

УДК 004.451

DOI: 10.46548/21vek-2022-1157-0008

РАЗРАБОТКА И ИССЛЕДОВАНИЕ МОДЕЛЕЙ ПОДСИСТЕМЫ ОЧЕРЕДЕЙ СООБЩЕНИЙ В РЕКОНФИГУРИРУЕМОЙ ВЫЧИСЛИТЕЛЬНОЙ СИСТЕМЕ

© 2022

Мартышкин Алексей Иванович, кандидат технических наук, доцент,
заведующий кафедрой «Программирование»

Пензенский государственный технологический университет

(440039, Россия, Пенза, проезд Байдукова/ул. Гагарина, д. 1а/11, e-mail: alexey314@yandex.ru)

Синев Михаил Петрович, кандидат технических наук, доцент кафедры «Вычислительная техника»

Шеянов Николай Николаевич, студент кафедры «Вычислительная техника»

Нешко Дарья Олеговна, студент кафедры «Вычислительная техника»

Никишин Кирилл Игоревич, кандидат технических наук, доцент кафедры «Вычислительная техника»

Пензенский государственный университет

(440026, Россия, Пенза, улица Красная, 40,

e-mails: mix.sinev@gmail.com, snn1998@gmail.com, daria-uno@yandex.ru, nkipnz@mail.ru)

Аннотация. В статье рассмотрены основные виды архитектур построения приложений для высокопроизводительных реконфигурируемых вычислительных систем, их преимущества и недостатки, а также затронут такой важный вопрос, как взаимодействие данных при их передаче между приложением-производителем и приложением-потребителем в режиме реального времени как внутри одного приложения, так и взаимодействие между несколькими приложениями. По итогам проведенного исследования получено готовое приложение, позволяющее осуществлять высокопроизводительную синхронизацию данных между сервисами посредством брокера сообщений. Для этого выбраны наиболее подходящие технологии, настроены конфигурации отдельных частей, разработаны алгоритмы и установлены структуры входящих сообщений, а также определена структура базы данных для хранения объектов. Еще одной немаловажной задачей, решаемой в рамках исследования является отладка приложения, оптимизация его алгоритма, функциональное тестирование и имитационное моделирование, а также корреляция настроек приложения и ресурсов реконфигурируемой вычислительной системы, на которой разворачивается данное приложение, т.к., если приложение будет сконфигурировано неправильным образом, это может значительно сказаться на общей производительности системы, работающей в режиме реального времени. В завершении исследования рассмотрены вопросы производительности приложения при различных конфигурациях. По полученным результатам тестирования построены графики, наглядно отображающие эффективность работы при конкретном наборе параметров.

Ключевые слова: реконфигурируемая вычислительная система, система реального времени, высокопроизводительная вычислительная система, приложение, взаимодействие процессов, потребитель, производитель, производительность, передача сообщений.

DEVELOPMENT AND RESEARCH OF MODELS OF THE MESSAGE QUEUE SUBSYSTEM IN A RECONFIGURABLE COMPUTING SYSTEM

© 2022

Martyshkin Alexey Ivanovich, candidate of technical sciences, docent,
head of sub-department «Programming»

Penza state technological University

(440039, Russia, Penza, BaydukovProyezd / Gagarin Street, 1a/11, e-mail: alexey314@yandex.ru)

Sinev Mihail Petrovich, candidate of technical sciences,

associate Professor of sub-department «Computer engineering»

Sheyanov Nikolay Nikolaevich, student of sub-department «Computer engineering»

Neshko Daria Olegovna, student of sub-department «Computer engineering»

Nikishin Kirill Igorevich, candidate of technical sciences,

associate Professor of sub-department «Computer engineering»

Penza State University

(440026, Russia, Penza, Krasnaya Street, 40,

e-mails: mix.sinev@gmail.com, snn1998@gmail.com, daria-uno@yandex.ru, nkipnz@mail.ru)

Abstract. The article discusses the main types of application architectures for high-performance reconfigurable computing systems, their advantages and disadvantages, and also touches on such an important issue as the interaction of data during their transmission between the producer application and the consumer application in real time both within one application and interaction between several applications. Based on the results of the study, a ready-made application was obtained that allows for high-performance data synchronization between services through a message broker. For this purpose, the most suitable technologies have been selected, configurations of individual parts have been configured, algorithms have been developed and structures of incoming messages have been established, and the

structure of the database for storing objects has been determined. Another important task solved within the framework of the study is debugging the application, optimizing its algorithm, functional testing and simulation, as well as correlation of application settings and resources of the reconfigurable computing system on which this application is deployed, because if the application is configured incorrectly, this can significantly affect the overall performance of the system running in real time. At the end of the study, the issues of application performance research under various configurations are considered. Based on the obtained test results, graphs are constructed that visually display the efficiency of work with a specific set of parameters.

Keywords: reconfigurable computing system, real-time system, high-performance computing system, application, process interaction, consumer, producer, performance, message transmission.

Введение. В современном мире все сферы деятельности оказались под влиянием информационных технологий. При функционировании всевозможных комплексов и систем (например, высокопроизводительных реконфигурируемых вычислительных систем (РВС)) обрабатывается колоссальное количество информации при операциях фильтрации, анализа, классификации и преобразования.

Важная часть работы любого приложения, функционирующего на конкретной аппаратной платформе – предоставление конечному пользователю актуальных и достоверных данных в режиме реального времени, т.к. любые, даже самые незначительные, задержки могут вызвать огромные накладные расходы и убытки. В связи с чем необходимо заранее провести детальный анализ для выявления наилучшего способа построения приложения, функционирующего на конкретной РВС. В крупных приложениях часто возникает ситуация, когда источником необходимых для работы данных является стороннее приложение. Этот факт накладывает дополнительные сложности на синхронизацию поступающих данных между производителем и потребителем данных, но что произойдет, если приложений-потребителей будет несколько? В такой ситуации необходимо произвести аналитическое исследование по построению архитектурного решения не только внутри приложения, но и во взаимодействии его с другими связанными приложениями.

В статье объектом исследования выступает высокопроизводительный обмен данными с множественными получателями. Предметом исследования являются способы взаимодействия приложений при синхронизации данных в режиме реального времени и наличии нескольких приложений-потребителей, а также оптимальные конфигурации приложений для наиболее эффективного выполнения поставленных задач в рамках имеющихся ресурсов РВС.

Цель настоящей работы – выявление и анализ наилучшего решения для обмена данными между приложением-производителем и приложением-потребителем в режиме реального времени, построение архитектуры и реализация приложения-потребителя, осуществляющего синхронизацию данных, а также выбор наиболее подходящей конфигурации серверных настроек и вычислительного оборудования РВС.

Материалы и результаты исследования. Проведение анализа предметной области начнем с архитектуры программного обеспечения (ПО). Известно большое количество разных подходов к проектированию

архитектуры ПО [1, 2]. Уже продолжительное время в мире разработки ПО преимущественно используются две популярные архитектуры разработки приложений: монолитная и микросервисная [3]. Выбор конкретного типа архитектуры зависит от решаемых задач и требований, выдвинутых к приложению. Анализируя задачу, в которой данные для синхронизации поступают от одного источника ко многим получателям и, принимая во внимание, что полученные данные в дальнейшем будут использоваться в рамках реализации других процессов, наилучшим решением для построения приложения будет использование микросервисной архитектуры, т.к. если возникнет потребность в синхронизации данных от других источников или нужно будет передавать эти данные в какой-либо другой модуль обработки, то лучше будет реализовать каждую из этих частей в виде отдельных сервисов (сервис синхронизации 1, сервис синхронизации 2, сервис обработки 1, сервис обработки 2 и т.д.) [4]. Одним из основных вопросов при проектировании микросервисной архитектуры является взаимодействие между ее составными компонентами с целью обмена данными [5].

Первым способом является взаимодействие посредством прямых *HTTP*-запросов. При этом запросы могут выполняться как синхронно, так и асинхронно. Данный способ является самым простым в исполнении, но он не подходит для масштабирования, т.к. если нескольким сервисам необходимо получать одни данные от этого сервиса, то необходимо будет реализовать все каналы связи. При большом количестве сервисов в архитектуре приложения данный подход будет требовать все больше ресурсов на поддержание корректной работы всех информационных потоков [6]. Вторым способом взаимодействия микросервисов является событийная коммуникация. Такой подход позволяет сохранять слабую связь между сервисами и при этом отслеживать лишь те события, которые необходимы для работы конкретного сервиса [7]. При таком подходе между сервисами не передаются полезные данные, вследствие этого, если один из сервисов является производителем данных, которые потребляет другой сервис, то использовать событийную коммуникацию является невозможным решением. Третьим рассматриваемым способом общения между микросервисами является обмен сообщениями. При таком варианте сервисы не взаимодействуют друг с другом напрямую, а используют для этого особый механизм – брокер сообщений [9-10]. Все сервисы имеют доступ

к брокеру и именно он является центральным узлом. Подход обмена сообщениями использует шаблон публикация-подписка, при котором множество сервисов может получать данные от одного производителя. Сложностями этого подхода является отсутствие гарантии доставки сообщений, а также согласование структуры отправляемых сообщений. Рассмотрев представленные варианты, сделаем вывод, что каждый подход решает конкретные проблемы и выбор наилучшего варианта зависит от поставленной задачи. Все представленные выше варианты предоставляют большие возможности по взаимодействию между сервисами.

Основная задача, рассматриваемая в статье – высокоэффективная обработка входящих сообщений для синхронизации данных. Задача состоит в том, что есть приложение-производитель данных, оно производит данные, которые необходимо получить, обработать и привести все атрибуты объекта к соответствию конкретной системы. Основной особенностью является то, что приложений-потребителей множество и каждое из них разрабатывается отдельной командой разработчиков для реализации конкретных функций. Это накладывает некоторые ограничения на формат сообщений – они должны содержать сведения, подходящие для всех приложений. Другой сложностью является то, что каждый объект имеет многочисленные вложенные объекты, обработка которых занимает продолжительное время, т.к. в рамках обработки выполняется большое количество SQL-запросов к БД на получение внутренних атрибутов, используемых в рамках конкретной системы. При этом приложение должно работать в режиме реального времени и не допускать отставания от приложения-производителя, а также не должно допускать обработки некорректных данных.

Для решения данной задачи предлагается реализовать отдельный сервис синхронизации данных, главной целью которого будет получение, первичная обработка и сохранение корректных данных в БД системы. Полученный модуль в дальнейшем, при необходимости, можно будет легко заменить на другой.

Для синхронизации данных между одним производителем и несколькими потребителями лучшей стратегией взаимодействия приложений будет использование подхода обмена сообщениями по типу публикация-подписка [8]. Данный подход позволяет передавать сообщение, содержащее полезные данные, всем приложениям, подписанным на данную тему. Для того, чтобы минимизировать возможные проблемы такого подхода и получить наиболее эффективный вариант обработки сообщений будем использовать подходящие программные средства, например, *Apache Kafka*.

Основным звеном приложения является брокер сообщений. Существует множество различных вариантов, таких как *ActiveMQ*, *Apache Kafka* и другие. В работе будем использовать open source брокеры сообщений. Для того чтобы окончательно решить какой из брокеров является наилучшим для решения поставлен-

ной задачи, нужно сравнить их производительность, для этого воспользуемся данными, представленными в работе «*Performance Comparison of Message Queue Methods*» (2019) автора Raje Sanika [11].

Рассмотрим четыре основных сценария: 1 производитель – 1 потребитель, 1 производитель – 2 потребителей, 2 производителя – 1 потребитель и 2 производителя – 2 потребителей. При исследовании будем рассматривать такие параметры, как пропускная способность, измеряемая в сообщениях в секунду, и задержка на количество сообщений, измеряемая в мс (рис. 1).

Как видно из рисунка 1, наибольшую пропускную способность и наименьшую задержку имеет *Apache Kafka*, поэтому для минимизации недостатков стратегии взаимодействия между сервисами посредством обмена сообщений будем использовать именно ее. *Apache Kafka* – потоковая платформа, публикующая потоки данных и подписки на них, а также осуществляет их хранение и обработку. Платформа предоставляет огромные возможности масштабирования, при этом не требует создания множества отдельных брокеров сообщений, а предоставляет централизованную платформу для взаимодействия микросервисов. Другим преимуществом *Apache Kafka* является возможность тонкой настройки времени хранения сообщений, что обеспечивает репликацию, целостность и хранение данных в течение любых промежутков времени. И, наконец, потоковая обработка повышает уровень абстракции, что в свою очередь позволяет *Apache Kafka* на основе потоков данных вычислять производные потоки и наборы данных динамически [12-13].

Последовательность синхронизации данных от сервиса-производителя до сервиса-потребителя состоит из следующего алгоритма, приведенного на рисунке 2.

Вначале сервис-производитель, в соответствии с некоторыми требованиями, выполняет формирование и обработку данных на своей стороне. После чего преобразует их в легковесный формат передачи данных по сети JSON и отправляет это сообщение в брокер *Apache Kafka*. В это время сервис-потребитель прослушивает темы, на которые он подписан на наличие новых сообщений. С этого момента начинается обработка сообщений в сервисе-потребителе.

Важной частью взаимодействия между сервисами является соглашение о структуре, пересылаемого сообщения [14-16]. Данное сообщение состоит из служебной информации, которая вкладывается самим брокером *Kafka*, а также из части наших данных, хранящихся в поле “value” (табл. 1).

Как только публикуются новые сообщения, слушатель вычитывает пакет сообщений, размер которого настраивается, и передает его в обработку в сервис обработки сообщений *MessageService*. Сообщения разбиваются на *LinkedHashMap<Integer, Value>*, где ключом является глобальный идентификатор во всех системах (производителе и других потребителях), а значением является информационная структура со всеми ее данными [17]. Каждое сообщение преобразуется в целевые информационные объекты. После

чего с помощью класса *Validator* происходит проверка информационных объектов на корректность данных, если данные некорректны, то в журнал логов выводится соответствующее сообщение, а обработка данного сообщения не производится. После чего происходит помещения данных информационного объекта в карту

[17-18]. Данный алгоритм позволяет не обрабатывать промежуточные данные объекта в рамках данного пакета сообщений. Это позволяет снизить количество вычислений, т.к. при наличии в пакете сообщений 10 записей, касающихся одного и того же объекта, она будет обработана всего 1 раз.

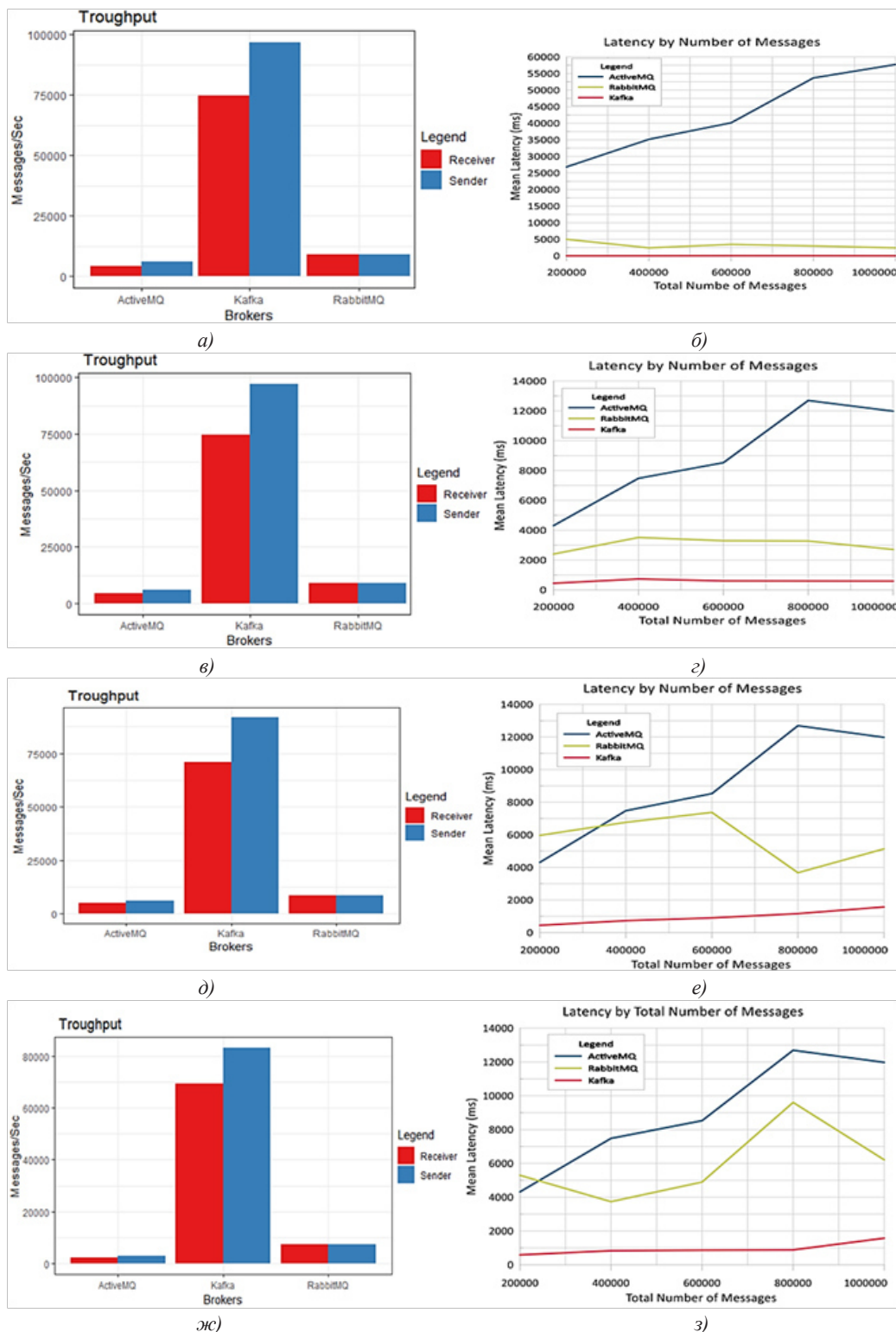


Рисунок 1 – Пропускная способность (а); задержка на количество сообщений (б) (1 производитель – 1 потребитель); пропускная способность (в); задержка на количество сообщений (г) (1 производитель – 2 потребителя); пропускная способность (д); задержка на количество сообщений (е) (2 производителя – 1 потребитель); пропускная способность (ж); задержка на количество сообщений (з) (2 производителя – 2 потребителя)

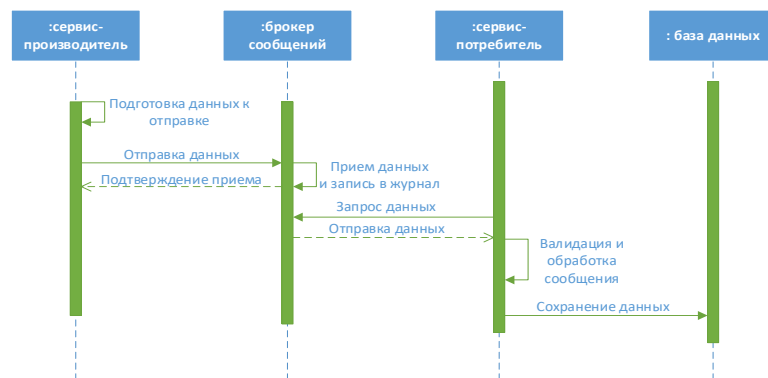


Рисунок 2 – Алгоритм синхронизации данных от сервиса-производителя до сервиса-потребителя

Таблица 1 – Атрибуты входящего json-сообщения

Название атрибута	Описание атрибута	Пример
topic	Тема, из которой получена эта запись	paper.topic
partition	Раздел темы, из которого получена эта запись	0
leaderEpoch	Необязательный лидер эпохи записи	0
offset	Смещение этой записи в соответствующем разделе Кафки	152
CreateTime	Время создания записи (мс)	1620386532558
serializedkeysize	Длина сериализованного ключа	-1
serializedvaluesize	Длина сериализованного значения	290
headers	Список заголовков записи	
isReadOnly	Заголовки только для чтения	false
key	Ключ записи, если он существует	null
value	Содержание записи	

Как только сформирована конечная карта всех обрабатываемых сообщений, она передается в *ThreadPoolTaskExecutor* – класс реализующий, менеджмент параллельной обработки данных. Каждое сообщение из карты передается в *MessageProcessor*, где происходит его обработка, параллельно с обработкой других сообщений. Размер *ThreadPoolTaskExecutor* задается через файл настроек [19-20].

В *MessageProcessor* происходит вся дальнейшая обработка сообщения, приведение к соответствию глобальных атрибутов объекта и локальных, вычисление некоторых параметров и т.д. В конце итоговые данные сохраняются в БД.

Как только обработка всех сообщений пакета закончена, то слушатель готов к приему нового пакета сообщений.

Имитационное моделирование. Моделирование на реальной вычислительной машине производится с целью максимального улучшения алгоритма и подбора такой конфигурации серверных настроек и материального обеспечения, которые обеспечат наивысшую эффективность в рамках поставленных ограничений:

- количество потоков, обрабатывающих пакет входящих сообщений;
- размер очереди, а именно максимальное количество сообщений в очереди;
- размер пакета.

Для того чтобы провести тестирование производительности необходимо создать необходимый пул входных значений. Первоначально этот пул состоит из одной тысячи тестовых сообщений.

Проводимое моделирование будет заключаться в том, что мы будем задавать различные значения для количества потоков, количества сообщений, размера

очереди и размера пакета, после чего будем производить прием заданного количества тестовых сообщений, замерять конечное время на их полную обработку и выполнять расчеты коэффициентов прироста производительности.

Измерения производились на виртуальной PBC, развернутой на машине – Ноутбук *DELL G5 5590, 15.6", IPS, Intel Core i7 9750H 2.6ГГц, 16ГБ, 1000ГБ, 256ГБ SSD, nVidia GeForce RTX 2060 - 6144 Мб, Windows 10*.

При этом исследования разделены на ряд итераций, внутри каждой из которых происходит изменение количества потоков обработки с последующей фиксацией результатов, при этом размер очереди, количество сообщений и размер вычитываемого пакета останутся неизменными. В рамках разных итераций будет изменяться размер очереди, количество сообщений и размер вычитываемого пакета. Результаты представлены в виде графиков времени обработки сообщений в зависимости от количества потоков-обработчиков.

Первая итерация была базисной, от которой мы в дальнейшем отталкивались и меняли параметры в соответствии с результатами проверок. Моделирование выполнялось для следующих данных:

- общее количество сообщений – 1000 штук;
- максимальное количество сообщений в очереди – 400 штук;
- количество сообщений, вычитываемых за одну операцию – 400 штук.

Пунктирной линией на графике, представленном на рисунке 3 отмечено контрольное значение по допустимому времени обработки. Контрольное значение определяется исходной прикладной задачей и может варьироваться в зависимости от постановки целевого

условия. Как можно увидеть из рисунка, при количестве потоков меньше 16 мы не можем преодолеть контрольное значение для времени обработки в 100000 мс, поэтому отсечем эту область при дальнейшем тестировании.

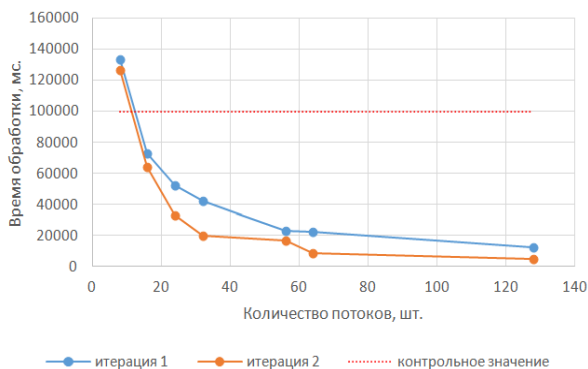


Рисунок 3 – Время обработки сообщений в зависимости от количества потоков-обработчиков на итерации 1

Моделирование на второй итерации выполнялось для следующих данных:

- общее количество сообщений – 1000 штук;
- максимальное количество сообщений в очереди – 128 штук;
- количество сообщений, вычитываемых за одну операцию – 512 штук.

Отметим, что присутствует рост производительности в связи с тем, что количество потоков и размер обрабатываемого пакета коррелируют между собой. При выставлении значений размера пакета и очереди кратным количеству потоков, получаем прирост эффективности.

Заключение В ходе проведения исследований разработана программная реализация приложения, осуществляющего синхронизацию данных между сервисами с типом взаимодействия, «публикация-подписка» в микросервисной архитектуре в режиме реального времени. Примером подобных приложений могут являться сервисы синхронизации различных классификаторов нормативно-справочной информации, а также реестров материально-технического обеспечения.

В работе проанализированы наиболее популярные подходы к проектированию архитектуры ПО высокопроизводительных реконфигурируемых вычислительных систем, а также способы взаимодействия составных частей данных систем между собой. Наглядно показана возможность применения брокера сообщений *Apache Kafka* для синхронизации данных.

Осуществлено функциональное тестирование и моделирование конечного приложения. Выполнен анализ полученных результатов и представлен выбор наилучшей конфигурации настроек компонентов высокопроизводительных реконфигурируемых вычислительных систем для выполнения задачи, а именно установлен рост производительности обработки при установке количества сообщений в пакете кратным количеству потоков обработки.

СПИСОК ЛИТЕРАТУРЫ:

1. ГОСТ Р 57100-2016/ISO/IEC/IEEE 42010:2011 Системная и программная инженерия. Описание архитектуры. – 52 с.
2. Крачтен Ф. Введение в Rational Unified Process. 2-е изд. М.: Вильямс, 2002. – 240 с.
3. Ричардсон К. Микросервисы. Паттерны разработки и рефакторинга. — СПб.: Питер, 2019. – 544 с.: ил. — (Серия «Библиотека программиста»).
4. Smid A., Wang R., Cerny T. Case study on data communication in microservice architecture //Proceedings of the Conference on Research in Adaptive and Convergent Systems. – 2019. – С. 261-267.
5. Равал С., Децентрализованные приложения. Технология Blockchain в действии. – СПб.:Питер, 2017. — 240 с.
6. Gadea C. et al. A reference architecture for real-time microservice api consumption //Proceedings of the 3rd Workshop on CrossCloud Infrastructures & Platforms. – 2016. – С. 1-6.
7. Stubbs J., Moreira W., Dooley R. Distributed systems of microservices using docker and serfnode //2015 7th International Workshop on Science Gateways. – IEEE, 2015. – С. 34-39.
8. Laigner R., Zhou Y., Salles M. A. V. A distributed database system for event-based microservices //Proceedings of the 15th ACM International Conference on Distributed and Event-based Systems. – 2021. – С. 25-30.
9. Dinh-Tuan H., Beierle F., Garzon S. R. MAIA: a microservices-based architecture for industrial data analytics //2019 IEEE International Conference on Industrial Cyber Physical Systems (ICPS). – IEEE, 2019. – С. 23-30.
10. Son S. C. et al. LwM2M based IoT Microservice Model with Replicas Synchronization Technique //2020 International Conference on Information and Communication Technology Convergence (ICTC). – IEEE, 2020. – С. 1802-1804.
11. Raje Sanika, "Performance Comparison of Message Queue Methods" (2019). UNLV Theses, Dissertations, Professional Papers, and Capstones. 3746.
12. Нархид Н., Шапира Г., Палино Т., Apache Kafka. Поточковая обработка и анализ данных. — СПб.: Питер, 2019. — 320 с.
13. Hicham R., Anis B. M. Processes meet Big Data: Scaling process discovery algorithms in Big Data environment //Journal of King Saud University-Computer and Information Sciences. – 2021.
14. Москвичева К. С., Долгачев М. В. Бумажная промышленность: Kafka против RabbitMQ. Сравнительное исследование двух отраслевых эталонных реализаций publish/subscribe //Форум молодёжной науки. – 2020. – №. 4. – С. 3-17.
15. Thein K. M. M. Apache kafka: Next generation distributed messaging system //International Journal of Scientific Engineering and Technology Research. – 2014. – Т. 3. – №. 47. – С. 9478-9483.
16. Le Noac'H P., Costan A., Bougé L. A performance evaluation of Apache Kafka in support of big data streaming applications //2017 IEEE International Conference on Big Data (Big Data). – IEEE, 2017. – С. 4803-4806.
17. Байэр К., Кинг Г., Грегори Г., Java Persistence API и Hibernate / пер. с англ. Д. А. Зинкевича; под науч. ред. А. Н. Киселева. – М.: ДМК Пресс, 2017. – 632 с.: ил.
18. Stärk R. F., Schmid J., Börger E. Java and the Java virtual machine: definition, verification, validation. – Springer Science & Business Media, 2012.
19. Мартин Р. Чистая архитектура. Искусство разработки программного обеспечения. — СПб.: Питер, 2018. — 352 с.: ил. — (Серия «Библиотека программиста»).
20. Deinum M. Java Enterprise Services //Spring Boot 2 Recipes. – Apress, Berkeley, CA, 2018. – С. 239-256.

Исследование выполнено за счет гранта Российского научного фонда № 21-71-00110, <https://rscf.ru/project/21-71-00110/>.

Статья поступила в редакцию 04.02.2022

Статья принята к публикации 10.03.2022