УДК 004.75


И.А. Притыченко, А.А. Суконщиков
Вологодский государственный университет

АРХИТЕКТУРА РАСПРЕДЕЛЕННОГО ПЛАНИРОВЩИКА ЗАДАНИЙ

В данной статье представлен аналитический обзор существующих вариантов планирования задач в вычислительных системах. Анализ показывает преимущества и недостатки каждого варианта. Предлагается собственная архитектура распределенного планировщика задач, основанная на распределенной системе обмена сообщениями Apache Kafka.

Планирование задач, система обмена сообщениями, Apache Kafka.

В современном мире в различных сферах деятельности присутствует множество ресурсоемких задач, требующих объемных вычислений. Для решения таких задач применяются вычислительные системы, состоящие из определенного количества узлов, связанных сетью передачи данных. Для отправки большой задачи на выполнение в распределенную вычислительную систему ее нужно декомпозировать, то есть разделить одну большую задачу на подзадачи, а затем весь список подзадач отправить на исполнение в определенные подсистемы. Из-за высокой сложности распределенных систем у них имеется множество точек отказа, поэтому если не предпринимать действий для обеспечения надежности системы отправки и выполнения подзадач, есть риск потери данных, а также некорректного исполнения задачи.

Еще один вид задач – это задача, распределенная во времени. В программном обеспечении очень накладно делать такие задачи синхронными (пользователь ожидает полного выполнения задачи, блокируя свой собственный поток исполнения), поэтому такие задачи делают асинхронными (пользователь регистрирует задачу и продолжает выполнять другие свои задачи, а потом получает уведомление о выполнении зарегистрированной задачи). Задачи, распределенные во времени, также имеют точки отказа, которые включают в себя риск неполного исполнения задачи или полной потери данных.

Рассмотрим виды задач, которые могут исполняться распределенными вычислительными системами, для исполнения которых нужен распределенный планировщик заданий.

Ресурсоемкие задачи. Это задачи, которые требуют больших и длительных вычислений, или для решения задачи требуется интеграция с несколькими подсистемами, что в свою очередь увеличивает длительность выполнения такой задачи.

Если использовать требования к программному обеспечению, описанные выше, то выполнение данной задачи без каких-либо доработок уже будет нарушать требования заказчика – требование к быстродействию системы. Для соответствия данному требованию необходимо перевести задачу из синхронного взаимодействия с пользователем в асинхронное. В рамках перехода на асинхронный формат взаимодействия с пользователем необходимо будет сделать систему регистрации заданий. То есть пользователь регистрирует задание, но не получает сразу результат выполнения, а получает идентификатор задания, по которому он позже сможет получить результат.

Задачи, выполнение которых нужно отложить во времени. Данный тип задач очень специфичен для программного обеспечения, но имеет место быть. В качестве примера можно привести такие задачи, как запланированное создание бэкапов, отслеживание статуса выполнения другой задачи и т.д.

На текущий момент реализовать планирование задач можно используя такие способы:

1. Использование планировщика задач из операционной системы.

Планировщик задач операционной системы – это программа (служба или демон), часто называемая сервисом операционной системы, которая запускает другие программы в зависимости от различных критериев, как, например:

- наступление определенного времени;
- переход операционной системы в определенное состояние (бездействие, спящий режим и т.д.);
- поступление запроса от администратора через пользовательский интерфейс или через инструменты удаленного администрирования.

Вычислительная система во время своей работы может обратиться к сервису планировщика задач операционной системы и зарегистрировать задание, указав необходимы критерии выполнения. Плюсами данного подхода к планированию задач являются:

- использование стандартного планировщика задач, что увеличивает переиспользование компонентов и уменьшает дублирование функций;
- более эффективная работа, так как системный планировщик работает в процессе операционной системы, а не конкретного приложения.

Из минусов такого подхода можно выделить:

- завязка программного обеспечения на конкретную операционную систему. Этот минус можно решить, используя общие интерфейсы для работы с планировщиком задач, но в общем случае будет привязка к операционной системе;

- в распределенных системах системный планировщик будет у каждого узла в отдельности, то есть синхронизироваться задания не будут.

Таким образом, системный планировщик рекомендуется использовать для таких вычислительных систем, где не требуется синхронизация заданий между узлами.

2. Использование возможностей технологий разработки.

Большинство языков программирования предоставляют «из коробки» методы для создания запланированной задачи. В случае, когда решений языка программирования недостаточно, то на помощь приходят фреймворки. Для примера возьмем язык программирования Java. В нем есть метод для периодического выполнения задач – ScheduledExecutorService. Благодаря этому сервису можно запускать периодически повторяемые задачи в отдельном потоке. Также, например со стороны фреймворка Spring, предоставляется возможность планировать задачи с помощью аннотации @Scheduled.

Плюсы использования данного способа при реализации планирования задач:

- удобство использования внутри приложения. В большинстве языков и фреймворков создание запланированной задачи занимает не более нескольких строк кода.

Минусы такого способа:

- в распределенных системах такой планировщик будет у каждого узла в отдельности, то есть синхронизироваться задания не будут.

3. Реализация собственного планировщика задач.

Вариантов реализации собственной системы планирования задач множество, но в данном контексте мы рассмотрим самый распространенный вариант создания планировщика заданий, который будет поддерживать распределенные системы – это планировщик заданий на основе polling-strategy с сохранением заданий в базу данных.

В основе такого планировщика лежит сервис, который принимает запрос на создание запланированного задания, создает его уникальный номер, сериализует параметры, вычисляет время его выполнения и сохраняет в базу данных. После этого каждый узел, подключенный к этой базе данных через определенный интервал времени, делает запрос в базу данных и получает задания, у которых пришло время выполнения и забирает эти здания на обработку.

Плюсами данного способа являются:

- возможность работать в распределенных вычислительных системах.

К минусам метода можно отнести:

- необходимость разворачивать и настраивать базу данных;

- при работе нескольких узлов необходимо корректно работать с транзакциями базе данных;

- при отказе базы данных – весь планировщик заданий перестанет работать [2].

Таким образом, мы рассмотрели три способа создания планировщика заданий. Каждый из способов имеет свои плюсы и минусы и область применения. Так как целью исследования является распределенное планирование задач, поэтому был выбран вариант с разработкой архитектуры собственного планировщика задач.

Для хранения данных необходимо выбрать распределенную, отказоустойчивую систему хранения. Это может быть как база данных с поддержкой распределенного размещения, так и система обмена событиями. В архитектуре хранение данных о запланированных задачах было не в базе данных, а в распределенной системе обмена сообщениями. В качестве системы была выбрана система Apache Kafka, разработанная компанией LinkedIn, сейчас платформа развивается и поддерживается как открытый проект в рамках фонда Apache Software Foundation [1].

Кратко архитектуру Apache Kafka можно охарактеризовать следующим образом:

- распределенность – отдельные узлы системы располагаются на нескольких аппаратных платформах (кластерах). Это обеспечивает ей высокую отказоустойчивость;

- масштабируемость – систему можно наращивать за счет простого добавления новых узлов (брокеров сообщений).

Для начала разберемся с терминологией, которая применяется в описании Kafka:

- продюсер – приложение или процесс, генерирующий и посылающий данные;

- потребитель – приложение или процесс, который принимает сгенерированное продюсером сообщение;

- сообщение – пакет данных, необходимый для совершения какой-либо операции;

- брокер – узел передачи сообщения от процесса-продюсера приложению-потребителю;

- топик – виртуальное хранилище сообщений одинакового или похожего содержания, из которого приложение-потребитель извлекает необходимую ему информацию.

Верхнеуровнево работа с Kafka построена таким образом:

- приложение-продюсер создает сообщение и отправляет его на узел Kafka;

- брокер сохраняет сообщение в топике, на который подписаны приложения-потребители;

- потребитель при необходимости делает запрос в топик и получает из него нужные данные.

С инфраструктурной точки зрения (рис. 1) приложение размещается в кластере и запускается в нескольких экземплярах, где каждый экземпляр может как планировать задания, так и выполнять запланированные задачи. Благодаря отправке всех сведений о запланированном задании в кластер Kafka получаем единое распределенное хранилище заданий.

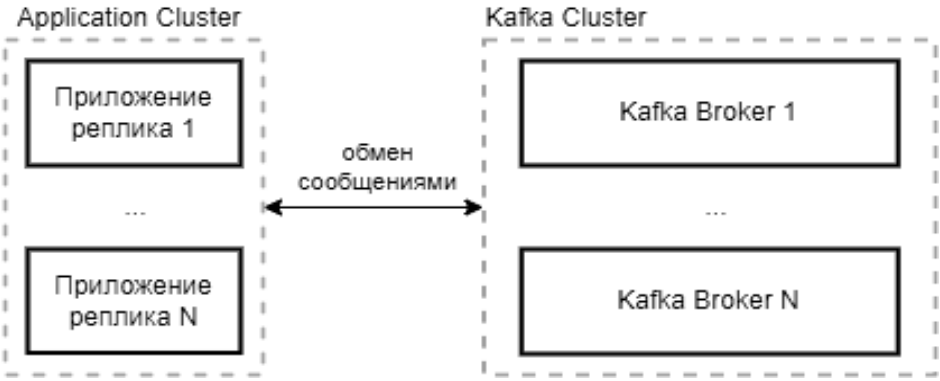

Рис. 1. Инфраструктурное представление архитектуры распределенного планировщика заданий

Рис. 2. Работа приложений с распределенным хранилищем сообщений

Каждая реплика приложения начинает слушать очередь сообщений с помощью Kafka Consumer (рис. 2), из которой будут поступать сообщения, содержащие информацию о запланированных заданиях. При поступлении задания на выполнение, приложение начинает транзакцию, в ходе которой производится выполнение бизнес-логики запланированного задания и после успешного выполнения завершает транзакцию с помощью специального сигнала «Acknowledge», который примет кластер Kafka и пометит сообщение с информацией о запланированном задании как прочитанное.

В обратном случае, то есть в случае планирования задания, приложение формирует сообщение с информацией о задании по заранее заданной схеме и отправляет его в кластер Kafka. При попадании сообщения в Kafka оно автоматически попадет в топик и будет доступно для получения одним из приложений кластера, которые слушают этот топик с помощью Kafka Consumer [3].

Теперь рассмотрим негативные сценарии, которые будут корректно обработаны данным планировщиком заданий:

1. Ошибка исполнения запланированного задания. В самой базовой реализации, транзакция не сможет завершиться корректно и будет получена ошибка. В свою очередь Kafka Cluster не получит специальный сигнал «Acknowledge» и передаст сообщение еще раз, это приведет к повторной попытке исполнения задания.

2. Отключение всей реплики приложения во время выполнения задания. В этом случае кластер Kafka через определенный интервал не получит успешную проверку готовности принимать сообщения от отключенной реплики приложения и удалит это приложение из consumer-группы, которая является балансировщиком сообщений между репликами. Затем будет произведено перераспределение сообщений, предназначен-ных для отключенного приложения между другими приложениями в кластере.

3. Самый негативный сценарий – падение всего Kafka Cluster. Вероятность возникновения данного события крайне мала, если инфраструктура георасределена и узлы Kafka независимы друг от друга. Но все же если будет отключен весь кластер Kafka, то планирование и выполнение задач будет остановлено.

Таким образом, в рамках этой статьи был проведен аналитический обзор существующих планировщиков заданий, была предложена собственная архитектура распределенного планировщика заданий, которая основана на распределенной системе обмена сообщениями Apache Kafka. В рамках аналитического обзора были выделены варианты реализации планировщика заданий в современной разработке, а также описаны их достоинства

и недостатки. В ходе разработки архитектуры было сделано небольшое теоретическое введение в Apache Kafka, рассмотрена схема взаимодействия на уровне инфраструктуры и на уровне приложения, а также негативные сценарии и реакция распределенного планировщика заданий на них.

Литература

1. Use Cases of Apache Kafka // Apache Kafka. – URL: kafka.apache.org/uses (дата обращения: 01.12.2022). – Text : Electronic.
2. Task Execution and Scheduling // Spring Docs. – URL: docs.spring.io/spring-framework/docs/3.1.x/spring-framework-reference/html/scheduling.html (дата обращения: 01.12.2022). – Text : Electronic.
3. Apache Kafka Architecture: A Complete Guide // Instaclustr. – URL: www.instaclustr.com/blog/apache-kafka-architecture (дата обращения: 01.12.2022). – Text : Electronic.

I.A. Pritychenko, A.A. Sukonshchikov
Vologda State University

ARCHITECTURE OF DISTRIBUTED TASK SCHEDULER

This article provides an analytical review of the existing options for task scheduling in computing systems. The analysis highlights the advantages and disadvantages of each option. The article proposes own distributed task scheduler architecture, which is based on the Apache Kafka distributed messaging system.

Task scheduling, messaging system, Apache Kafka.