Отличное и очень комплексное задание. Давайте детально его разберем для вашего кейса "Система рекомендаций для стримингового сервиса".

Часть 1: Оценка нагрузок

1. Определите предполагаемое количество одновременных пользователей

Здесь важно разделить общее количество пользователей сервиса и тех, кто активен одновременно.

*   **Общее количество пользователей (DAU — Daily Active Users):** Допустим, у сервиса 5 миллионов активных пользователей в день.
*   **Пиковая одновременная нагрузка:** В вечерние часы (с 18:00 до 23:00) активны до 20% от DAU. Это **1 000 000 пользователей** онлайн одновременно.
*   **Пользователи, генерирующие события для рекомендаций:** Не все просто смотрят. Допустим, 80% из них (800 000) активно потребляют контент (начинают, заканчивают просмотр, ставят лайки) в любой момент времени. Именно эти 800 000 пользователей являются "источниками" нагрузки на запись.

Вывод: Пиковое количество одновременных пользователей, генерирующих данные для системы рекомендаций — **~800 000**.

2. Оцените объем данных и их рост

Рассчитаем, сколько событий они генерируют.

*   **События на пользователя:** За один вечерний сеанс пользователь может посмотреть 3-5 единиц контента (фильм, серии). Для каждого просмотра генерируется минимум 2 события: "начало просмотра" и "конец просмотра". Плюс лайки/дизлайки. Возьмем в среднем **10 событий на пользователя в час** в пик.
*   **Пиковая скорость записи:** 800 000 пользователей * 10 событий/час = 8 000 000 событий в час. Это примерно **2 222 события в секунду (RPS — Requests Per Second)**.
*   **Объем данных в день:**
    *   В сутки: 8 млн событий/час * 4 пиковых часа + (оставшиеся 20 часов с низкой нагрузкой, допустим, 1/5 от пика) ~= 32 млн + 32 млн = **~64 миллиона событий в день**.
    *   Размер одного события: `user_id` (8 байт), `content_id` (8 байт), `event_type` (4 байта), `duration` (4 байта), `timestamp` (8 байт) + накладные расходы. Итого ~40 байт.
    *   Объем данных в день: 64 млн * 40 байт = **~2.56 ГБ сырых данных в день**.
*   **Рост в месяц/год:**
    *   В месяц: 64 млн * 30 дней = **~1.92 миллиарда событий** (~77 ГБ данных).
    *   В год: ~23 миллиарда событий (~920 ГБ данных).
    *   **Важно:** Это только сырые данные. Матрица взаимодействий и различные агрегированные данные для ML-моделей могут занимать сопоставимый или больший объем.

3. Определите типы и частоту операций

*   **Соотношение записей и чтений:** Система в первую очередь **write-heavy** (написоемкая). Каждое действие пользователя — это запись. Но чтений тоже очень много.
    *   **Запись:** Постоянный поток ~2-3 тыс. RPS в пике.
    *   **Чтение:** Происходит в двух сценариях:
        1. **Обучение ML-моделей:** Периодические (раз в несколько часов/день) очень тяжелые аналитические запросы, которые сканируют гигабайты данных для пересчета матрицы рекомендаций. Это пакетные (batch) чтения.
        2. **Выдача рекомендаций в реальном времени:** При заходе пользователя в каталог нужно быстро (за десятки миллисекунд) получить для него персонализированную подборку. Это точечные чтения по `user_id`, но требующие доступа к предварительно рассчитанным данным.
    *   **Примерное соотношение:** 90% записей, 10% чтений (по количеству операций, но по объему данных чтения могут быть сопоставимы).

*   **Наличие пиковых нагрузок:** Да, ярко выраженные пики в вечернее время и особенно в выходные дни. Нагрузка может в 5-10 раз превышать дневную.

Часть 2: Выбор СУБД и проектирование инфраструктуры

Выбор СУБД

Выбор: Гибридный подход (Polyglot Persistence)
Использование одной СУБД для всех задач неоптимально. Лучше использовать две, каждая для своей цели.

1. **Для хранения сырых событий и аналитики: PostgreSQL**
    *   **Обоснование:** Вы указали желание использовать реляционную СУБД. PostgreSQL — отличный выбор.
        *   **Надежность и стабильность:** Гарантирует, что ни одно событие не потеряется.
        *   **Простота записи:** Отлично справляется с постоянной потоковой записью миллионов строк.
        *   **Мощные аналитические возможности:** Window functions, эффективные JOIN, поддержка JSONB (если структура событий будет усложняться). Идеально подходит для сложных запросов при пересчете ML-моделей.
        *   **Масштабируемость:** Возможность использовать реплики для разгрузки аналитических запросов.

2. **Для хранения готовых рекомендаций и быстрого доступа: Redis**
    *   **Обоснование:** Ключ-значное хранилище в оперативной памяти.
        *   **Скорость чтения:** Микросекундные задержки. Это критично для выдачи рекомендаций в реальном времени.

*   **Структура данных:** Идеально подходит. Можно хранить для каждого `user_id` (ключ) список `content_id` (значение в виде List или Sorted Set), отсортированный по релевантности.
*   **Позволяет развязать системы:** ML-процесс рассчитывает рекомендации и загружает их в Redis, а веб-сервис просто забирает их оттуда, не нагружая основную базу.

Архитектура потока данных:
Пользователь -> Веб-сервис -> (Запись события) -> PostgreSQL -> (Раз в N часов ML-процесс) -> Аналитика в PostgreSQL -> (Обучение модели) -> (Запись результатов) -> Redis -> Веб-сервис -> (Чтение рекомендаций) -> Пользователь.

Формулировка нефункциональных требований

1. **Производительность:**
   *   **Запись событий:** 95% запросов на вставку в PostgreSQL должны выполняться < 10 мс.
   *   **Чтение рекомендаций:** 99% запросов к Redis на получение списка рекомендаций для пользователя должны выполняться < 5 мс.
   *   **Обучение модели:** Пакетный пересчет рекомендаций должен завершаться не более чем за 4 часа.

2. **Доступность:**
   *   **Цель:** 99.9% (порядка 8-9 часов простоя в год). Сервис рекомендаций не является абсолютно критическим для самого просмотра (стриминг может работать без него), но его недоступность напрямую влияет на вовлеченность и удержание пользователей.

Определение требований к комплектующим

Будем считать, что у нас отдельные серверы для PostgreSQL и Redis.

Для Сервера PostgreSQL (основная нагрузка — запись и аналитика):

*   **CPU:** **Много ядер для параллельных запросов.**
   *   *Обоснование:* Параллельные операции ввода-вывода при потоковой записи, а также возможность использования параллельных планов выполнения для тяжелых аналитических запросов при обучении моделей. Тактовая частота тоже важна, но количество ядер — приоритет.
*   **RAM:** **Большой объем для кэширования БД.**
   *   *Обоснование:* Чтобы максимально ускорить аналитические запросы, необходимо закэшировать в оперативной памяти "горячие" данные (последние несколько месяцев событий). Цель — чтобы Working Set (рабочий набор данных) помещался в RAM. Для 500 ГБ данных нужно не менее 128-256 ГБ RAM.
*   **Disk:** **Высокая скорость (IOPS) для операций ввода-вывода.**
   *   *Обоснование:* Постоянная запись потока событий и интенсивное чтение при аналитике требуют очень быстрого диска. **NVMe SSD** обязательны. Лучше использовать RAID 10 для отказоустойчивости и производительности.

Для Сервера Redis (основная нагрузка — быстрое чтение в памяти):

*   **CPU:** **Высокая тактовая частота для сложных вычислений (одного ядра).**
    *   *Обоснование:* Redis в основном однопоточен. Важна высокая производительность одного ядра для быстрой обработки каждого запроса. Много ядер можно использовать для работы нескольких инстансов Redis или для системных нужд.
*   **RAM:** **Большой объем для хранения всех данных.**
    *   *Обоснование:* Все данные Redis должны полностью помещаться в оперативной памяти. Рассчитаем: 5 млн пользователей * (1 ключ `user_id` + 100 рекомендованных `content_id` * 8 байт) ~= 4 ГБ. Плюс накладные расходы. **16-32 ГБ RAM** должно хватить с запасом. Тип памяти — чем быстрее, тем лучше.
*   **Disk:** **Скорость не критична, надежность — да.**
    *   *Обоснование:* Диск в Redis используется в основном для персистентности (периодического снапшота данных на случай сбоя). Высокие IOPS не требуются, подойдет стандартный SSD. Надежность обеспечивается репликацией на другой сервер, а не диском.