больших наборов данных;
   - поддержка отправки нескольких запросов по одному соединению;
   - двунаправленное взаимодействие для одновременной отправки запросов от клиента и получения ответа от сервера;
   - сжатие заголовков (уменьшение загруженности сети).
2. Применение Protobuf[10] (вместо стандартных JSON или XML) для обработки данных:
   - сериализация сообщений в строго типизированном двоичном формате, что может легко восприниматься любым языком программирования;
   - более сжатый формат сообщений.
3. Встроенная генерация кода для создания запросов, что позволяет отказаться от стороннего инструментария.
4. Встроенный компилятор протоколов.
5. Поддержка нескольких языков программирования и отсутствие зависимости от платформы.

Недостатки gRPC:

1. Отсутствие обратной совместимости с протоколом http 1.1, что ограничивает применение метода.
2. Использование Protobuf не обеспечивает удобочитаемости данных для программиста.
3. На данный момент, является менее популярным методом, что влечёт за собой отсутствие прямой совместимости с некоторыми инструментами.

Платформа gRPC была создана для ускорения передачи данных между микросервисами и другими системами, которым необходимо взаимодействовать друг с другом. Несмотря на то, что gRPC основывается на RPC, он имеет ряд существенных отличий от своего предшественника.

[10] Protobuf – технология упаковки данных от Google. Буферы протокола – это легкий и эффективный формат хранения структурированных данных, который можно использовать для сериализации или сериализации структурированных данных.

Характеристики RPC

Основными положительными чертами RPC можно назвать отсутствие недостатков, названных для gRPC, а также:

1. Использование универсальных подходов и методов.
2. Высокая скорость имплементации решения.

Также данная технология не обходится и без недостатков. Недостатки RPC:

1. Необходимость в использовании стороннего инструментария для генерации кода, как, например, Swagger[11] или Postman[12].
2. Отсутствие стандартной методологии – имеются лишь требования к архитектуре, при этом реализация может отличаться.
3. Заметно низкая скорость передачи данных, связанная с использованием несжатых форматов файлов и устаревших протоколов передачи данных.

Учитывая все достоинства и недостатки, можно сказать, что использование платформы gRPC в связке с Unity имеет больше плюсов, чем с её аналогом REST. Наиболее весомым отличием этих двух связок является ускоренная передача сообщений – по тестам Рувана Фернандо [5], при передаче сообщений на разных платформах соединения gRPC API значительно быстрее, чем соединения REST API, фактически в 7–10 раз (см. рис. 1). Этот факт, вместе с поддерживаемым многопоточным сообщением с несколькими клиентами является критически важным для массовой многопользовательской видеоигры, где количество игроков может достигать нескольких тысяч одновременно. Кроме этого, от скорости и точности обмена данными будет влиять, насколько интересным будет опыт игры.

REST API идеально подходит, когда системе требуются высокоскоростная итерация и стандартизация протокола HTTP. Благодаря универсальной поддержке сторонних инструментов, REST API должен быть первым аргументом в пользу интеграции приложений, интеграции микросервисов и разработки веб-сервисов – но не видеоигры.

Что касается gRPC, то большинство инструментов сторонних производителей до сих пор не имеют встроенных функций для совместимости с ним.

[11] Swagger, https://swagger.io
[12] Postman, https://www.postman.com

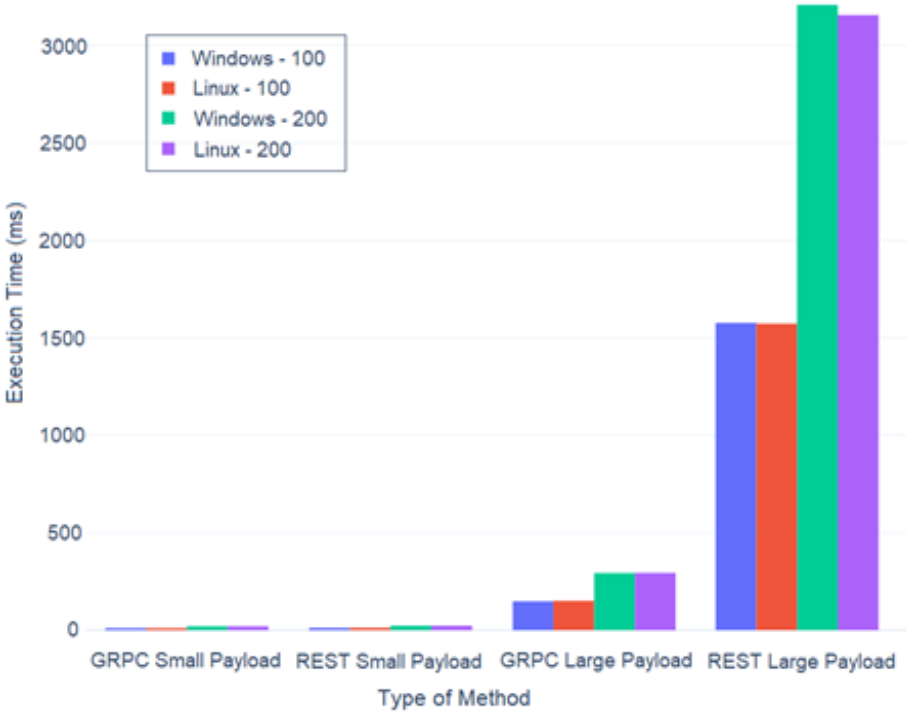

Рис. 1. Сравнение производительности REST и gRPC

Значит, gRPC используется в основном для построения внутренних систем, то есть инфраструктур, закрытых для внешних пользователей. Учитывая эту оговорку, API gRPC может быть полезен в следующих случаях:

- *Соединения легковесных микросервисов* – особенно там, где эффективность передачи сообщений имеет первостепенное значение.
- *Системы, где используется несколько языков программирования,* – благодаря поддержке генерации собственного кода для широкого спектра языков разработки.
- *Потоковая передача в реальном времени* – порождается способностью gRPC управлять двунаправленным потоком, не дожидаясь ответа от клиента.
- *Сети с низким энергопотреблением и низкой пропускной способностью* – использование сериализованных сообщений Protobuf обеспечивает простоту обмена сообщениями, большую эффективность и скорость для сетей с ограниченной пропускной способностью и низким энергопотреблением [6].

Контейнерная платформа Docker

Также микросервисы, написанные на Golang, очень удобно развёртывать за счёт Docker – платформы для разработки, доставки и запуска контейнерных приложений и управления их жизненным циклом. Docker позволяет запускать множество контейнеров на одной хост-машине[13]. Контейнеры в целом упрощают работу как программистам, так и администраторам, которые развертывают эти приложения, и позволяют упаковать в единый образ приложение и все его зависимости: библиотеки, системные утилиты и файлы настройки. Это упрощает перенос приложения на другую инфраструктуру. Контейнер – это набор процессов, изолированных от основной операционной системы. Приложения работают только внутри контейнеров и не имеют доступа к основной операционной системе (ОС). Это повышает безопасность приложений, потому что они не смогут случайно или умышленно навредить основной системе. Если приложение в контейнере завершится с ошибкой или зависнет, это никак не затронет основную ОС. Контейнеры хорошо вписываются в микросервисную архитектуру. Это подход к разработке, при котором приложение разбивается на небольшие компоненты, по возможности независимые. Обычно такой подход противопоставляется монолитной архитектуре, где все части системы сильно связаны друг с другом. Это позволяет разрабатывать новую функциональность быстрее, ведь в случае с монолитной архитектурой изменение какой-то части может затронуть всю остальную систему.

ОБЗОР МЕТОДОВ РАЗРАБОТКИ ВИДЕОИГР

Итак, выше были рассмотрены как достоинства, так и недостатки разных подходов к обеспечению многопоточности в многопользовательских видеоиграх. Связка Unity и Golang представляется удачной для видеоигр, в которых происходит передача данных в реальном времени. Это видеоигры в жанрах ММО[14],

[13] Хост-машина (от англ. host – «владелец, принимающий гостей») – любое устройство, предоставляющее сервисы формата «клиент-сервер» в режиме сервера по каким-либо интерфейсам и уникально определённое на этих интерфейсах. В более широком смысле под хостом могут понимать любой компьютер, подключённый к локальной или глобальной сети.

[14] ММО (сокр. от англ. Massively Multiplayer Online Game) – массовая многопользовательская онлайн игра.

MMORPG[15], Shooter[16] – игры, в которых происходит очень много обмена данными, где даже минимальная задержка может помешать наслаждаться геймплеем[17]. А также «большие» игры, в которые одновременно играет много игроков, – ведь, как и в случае с очень активными играми, будет очень большая нагрузка на сервер.

Приведем пример архитектуры такой видеоигры, основанной на микросервисном подходе. Микросервисы могут хранить какие-то данные о пользователе. Если эти данные изменяются, микросервис подключается к базе данных, в которой будут храниться эти данные. Также среди микросервисов присутствует файл *go.mod*, где хранятся все файлы репозитория, из которых будет использован программный код. Но самое интересное в использовании микросервисного подхода – это proto-файлы, где хранятся rpc-методы и структуры данных, которые они получают на вход или выдают на выходе. Из этих proto-файлов компилируется код *pb.go*, который содержит весь код Protobuf для заполнения, сериализации и извлечения типов сообщений запроса и ответа. Микросервис *grpc.pb.go* содержит

- тип интерфейса (или заглушка) для вызовов клиентов с помощью определенных в нём методов,
- тип интерфейса для реализации серверами также с определенными в нём методами.

Само Unity-приложение с контентом видеоигры содержит пользовательский интерфейс, через который и отправляются запросы на сервер. Сервер их обрабатывает и отправляет ответ. Пример архитектуры Unity-игры, использующей gRPC, приведён на рис. 2.

[15] MMORPG (сокр. от англ. Massively Multiplayer Online Role-Playing Game) – массовая многопользовательская ролевая игра.

[16] Shooter (англ. стрелок) – игровой жанр, достижение цели которого преимущественно заключается в ликвидации противника при помощи огнестрельного оружия.

[17] Геймплей (от англ. gameplay) – игровой процесс, который образуется благодаря действиям игрока.

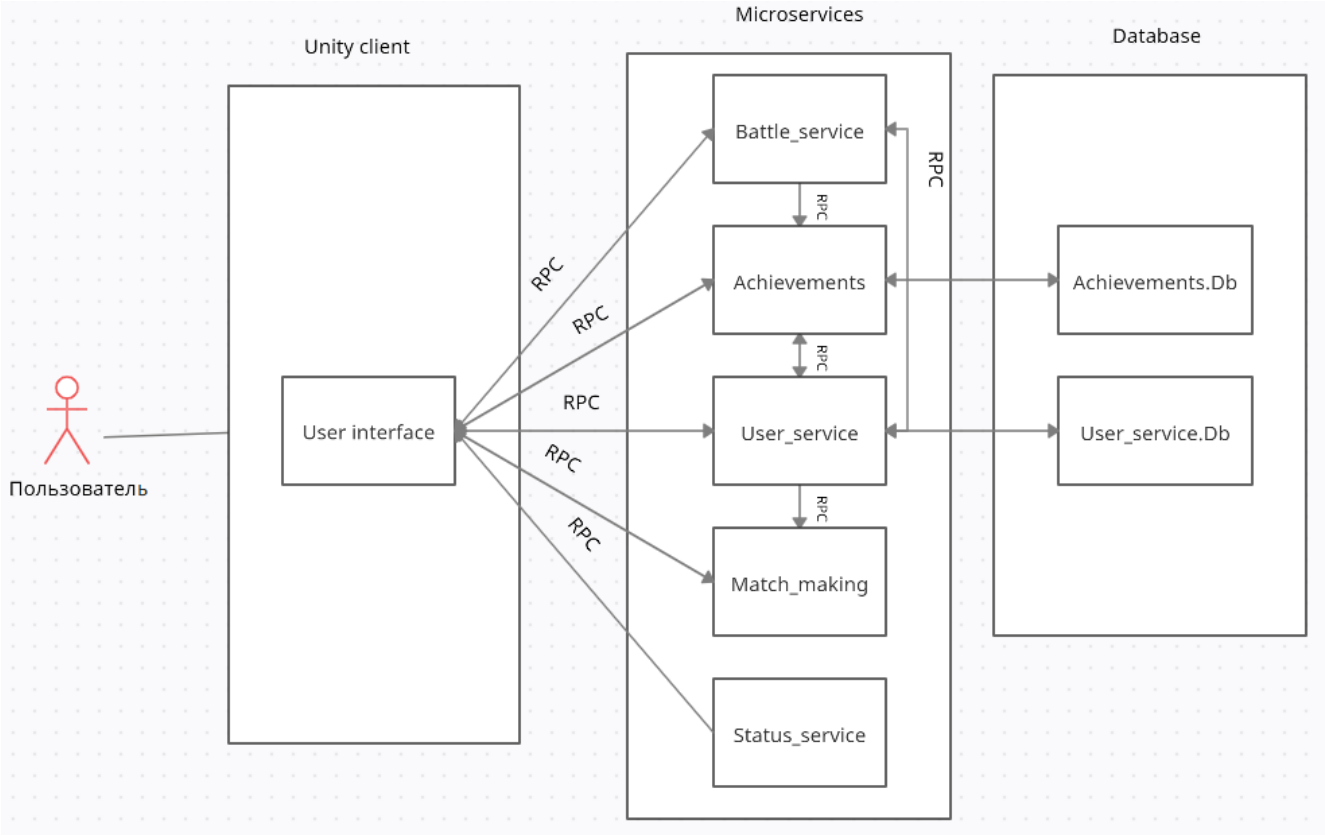

Рис. 2. Архитектура игры

Опишем, как происходит процесс создания микросервисов. Сначала создаётся proto-файл, в котором определены методы и сообщения, в частности, как на примере ниже (см. рис. 3), создаётся rpc-метод *GetUserProfile*, который принимает запрос *QueryUserProfile* и возвращает ответ *UserProfile*.

Далее на сервере создаётся метод *GetUserProfile*, который принимает в себя *QueryUserProfile* и возвращает *GetUserProfile* на основе proto. Unity же вызывает этот метод через код-прослойку на клиенте.

Использование докера позволяет легко собрать микросервисы и, в случае необходимости их изменения, легко сделать это, не задевая работу остальных компонентов системы. Для каждого микросервиса нужны свои proto-файлы, а для Unity можно создать отдельный proto-файл для удобства. Для того чтобы это всё работало, необходимо для каждого метода, в котором есть общение клиента с сервером, создать свой rpc-метод.

Докер – это тот технологический приём, который позволяет реализовать не только эффективный и быстрый, но и безопасный способ передачи данных между клиентом и сервером.

```
service UserService {
    rpc Auth(AuthRequest) returns (Session);
    rpc Registrate(RegistrateRequest) returns (Session);
    rpc ChangeUsername(CommandChangeUsername) returns (common.Empty);
    rpc ChangeAvatar(CommandChangeAvatar) returns (common.Empty);
    rpc ChangeTable(CommandChangeTable) returns (common.Empty);
    rpc ChangeHandColor(CommandChangeColor) returns (common.Empty);
    rpc ChangeHandJewelry(CommandChangeJewelries) returns (common.Empty);
    rpc GetUserEmojies(QueryUserEmojies) returns (ProfileItemResult);
    rpc GetAvatars(QueryAvatars) returns (ProfileItemResult);
    rpc GetUserInfo(QueryUserInfo) returns (UserInfo);
    rpc GetUserProfile(QueryUserProfile) returns (UserProfile);
    rpc GetUserUpdates(QueryUserUpdates) returns (UserUpdatesResult);
    rpc GetLeaderboard(QueryLeaderboard) returns (LeaderboardResult);
    rpc GetUserLegendaryTrackReward(CommandGetUserLegendaryTrackReward) returns (RewardResult);
    rpc GetLegendaryTrack (QueryLegendaryTrack) returns (LegendaryTrackResult);
    rpc GetOpenUserAchievements (QueryUserAchievements) returns (AchievementsResult);
        rpc GetAllUserAchievements (QueryUserAchievements) returns (AchievementsResult);
        rpc GetUserAchievementReward (CommandGetUserAchievementReward) returns (UserAchievementReward);
    rpc GetUserDailyTasks (QueryUserDailyTasks) returns (DailyTasksResult);
        rpc GetUserDailyReward (CommandGetUserDailyReward) returns (DailyReward);
    rpc MarkEmojiAsActive (MarkEmojiAsActiveRequest) returns (common.Empty);
    rpc UnmarkEmojiAsActive (UnmarkEmojiAsActiveRequest) returns (common.Empty);

    rpc CleanCardsAndTutorial (common.Empty) returns (common.Empty);
    rpc GetTutorialState(QueryUserStatus) returns (StatusUserResult);
    rpc SetTutorialState(CommandSetStatus) returns (shop.PurchaseProductResult);
}
```

Рис. 3. Пример proto-файла

Нюансы отладки многопоточных программ

Стоит отметить, что Go, являющийся многофункциональным языком программирования, требует эффективных методов отладки параллелизма. Но использование комбинированного инструмента статического и динамического тестирования и анализа параллелизма значительно облегчает процесс отладки реальных программ, предлагая автоматизированную динамическую трассировку для фикса-

ции поведения примитивов параллелизма, систематическое исследование пространства расписаний для ускорения поиска ошибок и обнаружения тупиков с дополнительными визуализациями и отчетами, набор требований к покрытию, характеризующий динамическое поведение примитивов параллелизма и метрики для измерения качества тестов. Эти подходы, безусловно, эффективны в обнаружении редких ошибок, а метод возмущения расписания позволяет быстро их обнаружить, делая «полевую» отладку программ на Go значительно проще.

Мы не рассматривали работу анализируемых продуктов с СУБД[18], так как скорость работы с ней зависит непосредственно от самой СУБД. Однако нужно отметить, что в [7] проведено сравнение производительности Golang и node.js в качестве бэкенда веб-приложений в сцепке с MySQL и MongoDB в качестве баз данных и сделаны выводы о том, что комбинация Go+MySQL превосходит по использованию памяти и процессора, а Node.js+MySQL превосходит по времени отклика.

ОБСУЖДЕНИЕ РЕЗУЛЬТАТОВ

Можно сделать вывод, что использование gRPC-архитектуры имеет больше плюсов, чем использование стандартного подхода REST. В частности, gRPC имеет передачу сообщений быстрее в 7–10 раз, чем REST. Но для утверждения выставленной гипотезы о большей эффективности предложенной связки необходимо, кроме *времени отклика,* оценить такие характеристики производительности, как *загрузка процессора* и *использование памяти*. Так как в качестве буфера протокола используется Protobuf, то размер данных сокращается – этим подтверждается меньшее использование памяти.

Многопоточность, используемая gRPC, ускоряет передачу сообщений, делает её более эффективной, но не подтверждает меньшую загрузку процессора, что ограничивает применение связки Unity+gRPC для тех многопользовательских видеоигр, которые слишком требовательны к характеристикам сервера.

Итак, Golang при производстве многопользовательских игр на Unity показывает лучшие результаты, чем стандартные решения.

[18] СУБД (аббрев.) – система управления базой данных.

ЗАКЛЮЧЕНИЕ

Индустрия разработки видеоигр, или gamedev, на данный момент имеет малую степень систематизации знаний и пока находится вне фундаментальных исследований. Требуется создать научный базис, основываясь на котором, можно будет формировать гипотезы, теории, методы, в том числе, для повышения качества разработки видеоигр, для уменьшения переработок специалистов, для смещения баланса разработки видеоигр с практико-ориентированной к научно-ориентированной.

На данный момент фокусом научных интересов лаборатории разработки игр, визуализации, дополненной/виртуальной реальности (Digital Media Lab) ИТИС[19] КФУ[20] являются: нейронные сети в разработке видеоигр (см., например, [8, 9]); автоматизация и абстрагирование разработки видеоигр, включая поиск оптимальных подходов задачи автоматизации генерации прототипов видеоигр (см., например, [10]); систематизация и поиск наилучших методов вовлечения и погружения игроков видеоигр. Практические интересы включают применение современных технологий и знаний в области создания видеоигр.

В статье сравнены современные подходы к реализации многопользовательских видеоигр и обсуждена эффективность технологических приемов, позволяющих получать быстрые способы передачи данных между клиентом и сервером. Также приведены основные методы для отладки многопоточных приложений на Golang в качестве бэкенда игрового приложения в связке с системой разработки игровых приложений Unity.

Данная статья является одним из этапов к пониманию процессов эффективного создания видеоигр.

БЛАГОДАРНОСТИ

Работа выполнена за счет средств Программы стратегического академического лидерства Казанского (Приволжского) федерального университета («ПРИОРИТЕТ-2030»).

[19] ИТИС – Институт информационных технологий и интеллектуальных систем.
[20] КФУ – Казанский федеральный университет.

СПИСОК ЛИТЕРАТУРЫ

1. *Jitendra M.S.N.V., Amiripalli S.S., Surendra T., Rao R.V., Chowdary P.R.* A study on game development using unity engine // AIP Conference Proceedings. 2021. Vol. 2375. No. 1. P. 040001-1–040001-13.

2. *Polančec D., Mekterović I.* Developing MOBA games using the Unity game engine // 40th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO). 2017. P. 1510–1515.

3. *Sia B.N.C.H., Koh Z.W., Chung M., Chen Z., Chua S.H., Ganesan D.A.L., Kuah M.Y., Tey K.J., Yeoh W.E.* Cryptocoinopoly: A Real Time Online Multiplayer Board Game // 14th International Conference on Advanced Computer Theory and Engineering. 2021. P. 27–31.

4. *Yasir R.M., Asad M., Galib A.H., Ganguly K.K., Siddik M.S.* Godexpo: an automated god structure detection tool for Golang // Proceedings of the 3rd International Workshop on Refactoring. 2019. P. 47–50.

5. *Fernando R.* Evaluating Performance of REST vs gRPC. 2019. URL: https://medium.com/@EmperorRXF/evaluating-performance-of-rest-vs-grpc-1b8bdf0b22da (дата обращения: 15.07.2022).

6. Protocol Buffers. 2019. URL: https://developers.google.cn/protocol-buffers/docs/overview (дата обращения: 15.07.2022).

7. *Effendy F. Taufik, Adhilaksono B.* Performance comparison of web backend and database: A case study of node.js, Golang and MySQL, Mongo DB // Recent Advances in Computer Science and Communications. 2021. Vol. 14. No. 6. P. 1955–1961.

8. *Козар Б.А., Кугуракова В.В., Сахибгареева Г.Ф.* Структуризация сущностей естественного текста с использованием нейронных сетей для генерации трехмерных сцен // Программные продукты и системы. 2022. No. 3. С. 329–339.

9. *Кугуракова В.В., Абрамов В.Д., Костюк Д.И., Шараева Р.А., Газизов Р.Р., Хафизов М.Р.* Генерация трехмерных синтетических датасетов // Электронные библиотеки. 2021. Т. 24(4). С. 622–652.

10. *Сахибгареева Г.Ф., Кугуракова В.В.* Практики балансирования компьютерных игр // Программные системы: теория и приложения. 2022. Т. 13. No. 3. С. 255–273.

COMPARISON OF CLIENT-SERVER SOLUTIONS IN THE DEVELOPMENT OF MASSIVELY MULTIPLAYER ONLINE GAMES ON UNITY

I. R. Muhamethanov[1] [0000-0002-0562-1434], M. R. Khafizov[2] [0000-0001-7275-9102], A. V. Shubin[3] [0000-0002-6203-3268]

[1,2,3] *Institute of Information Technologies and Intelligent Systems, Kazan (Volga Region) Federal University, ul. Kremlyovskaya, 35, Kazan, 420008*

[1]ilnurhamay@mail.ru, [2]mrimsh@mrimsh.ru, [3]shubin.aleksey.kpfu@gmail.com

Abstract

This paper presents a critique of the traditional approach used to create a multiplayer game in the Unity real-time interactive application development system, especially in the case of a large number of concurrent users. As a hypothesis, an alternative option, which is not common, but which solves many of the problems of the previous approach, is proposed. Two client-server solutions have been compared for developing multiplayer online games in Unity, and the advantages of both approaches have been described for different cases. A game development architecture using a more up-to-date method is proposed: instead of the Mirror library, a standard toolkit for Unity development, microservices written in Golang are used. We present solid proofs of the preference of the alternative approach, the main advantage of which is the support of modern architecture providing high-speed communication between microservices, supported tests on messaging on different platforms.

The test results confirm the hypothesis put forth, and we can conclude that the Unity bundle with Golang is more effective for multiplayer video games.

The article also contains basic methods for debugging multi-threaded application in Golang bundled with Unity game development system and suggests a technological method that allows to get a fast way of data transfer between the client and the server.

Keywords*: game engine, microservices, Unity, Golang, Mirror, videogame, multiplayer.*

REFERENCES

1. *Jitendra M.S.N.V., Amiripalli S.S., Surendra T., Rao R.V., Chowdary P.R.* A study on game development using unity engine // AIP Conference Proceedings. 2021. Vol. 2375. No. 1. P. 040001-1–040001-13.

2. *Polančec D., Mekterović I.* Developing MOBA games using the Unity game engine // 40th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO). 2017. P. 1510–1515.

3. *Sia B.N.C.H., Koh Z.W., Chung M., Chen Z., Chua S.H., Ganesan D.A.L., Kuah M.Y., Tey K.J., Yeoh W.E.* Cryptocoinopoly: A Real Time Online Multiplayer Board Game // 14th International Conference on Advanced Computer Theory and Engineering. 2021. P. 27–31.

4. *Yasir R.M., Asad M., Galib A.H., Ganguly K.K., Siddik M.S.* Godexpo: an automated god structure detection tool for Golang // Proceedings of the 3rd International Workshop on Refactoring. 2019. P. 47–50.

5. *Fernando R.* Evaluating Performance of REST vs gRPC. 2019. URL: https://medium.com/@EmperorRXF/evaluating-performance-of-rest-vs-grpc-1b8bdf0b22da (Accessed 15 July 2022).

6. Protocol Buffers. 2019. URL: https://developers.google.cn/protocol-buffers/docs/overview (Accessed 15 July 2022).

7. *Effendy F.Taufik, Adhilaksono B.* Performance comparison of web backend and database: A case study of node.js, Golang and MySQL, Mongo DB // Recent Advances in Computer Science and Communications. 2021. Vol. 14. No. 6. P. 1955–1961.

8. *Kozar B., Kugurakova V., Sakhibgareeva G.* Structuring natural text entities using neural networks for generating 3D-scenes // Software & Systems. 2022. Vol. 35. No. 3. P. 329-339.

9. *Kugurakova V., Abramov V., Kostyuk D., Sharaeva R., Gazizov R., Khafizov M.* Generation of three-dimensional synthetic datasets // Russian Digital Libraries Journal. 2021. Vol. 24. No. 4.

10. *Sahibgareeva G., Kugurakova V.* Game Balance Practices // Program systems: theory and applications. 2022. Vol. 13. No. 3(54). P. 255–273.

СВЕДЕНИЯ ОБ АВТОРАХ

МУХАМЕТХАНОВ Ильнур Радикович – лаборант кафедры программной инженерии Института ИТИС КФУ. Сфера научных интересов: разработка видеоигр, информационные технологии.

Ilnur Radikovich MUKHAMETKHANOV – laboratory assistant of the software engineering department of the Institute of ITIS KFU. Research interests: video games development, information technologies.

e-mail: ilnurhamay@mail.ru

ORCID: 0000-0002-0562-1434

ХАФИЗОВ Мурад Рустэмович – старший преподаватель Института ИТИС КФУ. Сфера научных интересов: разработка видеоигр, виртуальная реальность, синтетические данные.

Murad Rustemovich KHAFIZOV – senior lecturer at the Institute of Information Technologies and Intelligent Systems, Kazan Federal University. Research interests: game development, virtual reality, synth data.

e-mail: murkorp@gmail.com

ORCID: 0000-0001-7275-9102

ШУБИН Алексей Витальевич – лаборант кафедры программной инженерии Института ИТИС КФУ. Сфера научных интересов – разработка видеоигр, игровой дизайн.

Aleksey Vitalevich SHUBIN – lab assistant at the Department of Software Engineering of the Institute of ITIS KFU. Research interest – videogame development, game design.

e-mail: shubin.aleksey.kpfu@gmail.com

ORCID: 0000-0002-6203-3268

Материал поступил в редакцию 16 сентября 2022 года