Информатика и вычислительная техника

УДК 62; 004; 007 DOI: 10.14529/ctcr210101

ОБ ОДНОМ ПОДХОДЕ К РЕАЛИЗАЦИИ ИНФОРМАЦИОННОЙ ИНФРАСТРУКТУРЫ ОБНОВЛЯЕМОГО ИНФОРМАЦИОННОГО ПОИСКА

А.А. Шинкарев

ООО «Софтмаст-ИТ», г. Челябинск, Россия

Основные задачи в сфере информационных технологий, стоящие перед бизнесом сегодня, так или иначе касаются обработки информации и поиска новых сведений в ней. Используются статистические методы, модели машинного обучения и более простые методы и модели. Однако всем решениям, направленным на поиск информации, необходима информационная инфраструктура, которая была бы адекватна основным требованиям, предъявляемым к системам такого рода. **Цель исследования:** описание основных функциональных и технических требований, которые предъявляются к современным системам, в задачи которых входит реализация обновляемого информационного поиска; формирование основополагающих архитектурных предложений по дизайну системы в целом и ее ключевых частей в частности; выявление основных составляющих частей информационной системы и подходов к решению ключевых задач для решений, построенных на основе механизма обмена сообщениями. **Материалы и методы.** Рассматривается современная постановка задачи по созданию корпоративных информационных систем обновляемого информационного поиска. Сравниваются наиболее интересные в рамках рассматриваемой постановки задачи брокеры обмена сообщениями. **Результаты.** В статье обосновывается актуальность задачи создания информационных систем обновляемого поиска информации. Делается постановка задачи по созданию систем такого рода. Дается обзор дизайна архитектуры решения на высоком уровне абстракции. Раскрывается модульный состав рассматриваемой информационной системы. Обосновывается и принимается решение использовать в качестве брокера обмена сообщениями инструмент Kafka. Разбираются нюансы технических решений для устранения проблем дублирующихся записей и фильтрации последних результатов поиска информации.

Ключевые слова: корпоративные информационные системы, обновляемый информационный поиск, брокер обмена сообщениями, анализ данных, Kafka.

Введение

Сегодня поиск информации и анализ существующих данных являются важными задачами для такого рода деятельности, как информационные технологии, причем в самом широком смысле. Трудно представить, сколь сложный инструментарий скрыт за строкой информационного поиска. Для реализации программной и аппаратной составляющих систем такого рода необходимо огромное количество человеко-лет. Осложняют эту задачу и такие критически важные требования, как необходимость постоянно эволюционировать, то есть скачкообразно совершенствовать подходы и при этом поддерживать бесперебойное предоставление сервиса конечным пользователям.

Менее жесткие, но все же высокие требования предъявляются и к анализу данных и поиску новой информации в рамках корпоративных информационных систем. Важными задачами, чью инфраструктурную поддержку необходимо обеспечивать, являются поиск новых тенденций, агрегирование уже существующих данных и получение из них новых сведений, а также использование существующих корпоративных сервисов для получения ответов на вопросы, которые появляются у бизнес-пользователей к данным, создаваемым в ходе работы их бизнеса [1]. И настолько же важно сделать эти данные непротиворечивыми и пополняемыми с течением времени, когда появляются новые модели и методы работы с данными или просто хорошая идея о том, как извлечь новую полезную информацию, которую потом можно превратить в выгоду для компании.

Какими бы амбициозными ни были современные высокотехнологичные компании, для того чтобы работать с данными по-новому, необходимы новые современные инструменты, а именно информационная инфраструктура, которая бы могла поддержать реализацию новых идей. Такого рода связующим звеном сегодня все чаще становятся так называемые очереди сообщений, через которые идет обмен командами и событиями между подсистемами информационной системы компании [2]. Однако сам по себе инструмент обмена сообщениями недостаточен для решения задачи создания системы обновляемого информационного поиска. Необходима четкая постановка задачи, подход к ее решению с минимально необходимыми деталями реализации и архитектурный дизайн системы в целом.

1. Постановка задачи

Сформируем ключевые функциональные и технические требования к информационной системе обновляемого информационного поиска, которая могла бы в достаточной мере эффективно отвечать на вызовы, стоящие сегодня перед компаниями, извлекающими выгоду из данных за счет их умного анализа и предоставления в понятной для широкого круга людей форме.

Необходимо разработать дизайн системы по поиску информации в ответ на запросы пользователей. Ключевыми особенностями данной системы должны стать следующие функциональные и технические требования.

1. Расчеты должны выполняться асинхронно, то есть пользователи не обязаны ждать получения всех результатов за компьютером.

2. Результаты должны поступать к пользователям системы постепенно, по мере их готовности. Система должна отображать новые результаты, как только они будут сохранены, нет необходимости ждать, пока будет посчитано все.

3. Отображаемые результаты должны быть непротиворечивы. При одинаковых параметрах запросов разные пользователи должны видеть одни и те же результаты.

4. Должна существовать органично встроенная возможность добавлять новые поисковые механизмы, создающие новые результаты как по ранее созданным запросам, так и по новым.

5. Должна быть возможность по требованию пересчитать результаты ранее созданного и обсчитанного запроса, например, при обнаружении ошибок в расчетах.

6. Система должна быть готова к обработке таких ситуаций, как множественная доставка одного и того же сообщения потребителю, временная потеря сетевого соединения между частями системы.

7. Части системы не должны зависеть от языка реализации других частей. То есть разные подсистемы могут быть написаны на разных языках программирования.

8. Уровень представления и уровень расчетов должны быть слабо связаны между собой. Уровень представления зависит только от контракта уровня расчетов. Если в системе появляются новые типы данных, интерфейс должен уметь отображать их в простейшем виде, детальная реализация отображения может быть поставлена конечным пользователям позже, отдельно от обновлений логики расчетов.

2. Архитектурный дизайн системы

Для реализации предъявляемых функциональных и технических требований предлагается использовать микросервисный подход [3–6]. Это позволит избежать создания монолитной системы и реализовать гибкую инфраструктуру, которая сможет удовлетворить предъявляемым требованиям к расширяемости логики расчетов, возможности использовать разные языки программирования и слабосвязанности уровня представления и уровня расчетов.

В качестве связующего звена между частями системы предлагается использовать брокер обмена сообщениями [7]. Из решений с открытым исходным кодом, которые широко используются сегодня, можно выделить RabbitMQ и Kafka [8–10]. RabbitMQ представляет собой классический брокер обмена сообщениями, который не хранит их в очереди после успешной обработки. В отличие от RabbitMQ брокер Kafka позволяет хранить сообщения и после их обработки, в том числе давая возможность подключать новых потребителей сообщений как учитывая всю предыдущую историю запросов, так и не учитывая, работая только с запросами, отправленными после

регистрации нового потребителя. Это реализуется такими режимами потребления, как самый ранний (earliest) и последний (latest) соответственно [11].

В итоге в качестве брокера обмена сообщениями предлагается использовать Kafka. Это решение должно позволить удовлетворить предъявляемым требованиям к расширяемости логики расчетов, учитывая или не учитывая историю запросов. Также связывание сервисов системы между собой через брокер сообщений позволяет частично учитывать случаи повторной доставки и разрывов соединения при написании программного кода. Обработка ситуации ошибки расчетов, которая должна быть исправлена для всех запросов, также может быть реализована с использованием технических особенности работы Kafka, в частности с помощью версионирования потребителя в имени группы потребителей.

Само по себе использование брокера обмена сообщениями позволяет производить расчеты асинхронно. В сущности отправитель запроса на обработку и получение результатов не знает, кем будет выполнена процедура и будет ли она выполнена в принципе.

Рассмотрев основные архитектурные решения, такие как применение микросервисной архитектуры и использование брокера обмена сообщениями Kafka, разделим условную систему на следующие составные части.

1. Пользователи – инициаторы первоначальных запросов поиска информации.

2. Web Application – веб-приложение, доступное с помощью веб-браузера, в котором происходит заполнение формы при создании запроса пользователями.

3. Web API – веб-сервер, в который приходят запросы пользователей на поиск информации и который сохраняет их в базу данных запросов и результатов.

4. База данных запросов и результатов – база данных, которая хранит в себе данные пользователей, исходные данные запросов и полученные результаты. Возможно разделение этого хранилища на несколько отдельных баз данных, причем возможно применение как SQL, так и NoSQL решений [12, 13].

5. Брокер обмена сообщениями – связующее звено между частями системы, которое обеспечивает хранение сообщений и реализует механизм их обработки потребителями.

6. Сервисы обсчета – части системы, где происходит поиск информации по заданным критериям из изначального запроса и по специализации каждой конкретной группы сервисов. Сервисом расчета теоретически может являться как человек, сидящий за компьютером и реализующий определенный поиск, так и «умный» алгоритм поиска и анализа данных. Время обработки конкретного запроса пользователя может сильно отличаться от одной группы сервисов расчета к другой.

7. Сервис обработки результатов – сохраняет результаты работы сервисов обсчета в базу данных запросов и результатов.

8. Сервис мониторинга – обеспечивает наблюдение за состоянием и непротиворечивостью данных системы. Сигнализирует в случае неполноты данных, превышения ожидаемого времени обработки и передачи данных между частями системы, отказа частей системы.

Дизайн системы базируется на концепции потока событий и обмена сообщениями. Сервисы общаются друг с другом через брокера за счет обмена сообщениями через него. То есть контактируют только с шиной сообщений, но никогда друг с другом напрямую. Весь процесс взаимодействия асинхронный, то есть никто не ждет, когда его сообщение будет обработано всеми заинтересованными сторонами.

Очередь первоначальных запросов не должна очищаться и терять историю сообщений. В этом плане она должна быть реализована не как классическая fire and forget очередь, когда после обработки сообщение удаляется. Наоборот, очередь должна представлять собой хранимый лог сообщений (persistent messages log).

Очередь результирующих сообщений может не хранить их всегда, а иметь период очистки (retention period) и удалять сообщения, например, по прошествии недели.

Как только часть результата получена и сохранена в базе данных, ей присваивается уникальный идентификатор и она может быть кэширована (cached) до тех пор, пока не будет произведен принудительный перерасчет результатов по набору входных параметров.

Рассмотрев архитектурные особенности системы на высоком уровне, перейдем к детализации технической реализации слоя хранения результатов.

3. Обработка результатов и слой хранения данных

Если рассматривать реляционное хранилище для результатов, полученных сервисами обсчета на поисковые запросы пользователей, то предлагается хранить их в таблице с как минимум следующим набором обязательных атрибутов:

1) result_hash – хранит MD5-хэш, полученный по объекту результата обсчета, что позволяет считать контрольную сумму по результатам, в том числе для быстрого поиска одинаковых результатов;

2) result_type – хранит тип полученного результата;

3) schema_version – хранит версию контракта данных результата, что позволяет хранить в одной таблице сериализованный объект результата разных версий;

4) processor_type – уникальный в пределах системы человекочитаемый идентификатор типа сервиса обсчета, может существовать несколько экземпляров сервиса с одним и тем же значением этого атрибута, позволяет строить на его основе имя группы потребителей для брокера обмена сообщениями;

5) processing_occurrence_number – хранит в себе порядковый номер повторной обработки исходного запроса, который привел к созданию этого результата;

6) processing_session_id – уникальный идентификатор сессии запуска конкретного экземпляра сервиса обсчета, который получается за счет объединения текущей временной метки и UUID, что делает этот атрибут упорядочиваемым хронологически, что необходимо для поиска наиболее актуальных результатов и отфильтровывания устаревших;

7) message_id – уникальный идентификатор сообщения с результатом обсчета, которое генерируется каждый раз, когда мы пытаемся послать сообщение брокеру, позволяет на его основе строить логику обработки повторной доставки одного и того же результата.

Рассмотрев основные атрибуты таблицы результатов, перейдем к тому, для чего эти атрибуты предлагаются, а именно решению проблемы дублирующихся записей одних и тех же результатов.

4. Устранение дублирующихся результатов

Предлагается использовать операцию SQL Merge для вставки результатов, гарантируя при этом отсутствие дублей [14].

Для того чтобы предлагаемый механизм работал ожидаемым образом, необходимо определить ограничения на уникальность комбинаций атрибутов, описанных ранее.

Необходимо создать уникальный индекс на атрибут message_id, чтобы предотвратить образование дублирующихся записей при повторной доставке одного и того же сообщения от брокера обмена сообщениями.

Также необходим уникальный индекс по следующим атрибутам: parameters_id, result_type и result_hash. Этот индекс гарантирует, что для одной и той же комбинации входных параметров поиска мы не добавим одинаковые результаты одного и того же типа более чем 1 раз, даже если у них будут разные message_id. Например, сервис обсчета создает 3 результата и успешно отправляет только первые 2 из них, потом аварийно завершается и заново пробует обработать и отправить те же 3 результата, на этот раз успешно. Таким образом, нам необходимо сохранить 2 результата от первой попытки и только 1 результат от второй.

Операция SQL Merge используется, чтобы реализовать групповое потребление и сохранение результатов (batch consumption), при этом сохраняя гарантии транзакций ACID [15]. Это позволяет достигать большей пропускной способности обработки результатов по сравнению с обработкой и сохранением результатов по одному за раз с созданием транзакции на каждый отдельный результат. Также использование операции SQL Merge позволяет элегантно обрабатывать случай попытки вставить уже существующие данные и делать эту проверку средствами, встроенными в СУБД.

5. Фильтрация последних актуальных результатов

Предлагается реализация следующего механизма фильтрации и возврата пользователям только последних результатов обсчета в случае, когда процесс поиска повторялся несколько раз.

Среди всех результатов целевого result_type от всех сервисов обсчета, для которых определен одинаковый parameters_id, необходимо получить максимальное значение атрибута

schema_version и по нему получить список из processing_session_id, в рамках которых были получены результаты данной, максимальной, версии данных этого типа результата для всех затронутых типов сервиса расчета. После этого мы запрашиваем все результаты данного parameters_id для заданного result_type и попадающего в полученный на предыдущем шаге список processing_session_id.

Этот довольно сложный подход к фильтрации и, более того, имеющий определенные недостатки с точки зрения производительности. Однако он позволяет решать поставленные задачи по отбрасыванию старых результатов при появлении новых.

Одним из недостатков предлагаемого решения можно выделить случай генерации нового результата с новым значением атрибута schema_version для комбинации параметров поиска. В этом случае мы отбросим все ранее полученные результаты этого типа и вернем только единственный новый результат, что в определенных условиях может быть нежелательным поведением системы.

Заключение

Развитие информационной инфраструктуры и подходов к решению распространенных задач бизнеса продолжается сегодня довольно высокими темпами. Компании создают свое программное обеспечение или заказывают его у сторонних организаций. И при этом все стараются извлечь максимальную выгоду из накапливаемых данных, занимаются постоянной ретроспективой своих процессов, а также стараются прогнозировать дальнейшее развитие ситуации.

Таким образом, становится актуальной проблема поддержки этих амбициозных задач за счет создания и использования информационных систем и их инфраструктуры, адекватных специфике задачи поиска информации и актуализации результатов. Решая задачи, стоящие перед бизнесом, важно искать компромисс между желанием написать решение «под ключ» с нуля и использованием платных монструозных корпоративных информационных систем.

Связующим звеном разрозненных частей общей информационной системы может стать очередь сообщений. Это один из широко используемых сегодня вариантов борьбы со сложностью и решения проблемы горизонтальной масштабируемости. Однако мало лишь использования классической очереди сообщений для определенного класса задач, таких как обновляемый информационный поиск. Необходима поддержка режима работы брокера без удаления обработанных сообщений из очереди, тем самым реализуя хранимый лог сообщений. Подходящим по требованиям открытым (open source) решением является брокер сообщений Kafka.

Использованием Kafka позволяет соблюсти тот самый баланс между написанием своего собственного программного обеспечения для решения всей задачи и использованием полностью готового решения. Мы получаем инструмент, реализующий «из коробки» важные механизмы, такие как хранимый лог сообщений, групповая подписка на новые сообщения, режимы потребления последний и самый ранний, конкурентное потребление. Вкупе с важными для решения задачи обновляемого поиска возможностями Kafka рассчитан на работу с большим количеством часто публикуемых сообщений, что должно быть более чем достаточно для большинства корпоративных информационных систем, включая высоконагруженные сервисы поиска для внешних клиентов. Также, как уже отмечалось, Kafka – это открытое программное обеспечение, что в наши дни является бесспорным плюсом, когда видна тенденция переключения фокуса разработки программного обеспечения в сторону решений с открытым исходных кодом даже при реализации программного обеспечения только для внутреннего использования.

Конечно, важно понимать, что ценен не столько конкретный инструмент, такой как Kafka, а предлагаемый подход к построению систем, реализующих обновляемый информационный поиск. Необходимо учитывать такие нюансы, как идемпотентная обработка, версионирование, непротиворечивость результатов, а также возможность поддержки и развития решения. Лишь аккумулируя опыт из смежных сфер разработки программного обеспечения, удается создать поистине полезное и технически актуальное решение.

Литература/References

1. Loginovsky O.V., Shestakov A.L., Shinkarev A.A. Supercomputing Technologies as Drive for Development of Enterprise Information Systems and Digital Economy. *Supercomputing Frontiers and Innovations*, 2020, vol. 7, no. 1, pp. 55–70. DOI: 10.14529/jsfi200103

2. *Message Bus*. Available at: https://www.enterpriseintegrationpatterns.com/patterns/messaging/MessageBus.html (accessed 31.12.2020).

3. Kalske M., Mäkitalo N., Mikkonen T. Challenges When Moving from Monolith to Microservice Architecture. *Garrigós I., Wimmer M. (eds.) Current Trends in Web Engineering. ICWE 2017. Lecture Notes in Computer Science, vol 10544*. Springer, Cham., 2018. DOI: 10.1007/978-3-319-74433-9_3

4. Namiot D., Sneps-Sneppe M. On micro-services architecture. *International Journal of Open Information Technologies*, 2014, vol. 2, no. 9.

5. Viggiato M., Terra R., Rocha H., Valente M., Figueiredo E. *Microservices in practice: A survey study*, 2018.

6. *Microservices*. Available at: https://martinfowler.com/articles/microservices.html (accessed 31.12.2020).

7. *Message broker*. Available at: https://en.wikipedia.org/wiki/Message_broker (accessed 31.12.2020).

8. *Messaging that just works – RabbitMQ*. Available at: https://www.rabbitmq.com (accessed 31.12.2020).

9. *Apache Kafka*. Available at: https://kafka.apache.org/ (accessed 31.12.2020).

10. *RabbitMQ vs. Kafka*. Available at: https://medium.com/better-programming/rabbitmq-vs-kafka-1ef22a041793 (accessed 31.12.2020).

11. *Kafka Consumer*. Available at: https://docs.confluent.io/platform/current/clients/consumer.html (accessed 31.12.2020).

12. *SQL and NOSQL: Which is Better*. Available at: http://www.jetir.org/papers/JETIR1508005.pdf (accessed 31.12.2020).

13. *The Future is Polyglot Persistence*. Available at: https://martinfowler.com/articles/nosql-intro-original.pdf (accessed 31.12.2020).

14. *Merge (SQL)*. Available at: https://en.wikipedia.org/wiki/Merge_(SQL) (accessed 31.12.2020).

15. *ACID*. Available at: https://en.wikipedia.org/wiki/ACID (accessed 31.12.2020).

Шинкарев Александр Андреевич, канд. техн. наук, инженер-программист, ООО «Софтмаст-ИТ»; sania.kill@mail.ru.

Поступила в редакцию 31 декабря 2020 г.

DOI: 10.14529/ctcr210101

ON ONE APPROACH TO IMPLEMENTATION OF INFORMATION INFRASTRUCTURE FOR RENEWABLE INFORMATION SEARCH

***A.A. Shinkarev**, sania.kill@mail.ru*

LLC "Softmast-IT", Chelyabinsk, Russian Federation

Major tasks in the field of information technology that business faces today one way or another relate to data processing and search for new information in it. The methods in use include statistical methods, machine learning models, and simpler methods and models. However, all solutions aimed at information search require an information infrastructure that would meet the basic requirements

for systems of this kind. **The purpose of the study** was to describe the main functional and technical requirements to modern systems implementing renewable information search. The author meant to form fundamental architectural proposals for the design of the system as a whole and its key parts, as well as to identify the main components of the information system and approaches to solving key problems for solutions based on the message exchange mechanism. **Materials and methods.** The paper considers the modern statement of the problem of creating enterprise information systems for renewable information search. The author compares message brokers that are viewed as the most interesting in the framework of this statement. **Results.** The article substantiates the relevance of the problem of creating information systems for renewable information search and formulates the task to create a system of such kind. The author makes an overview of the architecture design of the solution at a high level of abstraction. The modular composition of the information system under consideration is discussed. Kafka is chosen and substantiated as the most suitable message broker. The nuances of technical solutions to eliminate duplicate records and filter the latest information search results are analyzed.

Keywords: enterprise information systems, renewable information search, message broker, data analysis, Kafka.

Received 31 December 2020

ОБРАЗЕЦ ЦИТИРОВАНИЯ

Шинкарев, А.А. Об одном подходе к реализации информационной инфраструктуры обновляемого информационного поиска / А.А. Шинкарев // Вестник ЮУрГУ. Серия «Компьютерные технологии, управление, радиоэлектроника». – 2021. – Т. 21, № 1. – С. 5–11. DOI: 10.14529/ctcr210101

FOR CITATION

Shinkarev A.A. On One Approach to Implementation of Information Infrastructure for Renewable Information Search. *Bulletin of the South Ural State University. Ser. Computer Technologies, Automatic Control, Radio Electronics*, 2021, vol. 21, no. 1, pp. 5–11. (in Russ.) DOI: 10.14529/ctcr210101