Ерш В.С. Идентификация и анализ устойчивости кластеров социотехнических систем на основе нечеткого когнитивного подхода // Международный журнал информационных технологий и энергоэффективности. – 2020. –
Т. 5 № 3(17) с. 32–42

Международный журнал информационных технологий и энергоэффективности

Сайт журнала:
http://www.openaccessscience.ru/index.php/ijcse/

УДК 004.82

ОБЗОР СРЕДСТВ ДЛЯ АСИНХРОННОГО ВЗАИМОДЕЙСТВИЯ СЛОЖНЫХ ПРОГРАММНЫХ СИСТЕМ

Ерш В.С.
Филиал ФГБОУ ВО «Национальный исследовательский университет «МЭИ» в г. Смоленске, Россия, (214013, г. Смоленск, Энергетический проезд, 1), e-mail: victory.s.e@mail.ru

Дано описание обмена сообщениями, видов сообщений и моделей, которые реализуются системой обмена сообщениями. Проведен сравнительный анализ брокеров сообщений, среди которых Apache ActiveMQ, Apache Kafka, RabbitMQ, IBM MQ, NATS, ZeroMQ. Результаты анализа сведены в сравнительную таблицу. Уделено внимание расширенным возможностям брокеров сообщений по потоковой обработке данных, осуществляемой с помощью библиотеки Apache Kafka Streams и расширенного протокола сообщений NATS Streaming.

Ключевые слова: Обмен сообщениями, брокер сообщений, потоковая обработка данных.

REVIEW OF TOOLS FOR ASYNCHRONOUS INTERACTION OF COMPLEX SOFTWARE SYSTEMS

Ersh V.S.
Smolensk Branch of the National Research University "Moscow Power Engineering Institute", Smolensk, Russia (214013, Smolensk, Energetichesky proezd, 1), e-mail:[2]victory.s.e@mail.ru

The description of messaging, types of messages and models that are implemented by the messaging system is given. A comparative analysis of message brokers, including Apache ActiveMQ, Apache Kafka, RabbitMQ, IBM MQ, NATS, ZeroMQ, was carried out. The analysis results are summarized in a comparative table. Attention is paid to the advanced capabilities of message brokers for data streaming. It is done by the Apache Kafka Streams library and the extended message protocol NATS Streaming.

Keywords: Message passing, message broker, data streaming.

Обмен сообщениями – это технология высокоскоростного асинхронного взаимодействия между программами с гарантией доставки информации [1]. Программы взаимодействуют между собой, обмениваясь пакетами данных, называемыми сообщениями. Канал, или очередь, – это логический маршрут, объединяющий программы и использующийся для транспортировки сообщений. Канал напоминает массив сообщений, доступный для одновременного использования многими приложениями.

Отправитель – это приложение, которое отправляет сообщение путем его размещения в канале. Получатель – это приложение, которое получает сообщение путем его считывания из канала, а затем удаляет его.

Сообщение представляет собой некоторую структуру данных − строку, байтовый массив, запись или объект [2]. Оно может быть интерпретировано непосредственно как содержащиеся в нем данные, как команда, которую необходимо выполнить получателю, или как описание события, произошедшего на стороне отправителя. Сообщение состоит из двух частей – заголовка и тела. Заголовок сообщения содержит метаданные, которые используются системой обмена сообщениями и игнорируются получателем сообщения. Это могут быть данные об отправителе, или информация, куда следует передать сообщение. Тело сообщения содержит полезную информацию, которая, как правило, игнорируется системой обмена сообщениями.

В распределённой компьютерной среде существует большое количество моделей для обмена сообщениями [3], которые различается по способу передачи данных. Одной из них является асинхронная модель, где система посылает сообщение и продолжает обработку. Пользователям асинхронной системы сообщений не нужно ждать ответа от получателя, так как они могут положиться на инфраструктуру работы с сообщениями, которая обеспечивает доставку.

Этой связующей инфраструктурой является *система обмена сообщениями*. Она имеет много общего с системой баз данных. Основная задача системы баз данных − обеспечить надежное хранение информации, а основная задача системы обмена сообщениями − гарантировать доставку сообщений от отправителя получателю.

Необходимость наличия системы обмена сообщениями обусловливается ненадежностью сетей передачи данных. Сбой в компьютерной сети – одна из самых распространенных причин неудавшейся доставки сообщения от отправителя к получателю. Система обмена сообщениями позволяет гарантировать доставку информации за счет повторной отправки сообщения (до тех пор, пока оно не будет принято получателем). В идеальных условиях сообщение доставляется с первого раза, однако, к сожалению, так бывает далеко не всегда.

Рассмотрим несколько моделей [3], которые реализуются системой обмена сообщениями.

Модель «Точка-точка» гарантирует, что каждое отдельное сообщение будет потреблено только одним получателем. У канала может быть много получателей, одновременно обрабатывающих сообщения, но только один из них сможет успешно принять конкретное сообщение.

В классических системах обмена сообщениями данная модель обычно реализуется через очереди [4]. Очередь действует, как буфер FIFO, на который может подписаться один или несколько получателей. Каждое сообщение доставляется только одному из подписанных получателей. Если несколько получателей попытаются одновременно потребить некоторое сообщение, очередь позаботится о том, чтобы эта операция удалась только одному из них.

Обмен сообщениями типа *«Точка-точка»* используется, когда вариант использования требует однократного действия с сообщением (один получатель).

Модель «Публикация-подписка» используется для реализации широковещательной рассылки. Имеется один входной канал, который разбивается на несколько выходных каналов, по одному на каждого подписчика. Когда оповещение о событии публикуется в канале, оно распределяется по всем подписанным пользователям.

Обмен сообщениями типа «Публикация-подписка» обычно используется, когда сообщения носят информационный характер, и потеря одного сообщения не критична. Например, канал может передавать показания температуры от группы датчиков один раз в секунду [5]. Ничего страшного не произойдет, если система, которая получает эти данные, пропустит сообщение, потому что новое сообщение поступит в ближайшее время.

Существует несколько видов сообщений [6]:

- Сообщение с командой, отправляя которое приложение вызывает функциональность другого удаленного приложения. Сообщения с командой обычно отправляются по каналу «точка-точка», чтобы каждая команда была потреблена и выполнена только один раз.
- Сообщение с данными документа просто передает данные от одного приложения другому. Получатель сам решает, что с ними делать. В качестве данных может выступать элементарная единица данных, объект или структура данных, которую можно разложить на несколько небольших частей [7].
- Сообщения о событиях, которое применяется, когда несколько приложений хотели бы использовать оповещения о событиях, чтобы согласовывать свои действия. Когда у субъекта появляется событие, о котором следует оповестить остальных, он создает объект события, упаковывает его в сообщение и размещает в канале как сообщение о событии. Наблюдатель получает сообщение о событии, извлекает событие и обрабатывает его.

Рассмотрим современные брокеры сообщений и их возможности.

1. Apache ActiveMQ

Apache ActiveMQ – система обмена сообщениями, которая была выпущена в 2004 году. Она имеет открытый исходный код, который реализует Java Message Service (JMS). ActiveMQ, кроме Java, можно также использовать из таких языков программирования, как .NET, C, Delphi Perl, Python, PHP и Ruby [7]. ActiveMQ поддерживает большое количество транспортных протоколов, среди которых STOMP, REST, AMQP, MQT, OpenWire, WebSockets [7].

Данный брокер обеспечивает функции кластеризации, хранения сообщений, кэширования, поддержку возможности использования баз данных. Он предоставляет поддержку взаимодействия с более чем одним клиентом или сервером. Связь управляется с помощью таких функций, как кластеризация компьютеров и возможность использовать любую базу данных в качестве поставщика сохраняемости, помимо виртуальной памяти, кеша и сохраняемости журнала. ActiveMQ использует несколько режимов для обеспечения высокой доступности, среди них механизмы блокировки на уровне строк файловой системы и базы данных, совместное использование хранилища сохраняемости через общую файловую систему.

В ActiveMQ ведется запись в журналы на диске. Это такая структура данных на диске, в которую можно добавлять данные, но нельзя изменять или удалять, и состоящая из нескольких файлов. Такой механизм является эффективным для хранения и извлечения сообщений, так как доступ к диску для обеих операций является последовательным. И на жестких дисках, и на SSD накопителях последовательный доступ оказывается быстрее произвольного.

ActiveMQ поставляется с рядом подключаемых стратегий сохранения сообщений. Они идут в форме адаптеров персистентности, которые можно рассматривать, как движки сохранения сообщений. Когда брокер получает персистентные сообщения, они сначала записываются на диск в журнал. Журнал содержит лог всех входящих сообщений, а также сведения о тех сообщениях, которые были подтверждены, как прочитанные получателем. Входящие сообщения сериализуются брокером в независимое от протокола представление объекта, а затем маршализируются в двоичную форму, которая записывается в конец журнала. Работа брокера – это автоматический процесс, не требующий расширенного администрирования и мониторинга.

2. Apache Kafka

Apache Kafka – брокер сообщений, созданный корпорацией LikedIn как распределенная программа [8]. Он отличается высокой скоростью работы, масштабируемостью, отказоустойчивостью, возможностью передавать данные пакетами и способностью секционировать.

Работа брокера осуществляется следующим образом: приложения-отправители посылают сообщения на узел Kafka, затем указанные сообщения обрабатываются приложениями-потребителями. Указанные сообщения сохраняются, а потребители подписываются для получения новых сообщений. Kafka гарантирует, что сообщения будут упорядочены в той последовательности, в которой они поступали на узел. Kafka не отслеживает, прочитал ли получатель сообщения и поэтому не удаляет их после прочтения. Вместо этого сообщения хранятся на узле их в течение некоторого времени и по его истечении удаляются. Получатели сами опрашивают Kafka, не появилось ли у него новых сообщений, и указывают, какие записи им нужно прочесть.

Kafka является распределенной системой [9]. Хранение и пересылка сообщений происходит параллельно на разных серверах системы, что гарантирует высокую надежность. Если несколько серверов перестанут работать, сообщения все равно будут пересылаться и обрабатываться дальше.

Система легко масштабируется горизонтально, что дает возможность увеличить производительность путем добавления новых серверов. Максимальное количество машин, которых можно добавить в систему, не ограничено.

Записи в Apache Kafka хранятся в виде журнала транзакций [8]. Он представляет собой очередь сообщений, в которую можно только добавлять записи, удалять или модифицировать их нельзя. Такой подход обеспечивает упорядоченность записей.

3. RabbitMQ

RabbitMQ также, как и Kafka, является распределенным и горизонтально масштабируемым брокером сообщений. Он создан на основе системы Open Telecom Platform, а в качестве движка базы данных для хранения сообщений используется Mnesia [10]. Брокер написан на языке программирования Erlang, не широко используемом языке программирования, но хорошо приспособленном для таких задач. RabbitMQ является

отличным решением для построения сервис-ориентированной архитектуры и распределения отложенных ресурсоемких задач.

Данная система обмена сообщениями выступает в качестве сервиса-посредника и имеет множество функций: поддержку шифрования, разграничение права доступа, сохранение сообщений на диск, работу в кластерах и дублирование сервисов. Для построения кластерных решений поддерживается горизонтальное масштабирование.

RabbitMQ позволяет взаимодействовать различным программам при помощи протокола AMQP. Брокер включает в себя сервер, библиотеки поддержки протоколов HTTP, XMPP и STOMP, клиентские библиотек для Java и .NET Framework. Имеется реализация клиентов для большинства популярных языков программирования, среди которых Perl, Python, Ruby, PHP [10].

Так как в RabbitMQ каждый получатель запрашивает/выгружает разное количество сообщений, то распределение работы может стать неравномерным, что повлечет задержки и потерю порядка сообщений во время обработки. Для предупреждения этого и балансировки нагрузки каждый получатель настраивает предел предварительной выборки– ограничение на количество скопившихся неподтвержденных сообщений.

RabbitMQ включает четыре способа маршрутизации в отличие от Apache Kafka, реализующего запись на диск без маршрутизации. [11] Это позволяет использовать мощный и гибкий набор шаблонов обменов сообщениями. RabbitMQ позволяет пересылать сообщения при помощи маршрутизации с применением прямого и разветвляемого обменов. Также брокер дает возможность выборочной маршрутизации с тематическим обменом, основываясь на соответствии символьной подстановке в предлагаемом ключе маршрутизации, а обмен заголовками представляет некий альтернативный подход к маршрутизации сообщений с применением самого сообщения.

4. IBM MQ

IBM MQ – это сервер управления очередями сообщений, который был разработан компанией IBM в 1993 году под названием MQSeries. Позже он был переименован в IBM MQ [12]. Данный брокер доступен на большом количестве платформ, таких как UNIX, Linux и Microsoft Windows.

Архитектура IBM MQ не зависит от времени. Сообщения отправляются, независимо от того, работает ли программа-получатель в данный момент времени. Если она не работает, сообщение попадает в очередь и будет доставлено, когда получатель станет доступным. Порядок сообщений при этом сохранится, также сообщениям можно задать уровень приоритета.

С помощью IBM MQ можно отправлять различные сообщения. Например, это могут быть файлы большого размера либо сообщения с командой, которая запускает приложения. Последние используются для построения архитектуры системы управляемой сообщениями. Также возможно преобразование данных для различных протоколов и архитектур. Можно изменить кодировку или порядок разрядов в байте.

В IBM MQ хорошо продумана архитектура маршрутизации. Она позволяет доставлять сообщения по альтернативному пути через сеть диспетчеров. IBM MQ может быть реализован

в виде кластера, в котором обработка сообщений выполняется несколькими экземплярами для увеличения производительности.

Основной компонент IBM MQ – диспетчер очереди, который отвечает за действия, непосредственно не связанные с перемещением данных: хранение файлов, временную привязку, запуск приложений и другие [13]. Диспетчеры открыты для взаимодействия через прямое программное соединение для программ, выполняющихся с диспетчером очереди на одном компьютере, и через сетевое соединение. Сетевое соединение позволяет повысить гибкость системы, а прямое, в свою очередь, улучшает быстродействие.

Для взаимодействия диспетчеров между собой организуются специальные каналы. Один канал используется для данных в одном направлении, второй канал – для обратных данных. В сети TCP/IP каждому каналу назначается отдельный порт.

5. NATS

NATS – это относительно новый брокер сообщений с открытым исходным кодом. Изначально он был написан на языке программирования Ruby, а позже перенесен на Go. Клиенты NATS доступны на многих наиболее широко используемых языках, среди которых Go, C и C #, Java, PHP, Python и Rust [14].

Основными принципами разработки NATS являются производительность, масштабируемость и простота использования. Чтобы подключить новый узел, процессу NATS достаточно указать адрес любого узла кластера, после чего он скачивает всю топологию и определяет другие узлы.

Производительность обеспечивается за счет того, что все сообщения в кластере доставляются напрямую от отправителя получателю. Промежуточные шаги отсутствуют, что гарантирует минимальную задержку. Но из-за того, что сообщения не записываются на диск, сервисы-получатели должны аккуратно завершать свою работу. Сначала они отписываются от новых сообщений, после чего обрабатывают те, которые получили раньше и только после этого останавливают процесс.

Сообщения в NATS группируются по темам. Каждый узел кластера знает подписчиков других узлов и на какие темы они подписаны. Кроме того, клиенты NATS также знают топологию кластера и способны самостоятельно переподключаться в случае потери связи со своим узлом [15].

6. ZeroMQ

ZeroMQ – это библиотека обмена сообщениями, которая позволяет организовать быстрый асинхронный обмен сообщениями между приложениями. Она работает на различных архитектурах и поддерживается большинством языков программирования. Изначально она позиционировалась как интерфейс для обмена сообщениями, а затем – как коммуникационный протокол, основанный на TCP/IP [16].

ZeroMQ отличается от рассмотренных выше систем тем, что не является полноценной системой. Это простой программный интерфейс, который дает возможность создать собственное связующее программное обеспечение любой сложности.

Преимуществом ZeroMQ является высокая производительность. Она достигается за счёт того, что в данной системе отсутствуют издержки протокола AMQP. Зато присутствует

широковещательный протокол с гарантированной доставкой, а также набор вызовов для многопотоковой рассылки сообщений нескольким адресатам. ZeroMQ использует собственную разработку, заключающуюся в агрегированной отправки нескольких сообщений в одном TCP-пакете. Это позволяет уменьшить количество системных вызовов.

По сравнению с сокетами ZeroMQ обладает большей простотой использования. Для того, чтобы передать сообщение в ZeroMQ достаточно инициировать отправку сообщения, а дробление и отправка делается в отдельном потоке, асинхронно с выполнением пользовательского кода. Такая особенность удобна для реализации механизмов событийной обработки. В ZeroMQ отсутствует типизации сообщений, они не интерпретируются интерфейсом и являются массивом двоичных данных. Таким образом, через ZeroMQ можно передавать что угодно, например, сообщения JSON или двоичные форматированные данные типа BSON, Protocol Buffers или Thrift, не испытывая при этом неудобств.

Несмотря на то, что ZeroMQ является низкоуровневым интерфейсом, он обеспечивает хорошую масштабируемость. В ZeroMQ реализована децентрализованная схема обмена сообщениями. Сокет ZeroMQ равномерно распределяет нагрузку по сети за счет возможности быть подключенным к нескольким получателям. Одновременно с этим существует возможность получения сообщения от множества отправителей.

Таким образом, в ZeroMQ успешно реализован компромисс между эффективностью и функциональностью, позволяющий организовать быстрый асинхронный обмен сообщениями между высоконагруженными приложениями.

С учетом вышеизложенной информации о брокерах, было проведено сравнение, результаты которого приведены в таблице 1.

В плане масштабируемости рассмотренные брокеры имеют равнозначные возможности. Все они также поддерживают популярные языки программирования и протоколы. Все брокеры, кроме IBM MQ, являются программным обеспечением с открытым исходным кодом. Они хранят сообщение на дисках. Apache Active MQ, RabbitMQ и NATS дополнительно к этому поддерживают хранение в базах данных.

Что касается *потоковой обработки данных*, то по результатам сравнения выделяются два брокера: Apache Kafka и NATS.

Обработка большого массива данных в реальном времени предоставляет широкие возможности для использования в современных системах. Apache Kafka Streams и NATS Streaming позволяют обрабатывать информационные потоки в режиме реального времени.

Библиотека Apache Kafka Streams позволяет распределенным приложениям в режиме реального времени обрабатывать данные, хранящиеся в Kafka, реализуя стандартные классы этой платформы потоковой обработки без развертывания отдельных кластеров [17]. Благодаря этому инструменту можно организовать потоковую обработку данных прямо внутри кластера, без привлечения дополнительных технологий.

Kafka Streams API полностью скрывает сложность обслуживания отправителей и потребителей сообщений Kafka, позволяя сосредоточиться на логике обработки информационных потоков. Она способна выполнять обработку данных с сохранением и/или без сохранения состояния в режиме реального времени [18].

Таблица 1 – Сравнительная таблица систем обмена сообщениями

№ п/п	Название	Apache ActiveMQ	Apache Kafka	RabbitMQ	IBM MQ	NATS	ZeroMQ
1	2	3	4	5	6	7	8
1	Язык программирования, на котором разработана система	Java	Java	Erlang	–	Ruby, Go	C++
2	Поддерживаемые языки	C, C#, Haxe, Node.js, Perl, Racket, Python, Ruby on Rails	C, C++, Go, Java, .NET, Python, Scala	Java, Ruby, Python, PHP, Perl Rust, Go, JavaScript, C, C++	C, C#, C++, Java, Visual Basic, Assembler, Node.js, Ruby, Python	Go, C, C#, Java, PHP, Python, Rust	C, C++, C#, Java, Python, Ruby, Go, NodeJS, Perl, Rust
3	Открытый исходный код	Да	Да	Да	Нет	Да	Да
4	Хранение сообщений	БД / Диск	Диск	БД / Диск	Диск	БД / Диск	Диск
5	Масштабируемость	Да	Да	Да	Да	Да	Да
6	Поддерживаемые протоколы	OpenWire, Stomp, WSIF, WS, AUTO, AMQP, MQTT, REST	Binary over TCP	HTTP, MQTP, STOMP, AMQP	AMQP, MQI, REST, JMS	Google Protocol Buffer	IPC, TCP, UDP, TIPC, WebSocket
7	Возможность потоковой обработки	Нет	Да	Нет	Нет	Да	Нет

Расширенный протокол сообщений NATS Streaming также предоставляет пользователям возможность потоковой обработки больших объемов данных. Он реализует собственный расширенный формат сообщений с использованием Google Buffer Protocol [19]. Эти сообщения передаются как полезные данные двоичного сообщения через базовую платформу NATS.

NATS Streaming предлагает настраиваемое сохранение сообщений: в памяти или в базе данных. Подсистема хранения использует общедоступный интерфейс, который позволяет пользователям разрабатывать свои собственные реализации. Протокол гарантирует доставку сообщений за счет того, что сообщения сохраняются в памяти и повторно доставляются по мере необходимости.

Apache Kafka Streams и NATS Streaming основаны на следующих важных концепциях потоковой обработки [20]:

- управление состоянием приложения;
- быстрые и эффективные агрегации и объединения;
- различие между временем события и временем его обработки;
- непрерывная обработка данных, поступающих с опозданием и помехами.

Описанные выше сервисы применяются для управления запросами о состоянии приложений в режиме реального времени и безкластерной разработки распределенных сервисов с последующим масштабированием. Они широко используется на практике в различных IT-проектах государственных и частных компаний.

Список литературы

1. Хоп Г., Вульф Г. Шаблоны интеграции корпоративных приложений. М.: ООО «И.Д. Вильямс», 2007. 672 с.
2. Беседина К.В. Современные подходы к интеграции корпоративных приложений / К. В. Беседина // Academy. 2017. № 5 (20). С. 41-43.
3. Линев Ф. А. Обзор систем обмена сообщениями / Ф. А. Линев // Молодой ученый. 2017. № 19 (153). С. 29-32.
4. Фаулер М. Архитектура корпоративных программных приложений. М.: ООО «И.Д. Вильямс», 2016. 544 с.
5. Оринштейн Д. Интеграция корпоративных приложений. М.: Microsoft Press, 2002. 454 с.
6. Пролетарский А.В., Березкин Д.В., Гапанюк Ю.Е., Козлов И.А., Попов А.Ю., Самарев Р.С., Терехов В.И. Методы ситуационного анализа и графической визуализации потоков больших данных Вестник МГТУ им. Н.Э. Баумана. Сер. Приборостроение. 2018. № 2. С. 98-123.
7. Jakub Korab. Understanding Message Brokers. Learn the Mechanics of Messaging though ActiveMQ and Kafka. O'Reilly, 2017. 132 p.
8. Apache Kafka Documentation [Электронный ресурс] URL: https://kafka.apache.org/documentation/gettingStarted (дата обращения 15.09.20).
9. Estrada R. Apache Kafka 1.0 Cookbook: Over 100 practical recipes on using distributed enterprise messaging to handle real-time data Packt Publishing, 2017. 250 с.
10. Alvaro Videla, Jason J. W. Williams RabbitMQ in Action: Distributed Messaging for Everyone 2012. 312 с.
11. Чадов А. Ю.; Михальченко К. Э. Сравнительный анализ AMQP брокеров сообщений для использования в качестве элемента децентрализованной системы разграничения доступа Комплексная защита информации: материалы XXIV научно-практической конференции. Витебск. 21–23 мая 2019 г.: УО ВГТУ. Витебск, 2019. С. 261-267.

12. Introduction to IBM MQ [Электронный ресурс] URL: https://www.ibm.com/support/knowledgecenter/en/SSFKSJ_8.0.0/com.ibm.mq.pro.doc/q001020_.htm (дата обращения 15.09.20).
13. Марц Н., Уоррен Дж. Большие данные. Принципы и практика построения масштабируемых систем обработки данных в реальном времени. М.: Вильямс, 2016. 368 с.
14. NATS Documentation [Электронный ресурс] URL: https://docs.nats.io/ (дата обращения 15.09.20).
15. Andrade H.C.M., Gedik B., Turaga D.S. Fundamentals of stream processing: application design, systems, and analytics. Cambridge University Press, 2014. 558 p.
16. ZeroMQ. An introduction [Электронный ресурс] URL: http://wiki.zeromq.org/blog:introduction (дата обращения 15.09.20).
17. Andrade, Henrique C.M., Bugra G., Deepak S. Turaga. Fundamentals of Stream Processing: Application Design, Systems, and Analytics. NY: Cambridge University Press, 2014. 529 p.
18. Нархид Н., Шапира Г., Палино Т. Apache Kafka. Потоковая обработка и анализ данных. СПб.: Питер, 2019. 320 с.
19. William P. Bejeck Jr. Kafka Streams in Action: Real-time apps and microservices with the Kafka Streams API Manning Publications, 2018. 280 p.
20. Самарев Р.С. Обзор состояния области потоковой обработки данных / Р. С. Самарев // Труды Института системного программирования РАН. 2017. № 1 (29). С. 231-260.

References

1. Hop G., Wolf G. Patterns of Enterprise Application Integration. M.: J.D. Williams LLC, 2007. 672 p.
2. Besedina K.V. Modern Approaches to Integrating Corporate Applications / K.V. Besedina // Academy. 2017. No 5 (20). pp. 41-43.
3. Linev F.A. Review of Messaging Systems / F.A. Lynev // Young Scientist. 2017. No 19 (153). pp. 29-32.
4. Fowler M. Architecture of Corporate Software Applications. - M.: J.D. Williams LLC, 2016. 544 p.
5. Orinstein D. Corporate Application Integration. M.: Microsoft Press, 2002. 454 p.
6. Proletarian A.V., Berezkin D.V., Gapanyuk Y.E., Kozlov I.A., Popov A.Y., Sama-rev R.S., Terekhov V.I. Methods of situational analysis and graphic visualization of big data flows - Herald of the Moscow State University by N.E. Bauman. Sir. Instrumentation. 2018. No 2. pp. 98-123.
7. Jakub Korab. Understanding Message Brokers. Learn the Mechanics of Messaging though ActiveMQ and Kafka. O'Reilly, 2017. 132 p.
8. Apache Kafka Documentation [E-resource] URL: https://kafka.apache.org/documentation/gettingStarted (Address date 15.09.20).
9. Estrada R. Apache Kafka 1.0 Cookbook: Over 100 practical recipes on using distributed enterprise messaging to handle real-time data Packt Publishing, 2017. 250 p.
10. Alvaro Videla, Jason J. W. Williams RabbitMQ in Action: Distributed Messaging for Everyone 2012. 312 p.
11. Chadov A. Yu.; Michalchenko K.E. Comparative analysis of AM'P brokers messages for use as an element of a decentralized access delineation system / Comprehensive information

protection: materials of the XXIV Scientific and Practical Conference. Vitebsk. May 21-23, 2019: VGTU. pp. 261-267.

12. Introduction to IBM MQ [E-resource] URL: https://www.ibm.com/support/knowledgecenter/en/SSFKSJ_8.0.0/com.ibm.mq.pro.doc/q001020_.htm (Address date15.09.20).
13. Martz N., Warren J. Big data. Principles and practices in building scalable real-time data processing systems. M.: Williams, 2016. 368 p.
14. NATS Documentation [Электронный ресурс] URL: https://docs.nats.io/ (Address date 15.09.20).
15. Andrade H.C.M., Gedik B., Turaga D.S. Fundamentals of stream processing: application design, systems, and analytics. Cambridge University Press, 2014. 558 p.
16. ZeroMQ. An introduction [E-resource] URL: http://wiki.zeromq.org/blog:introduction (Address date 15.09.20).
17. Andrade, Henrique C.M., Bugra G., Deepak S. Turaga. Fundamentals of Stream Processing: Application Design, Systems, and Analytics. – NY: Cambridge University Press, 2014. 529 p.
18. Narhid N, Shapira G, Palino T. Apache Kafka. Streaming and data analysis. St. Petersburg: Peter, 2019. 320 p.
19. William P. Bejeck Jr. Kafka Streams in Action: Real-time apps and microservices with the Kafka Streams API Manning Publications, 2018. 280 p.
20. Samarev R.S. Review of the State of Streaming Data / R.S. Samarev // Proceedings of the Institute of System Programming ras. 2017. No 1 (29). pp 231-260.