«голубь» не только этически предпочтительна, но и эволюционно устойчива, занимая большую часть равновесия.

Список литературы

1. Деулофеу Х. Теория игр. Дилемма заключённого и доминантные стратегии. Москва: Ленанд, 2014. 272 с.

2. Nodder C. Evil by Design: Interaction Design to Lead Us into Temptation. Indianapolis: John Wiley & Sons, 2013. 258 p.

3. Hick W. E. On the rate of gain of information // Quarterly Journal of Experimental Psychology. 1952. Vol. 4. No. 1. P. 11–26.

4. Willis J., Todorov A. First Impressions: Making Up Your Mind After a 100-ms Exposure to a Face // Psychological Science. 2006. Vol. 17. No. 7. P. 592–598.

5. Wong A. L., Goldsmith J., Forrence A. D., Haith A. M., Krakauer J. W. Reaction times can reflect habits rather than computations // eLife. 2017. Vol. 6. Article e28075. P. 1–18. DOI: 10.7554/eLife.28075.

СРАВНЕНИЕ ПРОИЗВОДИТЕЛЬНОСТИ JAVA И C++ ПРИ СОРТИРОВКЕ ЦЕЛЫХ ЧИСЕЛ И ЗАПИСИ/ЧТЕНИИ ФАЙЛОВ

Романенко И. В.

ФГБОУ ВО «Мелитопольский государственный университет», Мелитополь, e-mail: ryukjwfkr@gmail.com

Научный руководитель: Ступницкий В. С.

Введение

Выбор языка программирования оказывает существенное влияние на производительность, надёжность и сопровождаемость программного обеспечения. Особенно важен этот выбор при разработке приложений, требующих эффективной обработки данных и выполнение базовых операций, таких как сортировка и работа с файлами. Среди множества языков особое внимание заслуживают C++ и Java, два объектно-ориентированных языка, широко применяемых в промышленной разработке, но реализованных на разных уровнях системной абстракции. C++ компилируемый язык, близкий к аппаратному уровню, предоставляющий разработчику полный контроль над ресурсами системы. Однако эта гибкость сопряжена с повышенной сложностью: ошибки в управлении память. (например, утечки или двойное освобождение) могут привести к нестабильной работе. Эти недостаток можно смягчить за счёт строго соблюдение правил RAII (Resource Acquisition Is Initialization) и использования современных средств языка, таких как автоматические локальные переменные и контейнеры стандартной библиотеки (например, std::vector), которые управляют памятью самостоятельно. Java, напротив, выполняется в среде виртуальной машины и обеспечивает автоматическое управление памятью с помощью сборщика мусора. Это повышает надёжность и упрощает разработку, но вносит накладные расходы, что может снизить производительность в вычислительных интенсивных задачах. Этот недостаток так же можно частично компенсировать настройкой параметров JVM и применением пулов памяти или off-heap решений. Эти особенности делают сравнение языков актуальным: несмотря на схожесть в синтаксисе и парадигмах, они поразному влияют на эффективность выполнения одних и тех же операций. Ранние исследования (например, Prechelt, 1999; Nikishkov et al., 2003) отмечали значительное отставание Java по скорости и потреблению памяти. Однако современные версии компиляторов и виртуальных машин существенно улучшили ситуацию, что требует актуального повторного сравнения в контексте типовых задач.

Цель исследования – провести сравнительный анализ производительности языков программирования C++ и Java при выполнении базовых операций, таких как сортировка целых чисел, чтение и запись файлов, с целью выявления их относительных преимуществ и недостатков в типичных сценариях использования.

Материалы и методы исследования

Для сравнения производительности языков C++ и Java были созданы два приложения с идентичным дизайном, осуществляющие загрузку, сортировку и сохранение массивов целых чисел. Эксперимент проводился на наборах данных разного объёма (1000, 10 000, 100 000 элементов). Время выполнение каждой операции измерялось отдельно. Для измерения времени в C++ использовалась стандартная библиотека ctime. Она содержит тип clock_t и функцию clock. Вместе они позволяют получить количество тактов процессора с момента запуска системы. Чтобы измерить длительность события (например, загрузки данных), вычитали начальное время из конечного. Для этого фиксировали время до и после события. Для Java использовался стандартный пакет System, в котором есть встроенная функция времени. Для измерения времени в миллисекундах использовалась функция currentTimeMillis. Время фиксировалось до и после события, затем вычислялась разница. У Java нет таких же возможностей для ручной оптимизации кода, как у C++, поскольку она компилируется собственным компилятором javac в байт-код, который затем выполняется на виртуальной машине Java (JVM). Этот компилятор автоматически оптимизирует код. Чтобы сделать сравнение максимально справедливым, среда разработки C++ была настроена на максимальную оптимизацию кода с использованием флага O2 и запуск в режиме Release, что исключает накладные расходы на откладку и оптимизирует код настолько, насколько это возможно. Исходя из этого составим схему принципа работы системы (рис. 1).

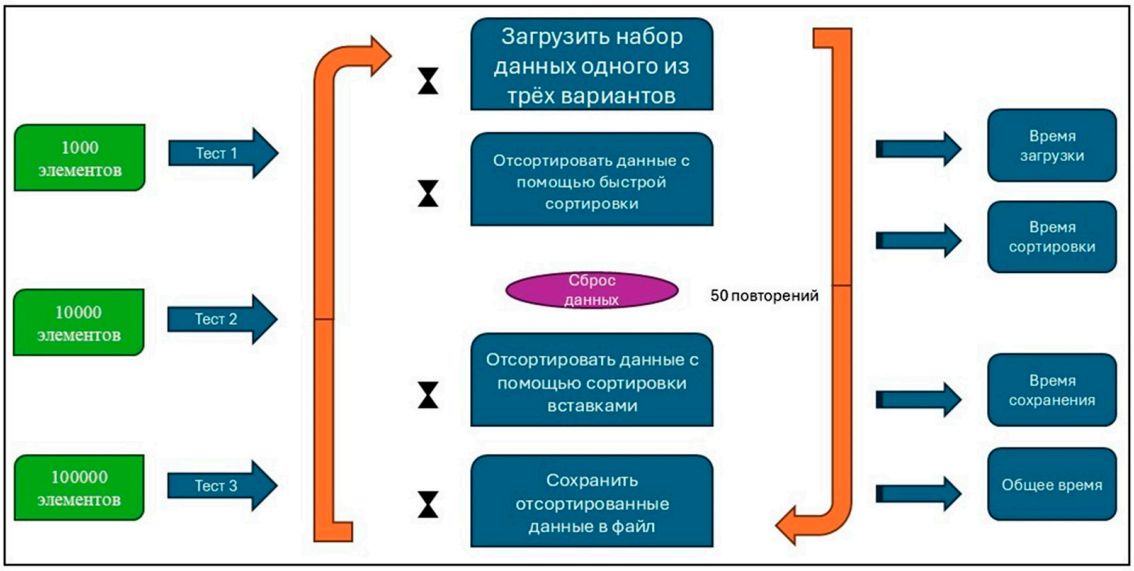

Рис. 1. Схема принципа работы

Каждый тест повторялся 50 раз для повышения достоверности результатов. Сравнительный анализ проводился на основе усреднённых значений времени выполнения. Приложения были созданы в разных интегрированных средах разработки, Java приложение было создано в VS Code, а приложение на C++ было запрограммировано в Visual Studio 2022. Оба проекта были созданы как 32-разрядные. Этот тест проводился на 64-разрядной операционной системе с Windows 11, 16 ГБ оперативной памяти и процессором i5, работающим на частоте 4,4 ГГц.

Результаты исследования и их обсуждение

Результаты показывают, что Java в целом работает быстрее, чем C++, а также быстрее загружает два из трёх наборов данных. C++ в целом быстрее при сортировке данных обоими алгоритмами и при сохранении в файлы. При чтении данных из файла посимвольно Java оказалась быстрее используя объекты File и Scanner, по сравнению с C++, который использовался стандартную библиотеку fstream. Java быстрее загружала два больших набора данных, но была медленнее при загрузке самого маленького – примерно на две секунды. Разница для других двух наборов составила около одной секунды для 10 000 целых чисел и около 22 секунд для 100 000. Медленная загрузка данных в C++ привела к тому, что общее время выполнения оказалась больше. В среднем, Java была быстрее благодаря более эффективному чтению файлов, однако при сортировке и обработке данных C++ в целом быстрее, особенно при сортировке самого большого набора с помощью сортировки вставками (на восемь секунд быстрее) и быстрой сортировки (на более чем секунду быстрее). Сохранение данных в конце каждой итерации было быстрее в C++ на несколько секунд, особенно для самого большого набора данных, где разница составила около двух секунд. Результаты показывают, что с увеличением размера наборов данных разница во времени растёт значительно. Масштабируемость Java лучше при загрузке данных, но при сортировке с помощью quicksort и сохранении данных C++ масштабируется лучше. Сортировка вставками плохо масштабируется в обоих приложениях. Таблицы и графики наглядно показывают эти различия (табл. 1, 2).

Таблица 1

Результаты тестирования производительности C++
(время выполнения в миллисекундах)

Размер набора данных	Load	Qsort	iSort	Save	Total
1000	0,9 мс	0,04 мс	0,1 мс	1,64 мс	2,68 ms
10000	7,5 мс	0,5 мс	12,62 мс	8,52 мс	29,14 ms
100000	73,18 мс	6,04 мс	1208,58 мс	16,94 мс	1304, 74 ms

Таблица 2

Результаты тестирования производительности Java
(время выполнения в миллисекундах)

Размер набора данных	Load	qSort	iSort	Save	Total
1000	2,34 мс	0,06 мс	0,22 мс	2,18 мс	4,8 мс
10000	6,44 мс	0,76 мс	12,48 мс	8,08 мс	27,76 мс
100000	51,28 мс	7,66 мс	1216,06 мс	19,9 мс	1249,9 мс

Ось Y графиков отображает размер набора данных, а ось X – время в миллисекундах. На первом графике отчётливо видно, что C++ (оранжевый) значительно медленнее и хуже масштабируется по сравнению с Java (синий) при загрузке из файла. C++ быстрее только при работе с наименьшим набором данных. Так же нужно обратить внимание, что загрузка самого большого файла заняла больше семи секунд (73 миллисекунд) (рис. 2).

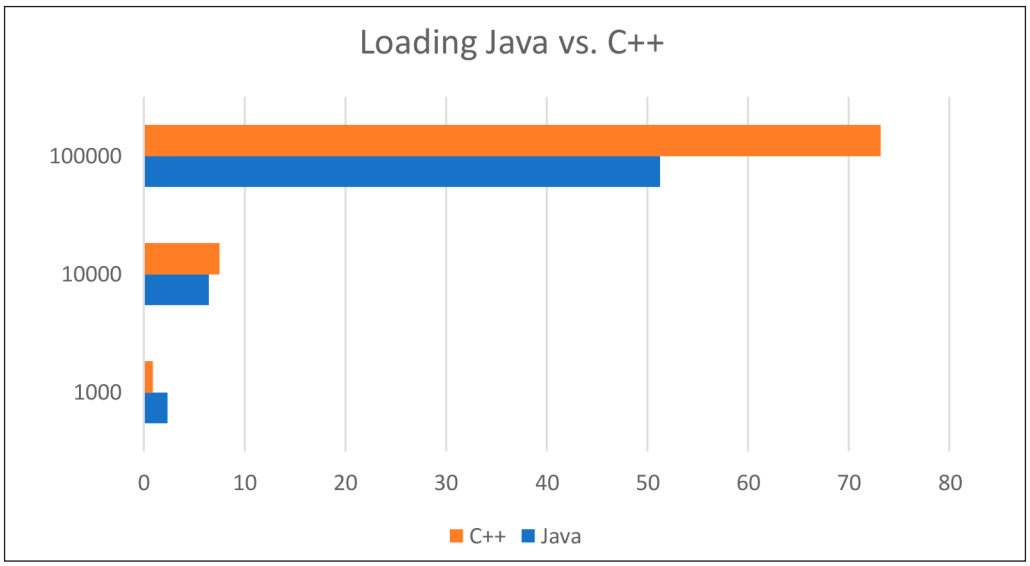

Рис. 2. Сравнение загрузки

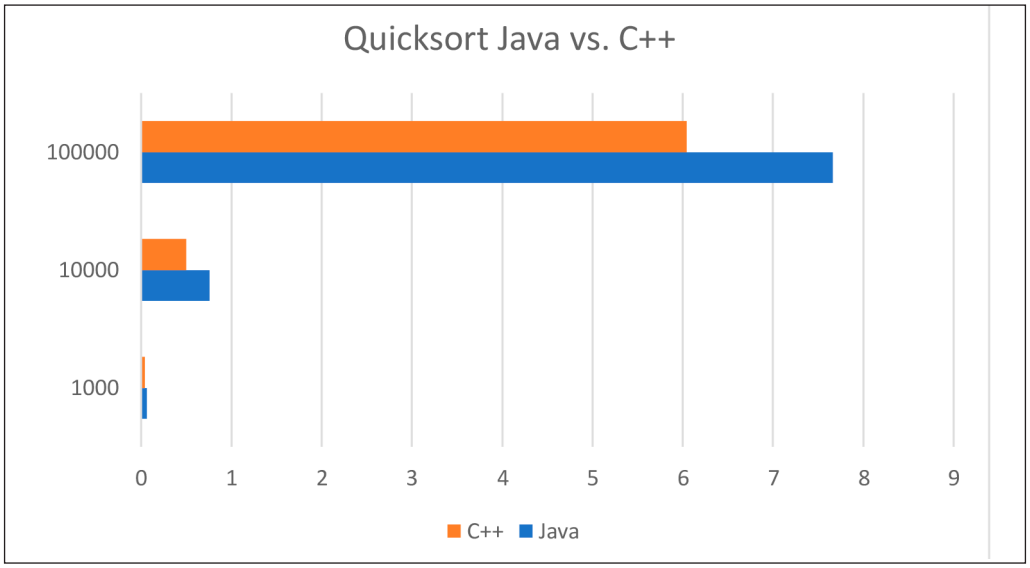

Рис. 3. Quicksort сравнение

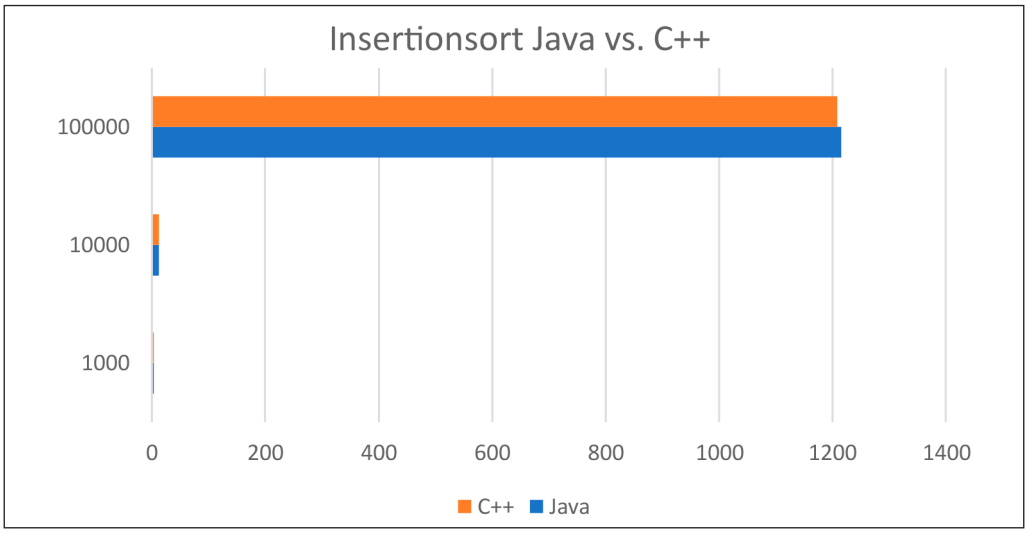

Рис. 4. Сравнение вставок и сортировки

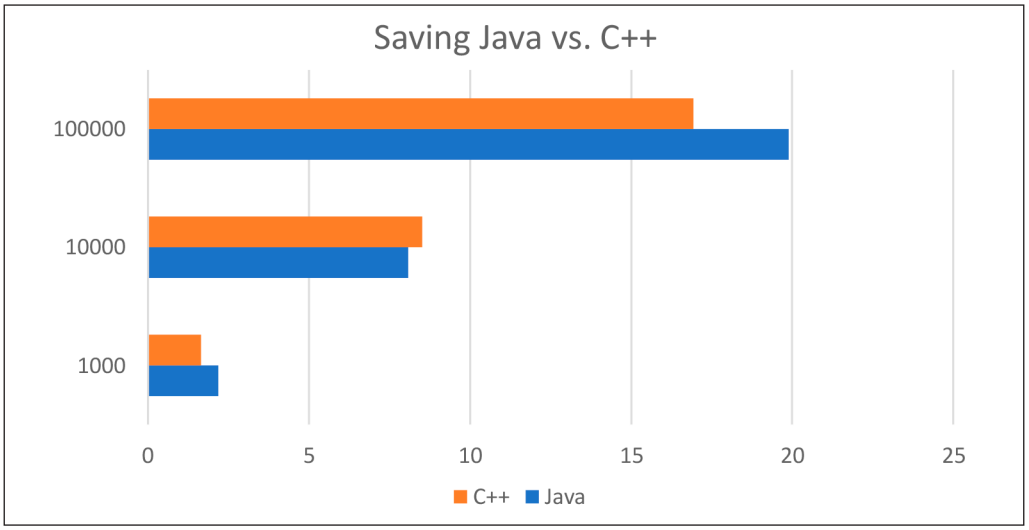

Рис. 5. Сравнение сохранения

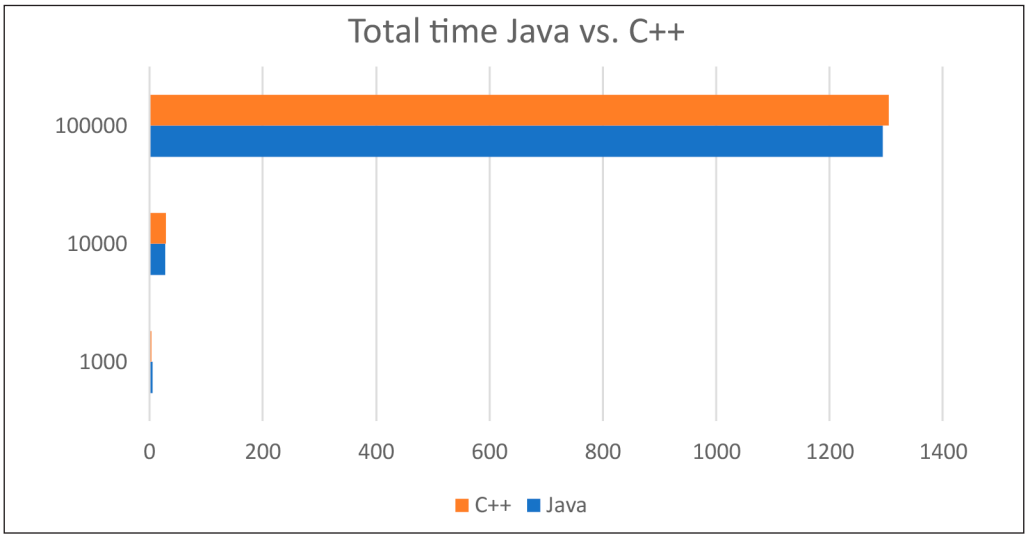

Рис. 6. Общее сравнение

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].