На втором графике показаны результаты сравнения быстрой сортировки: C++ (оранжевый) оказалась быстрее, разница составляет всего несколько миллисекунд (рис. 3).

На третьем графике разница во времени не так велика, однако операция заняла почти 12 секунд (1200 мс). Алгоритм сортировки вставками плохо масштабируется и показала очень низкую производительность как в Java, так и в C++. C++ (оранжевый) справилась быстрее, но всего на несколько миллисекунд. Результаты для самого маленького набора данных на этом графике не видны, но C++ выполнила операцию за 0,1 мс, а Java – за 0,22 мс (рис. 4).

На четвёртом графике показано, что C++ был быстрее Java при сохранении данных в файл. Java был быстрее во втором тесте (рис. 5).

На пятом графике показано среднее общее время, затраченное приложениями на выполнение всех 50 итераций, Java (синий) оказалась медленнее C++ во всех тестах кроме одного – с самым маленьким набором данных (1000 целых чисел). Результаты для самого маленького набора трудно разобрать на этом графике, но общее время выполнения составило: C++ – 2.68 мс, Java – 4,8 мс (рис. 6).

Заключение

Результаты показывают, что C++ быстрее при сортировке и обработке данных, особенно при использовании quicksort и сортировки вставками, а также при сохранении файлов. Однако загрузка данных в C++ оказалась медленнее из-за посимвольного чтения, что привело к худшему масштабированию. В Java загрузка данных была быстрее благодаря более эффективной работе с токенами в библиотеке Scanner. В целом, C++ показала лучшую производительность при обработке данных, но Java при загрузке. Масштабируемость сортировки вставками оказалась плохой для обоих языков, в то время как quicksort масштабировался лучше. В реальных приложениях, где загрузка данных происходит не так часто, C++ может быть предпочтительнее благодаря более высокой скорости обработки.

Список литературы

1. Prechelt L. Technical opinion: Comparing java vs. C/C++ efficiency differences to interpersonal differences // Communications of the ACM. 1999. № 42(10). P. 109-112. DOI: 10.1145/317665.317683.

2. Nikishkov G. P., Nikishkov Y. G., Savchenko V. V. Comparison of C and java performance in finite element computations // Computers and Structures. 2003. № 81(24). P. 2401-2408. DOI: 10.1016/S0045-7949(03)00301-8.

3. METANIT: материалы по таким направлениям, как язык C#, C/C++ и технологии на базе Java. URL: https://metanit.com/ (дата обращения: 15.12.2025).

4. Java currentTimeMillis function. (n.d.). Retrieved from. URL: https://docs.oracle.com/javase/7/docs/api/java/lang/System.html#currentTimeMillis() (дата обращения: 15.12.2025).

5. Шилдт Г. Java: руководство для начинающих. 7-е изд. М.: ООО «И.Д. Вильямс», 2020. 688 с.

6. Савченко В. В. Объектно-ориентированное программирование: учебное пособие. 2-е изд., перераб. и доп. М.: Академия, 2013. 288 с.

АРХИТЕКТУРА МАСШТАБИРУЕМОЙ ЛЕНТЫ НОВОСТЕЙ

Романенко В. В., Луцкий Е. А.

ФГБОУ ВО «Мелитопольский государственный университет», Мелитополь, e-mail: romanenkoviktor080@yandex.ru

Научный руководитель: Луцкий Е. А.

Введение

В эпоху цифровых технологий социальные сети и информационные платформы накапливают колоссальные объёмы данных. По оценкам аналитиков, ежедневно в глобальной сети создаётся более 2,5 квинтиллиона байт информации, значительная часть которой поступает через приложения социальных сетей. Центральным компонентом таких платформ является лента новостей – динамически обновляемый набор контента, упорядоченный в соответствии с интересами каждого пользователя.

Традиционные подходы к организации информационных систем, основанные на централизованной архитектуре и реляционных базах данных, оказываются неэффективными при работе с такими объёмами данных. Необходимость обеспечить низкую задержку, высокую пропускную способность и одновременно сохранить логическую консистентность данных диктует новый подход к проектированию архитектуры. Существующие системы должны обрабатывать информацию в реальном времени, выстраивая персонализированные ленты для миллионов одновременно активных пользователей.

Задача построения масштабируемой ленты новостей объединяет несколько сложных проблем: управление потоками данных высокой пропускной способности, распределённое хранение информации, алгоритмическое ранжирование контента и кэширование часто запрашиваемых данных. Решение этих проблем требует применения современных технологий распределённых систем, включая системы обработки потоков данных, NoSQL базы данных и механизмы горизонтального масштабирования.

Цель данной статьи – изложить комплексный подход к проектированию архитектуры масштабируемой ленты новостей, рассмотрев все ключевые компоненты, их взаимодействие, и современные технические решения, применяемые крупными платформами.

Слой сбора и потоковой обработки данных

Первым критически важным компонентом архитектуры является система для сбора и обработки потоков данных в реальном времени. Apache Kafka выступает в качестве центрального брокера сообщений, обеспечивающего асинхронную доставку событий (публикация статей, лайки, комментарии и т.д.) от производителей данных к потребителям [1].

Архитектура Kafka основана на концепции распределённого, отказоустойчивого журнала событий. Ключевые преимущества [2]:

– Высокая пропускная способность: система способна обрабатывать сотни тысяч или миллионы событий в секунду;

– Надёжность доставки: репликация сообщений на нескольких узлах обеспечивает сохранение данных даже при отказе отдельных серверов;

– Масштабируемость: добавление новых партиций позволяет горизонтально расширять пропускную способность;

– Отстранение производителей и потребителей: асинхронная архитектура позволяет отделить логику создания контента от логики его обработки и доставки.

Для обработки потоков данных используются специализированные фреймворки, такие как Apache Flink, Apache Spark Streaming, Kafka Streams. Эти системы позволяют применять трансформации, фильтрацию и агрегацию к потокам событий в реальном времени. Например, система может немедленно агрегировать статистику просмотров, лайков и комментариев, и обновлять ранжирование контента на основе свежих данных [3].

Архитектура ленты новостей сочетает два подхода: потоковую обработку для немедленного обновления метрик и батч-обработку для более сложных аналитических задач и переобучения моделей машинного обучения. Такой гибридный подход, часто называемый Lambda архитектурой, позволяет достичь оптимального баланса между скоростью и точностью обработки.

При проектировании сервиса формирования персональной ленты ключевым архитектурным решением является выбор стратегии распространения новых публикаций и событий. На практике выделяют три подхода [4]:

– Fan-out on write. При записи нового поста система проактивно распределяет запись в пользовательские ленты. Это позволяет получить очень быстрые операции чтения, однако записи становятся затратными – при наличии «звёздных» аккаунтов количество операций записи растёт линейно с числом подписчиков и может стать узким местом;

– Fan-out on read (pull). При обращении пользователя система динамически генерирует ленту, агрегация кандидатов и ранжирование выполняются на чтении. Записи дешёвые, но время ответа и нагрузка на службе чтения выше;

– Гибридный подход. Комбинация: precompute для большинства «обычных» подписчиков и on-read для «звёздных». Также часто precompute выполняется лишь для теплой части аудитории или для «звёздных» сигналов. Такой подход даёт компромисс между стоимостью записи и целевыми SLA на чтение.

Распределённое хранилище данных

Для хранения масштабируемой ленты новостей требуются базы данных, способные эффективно работать с неструктурированными или слабоструктурированными данными и предусматривающие горизонтальное масштабирование. NoSQL – стандартный выбор в этой области [5].

Основные NoSQL технологии:

– Apache Cassandra – распределённая СУБД с ориентиром на высокую доступность и большую пропускную способность при записи. Использует модель широких столбцов и проектируется под конечную (eventual) согласованность данных;

– MongoDB – документоориентированная база, хранящая записи в виде JSON-подобных документов. Характеризуется гибкой схемой данных и поддержкой транзакций на уровне отдельных документов;

– Redis – in-memory хранилище «ключ-значение», обеспечивающее чрезвычайно низкую задержку доступа. Широко применяется для кэширования, управления сессиями и хранения горячих данных.

Одной из проблем масштабирования является распределение данных между несколькими серверами. Шардирование – техника, при которой набор данных разбивается на логические подмножества, каждое из которых хранится на отдельном узле.

Стратегии шардирования:

– Шардирование по пользователю: все данные одного пользователя хранятся на одном шарде, что обеспечивает быстрый доступ при чтении ленты;

– Шардирование по идентификатору поста: данные о каждом посте распределяются по шардам, что облегчает репликацию и параллельную обработку;

– Адаптивное шардирование: система динамически перераспределяет данные в зависимости от текущей нагрузки, перемещая часто используемые данные на узлы с большей пропускной способностью.

Кэширование и ускорение доступа

Кэширование является критическим компонентом для достижения низкой задержке. Архитектура включает несколько уровней кэша [6]:

– Кэш на уровне приложения: данные, часто запрашиваемые приложением, хранятся в оперативной памяти приложения;

– Кэш на уровне сервиса: общий кэш Redis, разделяемый между несколькими экземплярами приложения, обеспечивает согласованность данных и снижает нагрузку на основное хранилище;

– Кэш на уровне CDN: контент кэшируется на географически распределённых узлах Content Delivery Network для ускорения доставки конечным пользователям.

Поскольку объём кэша ограничен, необходимо определить, какие данные удалять при переполнении. Наиболее распространённые стратегии [6]:

– LRU (Least Recently Used): удаляются данные, к которым давно не было обращений;

– LFU (Least Frequently Used): удаляются данные с наименьшим числом обращений;

– TTL (Time To Live): данные автоматически удаляются по истечении заданного времени жизни.

Инвалидация кэша требует синхронизации между системой хранения и кэшем. При обновлении данных в основной БД соответствующие записи в кэше должны быть немедленно обновлены или удалены.

Алгоритмы ранжирования контента

Ядром ленты новостей является алгоритм, определяющий порядок отображения контента. Эффективное ранжирование учитывает множество факторов:

– Социальный граф: посты от близких друзей обычно имеют более высокий приоритет;

– История взаимодействия: система анализирует, какой контент пользователь обычно лайкает или комментирует, и предпочитает подобный контент;

– Временной компонент: свежий контент обычно более интересен, чем старый;

– Метаданные контента: тематика, жанр, язык поста;

– Ранжирование может быть реализовано на основе различных моделей машинного обучения, от простых линейных моделей до глубоких нейронных сетей.

Основным подходом является коллаборативная фильтрация – метод, предполагающий, что пользователи с похожими интересами будут одинаково оценивать один и тот же контент.

Методы коллаборативной фильтрации:

– На основе пользователей: находятся пользователи с похожими предпочтениями и рекомендуется контент, понравившийся соседним пользователям;

– На основе элементов: если пользователю понравился пост, рекомендуются похожие посты на основе выученных векторных представлений – embeddings;

– Матричная факторизация: предпочтения пользователей разлагаются в виде произведения двух матриц меньшего ранга, что позволяет выявить скрытые признаки.

Первым этапом для рекомендательной системы обычно является генерация ограниченного набора кандидатов из огромного пула контента. Современные практики используют векторные представления элементов и пользователей и применяют методы приближённого поиска ближайших соседей (ANN) для масштабируемого извлечения похожих элементов [7].

Интеграция в pipeline:

– комбинировать несколько генераторов (graph-based, content-based via ANN, recent windows и business-rules);

– использовать ANN-индексы для предварительного отбора ~100 – 1000 кандидатов, далее применять быстрый LTR и затем более тяжёлые re-rankers;

– учитывать требования к задержке при выборе реализации ANN.

Обеспечение надёжности и консистентности

Для обеспечения отказоустойчивости каждая единица данных реплицируется на несколько узлов. Стратегии репликации:

– Синхронная репликация: изменение считается завершённым только после того, как оно записано на все реплики. Обеспечивает строгую консистентность, но замедляет операции;

– Асинхронная репликация: клиент получает подтверждение после записи на основной узел, а репликация на другие узлы происходит позднее. Быстрее, но несёт риск потери данных при сбое основного узла.

В распределённых системах возникают сложности при обработке конфликтов между записями, выполняемыми на разных узлах. Разрешение конфликтов может быть основано на временных метках или на применении пользовательских функций слияния [8].

Мониторинг и оптимизация производительности

Операционная сложность масштабируемой ленты новостей требует постоянного мониторинга ключевых метрик производительности:

– Задержка: время отклика системы на запрос;

– Пропускная способность: количество обработанных запросов в единицу времени;

– Доступность: процент времени, в течение которого система функционирует без ошибок.

Инструменты для мониторинга и логирования такие как: ELK Stack, Prometheus, Grafana позволяют выявлять узкие места и оптимизировать системные параметры.

Заключение

Проведённое исследование позволило рассмотреть проблему проектирования масштабируемой ленты новостей и сформировать архитектурное решение, ориентированное на обработку больших объёмов данных в режиме, близком к реальному времени. В ходе работы была разработана архитектура, объединяющая системы потоковой обработки событий, распределённые NoSQL-хранилища, механизмы горизонтального масштабирования и многоуровневого кэширования. Использование брокера сообщений и стриминговых фреймворков обеспечивает высокую пропускную способность и оперативное об-

новление метрик взаимодействия пользователей с контентом. Применение стратегий шардирования и репликации позволяет достичь отказоустойчивости и гибкости системы при росте нагрузки, а использование кэширования существенно снижает задержку при формировании ленты. Рассмотренные алгоритмы ранжирования и рекомендательные подходы обеспечивают персонализацию контента с учётом социальных связей, истории взаимодействий и временных факторов. Полученные результаты подтверждают, что предложенная архитектура соответствует требованиям современных информационных платформ и может быть использована в качестве практической основы для построения высоконагруженных систем формирования новостных лент.

Список литературы

1. Kreps J., Narkhede N., Rao J. Kafka: A distributed messaging system for log processing, 2011.

2. Singh I., Vivek C. Real-time event joining in practice with Kafka and Flink // arXiv preprint arXiv:2410.15533, 2024.

3. Carbone P., Katsifodimos A. et al. Apache Flink: stream and batch processing in a single engine // Bulletin of the IEEE Computer Society Technical Committee on Data Engineering, 2015.

4. Алексеева К. А. Применение паттерна FAN-OUT для обновления лент в высоконагруженных веб-приложениях: выпускная квалификационная работа. Минск: БГУИР, 2025.

5. Davoudian A., Chen C., Liu M. A survey on NoSQL stores // ACM Computing Surveys (CSUR), 2018.

6. Megiddo N., Modha D.S. ARC: A self-tuning, low overhead replacement cache // Proceedings of the 2nd USENIX Symposium on File and Storage Technologies (FAST), 2003.

7. Ульянов М. В. Приближённые ближайшие соседи Москва, 2015.

8. Bailis P., Venkataraman S., Franklin M.J., et al. Probabilistically Bounded Staleness for Practical Partial Quorums, 2012.

ОБЗОР МЕТОДОВ ОЦЕНКИ ИНВЕСТИЦИЙ В КИБЕРБЕЗОПАСНОСТИ. ПРОЦЕСС УПРАВЛЕНИЯ РИСКАМИ

Романюк М. А., Мозговенко А.А.

ФГБОУ ВО «Мелитопольский государственный университет», Мелитополь, e-mail: ya@amozgovenko.ru

Задачи исследования:

1. Систематизировать основные виды угроз, связанных с распространением сообщений в социальных сетях.

2. Оценить масштаб и последствия угроз для отдельных пользователей и социума.

Материалы и методы исследования

Современные исследования в области цифровой безопасности выделяют несколько ключевых направлений анализа угроз в социальных сетях:

1. Киберпреступность и мошенничество. Работы последних лет (2022–2025 гг.) акцентируют внимание на эволюции фишинговых атак и методах социальной инженерии. Отмечается рост использования deepfake – технологий для создания поддельных сообщений и имитации доверенных источников.

2. Дезинформация и манипулирование. Исследования в области медиапсихологии и политической коммуникации фиксируют усиление роли соцсетей как каналов распространения фейков и пропаганды. Особое внимание уделяется алгоритмическому усилению дезинформации и её влиянию на общественное мнение.

3. Психологические угрозы. Публикации по киберпсихологии описывают рост случаев кибербуллинга, сталкинга и онлайн харассмента. Анализируются долгосрочные последствия для психического здоровья жертв, включая тревожные расстройства и суицидальные мысли.

4. Утечка данных и приватность. Исследования в сфере информационной безопасности демонстрируют, что сообщения в соцсетях часто становятся источником компрометации персональных данных через:

– скрапинг информации;

– анализ метаданных;

– эксплуатацию уязвимостей платформ.

5. Экстремистские угрозы. Работы по криминологии и противодействию терроризму отмечают использование соцсетей для вербовки и координации противоправной деятельности. Активно изучаются методы выявления и блокировки экстремистского контента.

Основная цель – создать типологию угроз, порождаемых сообщениями в социальных сетях.

Результаты исследования и их обсуждение

С увеличением использования социальных сетей многие пользователи стали уязвимыми к угрозам своей приватности и безопасности. Эти опасности могут быть разделены на 4 главные категории (рис. 1.1). Первая категория содержит классические угрозы, в частности, угрозы конфиденциальности и безопасности, которые не только угрожают пользователям соцсетей, но и пользователям Интернета, не использующим социальные сети. Вторая категория охватывает современные угрозы, то есть угрозы, в основном уникальные для среды социальных сетей, и которые используют инфраструктуру социальных сетей для угрозы конфиденциальности и безопасности пользователя. Третья категория состоит из комбинированных угроз, где мы описываем, как сегодняшние нападающие могут, и часто делают, совмещать разные типы атак для создания более сложных и летальных приступов. Четвертая и последняя категория включает в себя угрозы, специально ориентированные на детей, использующих социальные сети.

Классические угрозы, такие как вредоносное ПО, фишинг, спам и межсайтовый скриптинг (XSS), остаются актуальными в эпоху социальных сетей. Они быстро распространяются среди пользователей, используя их личные данные и доверие.