Сравнительный анализ подходов к разработке архитектуры и систем управления базами данных для высоконагруженных WEB-сервисов

**Человек:** В современном мире для разработчиков все острее встает проблема обработки и хранения больших объемов данных. Сообщения в социальных сетях, фотографии, потоковое видео – все это создает высокую нагрузку на программное обеспечение, используемое на серверах. По этой причине стандартные подходы, используемые для проектирования архитектуры настольных приложений, чаще всего будут неэффективны, так как в большинстве случаев они не учитывают нагрузку на приложение со стороны огромного числа пользователей. На сегодняшний день нет четкого определения для высоконагруженных систем. В большинстве случаев этот термин применяются в ситуациях, когда приложение перестает справляться с моментальной нагрузкой, возложенной на него. Нельзя указать конкретных значений, по достижении которых система считается высоконагруженной, поскольку все приложение специфичны и одинаковое количество запросов может приводить к абсолютно разным нагрузкам на ресурсы. В ходе исследования систем управления базами данных было проведено несколько опытов замеряющих скорость выполнения основных операций с базами данных: добавление, выборка и удаление. На основании результатов этих опытов были сделаны выводы и даны рекомендации по выбору системы управления базами данных. В данной статье рассмотрены подходы к разработке высоконагруженных систем, выделены недостатки и преимущества каждого из подходов и приведены примеры использования этих подходов такими популярными сервисами, как ВКонтакте, Facebook, Google и Яндекс. Приведен сравнительный анализ систем управления базами данных MySQL и MongoDB. В заключении даны рекомендации по выбору СУБД в зависимости от подхода к проектированию архитектуры высоконагруженного проекта.

**Key words:** высоконагруженные программные системы, архитектура приложений, хранение данных, базы данных, MySQL, MongoDB, СУБД, программное приложение, большие объемы данных, реляционные базы данных

=================================

**FastText\_KMeans\_Clean:** Недостатки данного подхода:. Во-вторых, из-за высокой нагрузки на API, для его корректной работы необходимо высокопроизводительное аппаратное обеспечение. Для проведения эксперимента по замеру средней скорости выборки элементов был разработан скрипт, создающий 250 000 записей в базе и проводящий выборку всех элементов базы через каждые 5 000 записей. Результаты эксперимента в таблице 2:. Поиск среднего значения. Если имеется достаточное количество разработчиков, обладающих знаниями в области производства высоконагруженных систем, то можно проектировать сервис-ориентированное приложение используя эвристический подход. В ходе эксперимента, приведенного в статье, было доказано, что не реляционная СУБД справляется с основными простейшими операциями на порядок быстрее MySQL, и если специфика проекта не подразумевает осуществление сложных запросов к базе, например, таких как JOIN, то лучше использовать не реляционную базу данных. Дополнительным преимуществом MongoDB является относительная простота горизонтального масштабирования, по сравнению с MySQL.

**Key words part:** 0.7931034482758621

=================================

**FastText\_KMeans\_Raw/:** Недостатки данного подхода:. Вертикальное – увеличение производительности системы за счет установки более мощного аппаратного обеспечения. Рисунок 3 - График зависимости среднего времени выполнения запроса (в секундах) от количества элементов в базе данных при выборке элементов:. В ходе эксперимента, приведенного в статье, было доказано, что не реляционная СУБД справляется с основными простейшими операциями на порядок быстрее MySQL, и если специфика проекта не подразумевает осуществление сложных запросов к базе, например, таких как JOIN, то лучше использовать не реляционную базу данных.

**Key words part:** 0.6551724137931034

=================================

**FastText\_PageRank\_Clean/:** При промышленном – отдельно. Промышленный подход к разработке архитектуры. Преимущества подхода:. Результаты эксперимента в таблице 2:. Поиск среднего значения. Результаты эксперимента в таблице 3:. Графически зависимость отображается следующим образом:. Удаление элемента по индексу.

**Key words part:** 0.3448275862068966

=================================

**FastText\_PageRank\_Raw/:** При промышленном – отдельно. Промышленный подход к разработке архитектуры. Преимущества подхода:. Результаты эксперимента в таблице 2:. Поиск среднего значения. Результаты эксперимента в таблице 3:. Графически зависимость отображается следующим образом:. Удаление элемента по индексу.

**Key words part:** 0.3448275862068966

=================================

**Mixed\_ML\_TR/:** Разница между подходами заключается в способе разработки средств масштабирования. Основным преимуществом данного подхода является быстрая разработка приложений. Преимущество данного подхода заключается в том, что отсутствует огромная нагрузка на главную шину, по которой в случае промышленного подхода пропускаются все данные и проходят запросы. В данном эксперименте сравнивалась производительность основных операций с базами данных, таких как вставка, выборка, удаление и поиск среднего значения. Для проведения эксперимента по замеру средней скорости выборки элементов был разработан скрипт, создающий 250 000 записей в базе и проводящий выборку всех элементов базы через каждые 5 000 записей. Рисунок 3 - График зависимости среднего времени выполнения запроса (в секундах) от количества элементов в базе данных при выборке элементов:. Если имеется достаточное количество разработчиков, обладающих знаниями в области производства высоконагруженных систем, то можно проектировать сервис-ориентированное приложение используя эвристический подход. В случае, когда у компании достаточное количество временных и денежных ресурсов лучший вариант – промышленный подход к реализации высоконагруженной системы, поскольку есть возможность купить дорогостоящее аппаратное обеспечение и потратить много времени на разработку API.

**Key words part:** 0.6551724137931034

=================================

**MultiLingual\_KMeans/:** Разница между подходами заключается в способе разработки средств масштабирования. Преимущество данного подхода заключается в том, что отсутствует огромная нагрузка на главную шину, по которой в случае промышленного подхода пропускаются все данные и проходят запросы. Для проведения эксперимента по замеру средней скорости выборки элементов был разработан скрипт, создающий 250 000 записей в базе и проводящий выборку всех элементов базы через каждые 5 000 записей. Если имеется достаточное количество разработчиков, обладающих знаниями в области производства высоконагруженных систем, то можно проектировать сервис-ориентированное приложение используя эвристический подход.

**Key words part:** 0.6551724137931034

=================================

**Multilingual\_PageRank/:** При промышленном – отдельно. Недостатки данного подхода:. Такой подход называется эвристическим. Каждые 5 000 записей замерялось среднее время выполнения запроса. Аналогично прошлому эксперименту, считалось среднее время выполнения запроса на интервале через 5 000. Как видно из графика, при малом количестве данных в базе (до 50 000 записей) разница практически не ощутима. Результаты эксперимента в таблице 3:. Удаление элемента по индексу.

**Key words part:** 0.5172413793103449

=================================

**RuBERT\_KMeans\_Without\_ST/:** При эвристическом подходе средства разрабатываются совместно с бизнес-логикой. При промышленном – отдельно. Второе неоспоримое преимущество – возможность найма разработчиков, далеких от создания сервисов с высокой нагрузкой. Результаты эксперимента в таблице 3:. Если стоит задача в сжатые сроки произвести основной функционал приложения, к примеру, для показа прототипа инвестору, то логичнее проектировать монолитное приложение, поскольку это позволить уменьшить срок разработки. Если имеется достаточное количество разработчиков, обладающих знаниями в области производства высоконагруженных систем, то можно проектировать сервис-ориентированное приложение используя эвристический подход. В ходе эксперимента, приведенного в статье, было доказано, что не реляционная СУБД справляется с основными простейшими операциями на порядок быстрее MySQL, и если специфика проекта не подразумевает осуществление сложных запросов к базе, например, таких как JOIN, то лучше использовать не реляционную базу данных.

**Key words part:** 0.7586206896551724

=================================

**RuBERT\_KMeans\_With\_ST/:** Основным преимуществом данного подхода является быстрая разработка приложений. Каждые 5 000 записей замерялось среднее время выполнения запроса. Результаты эксперимента в таблице 3:. Графически зависимость отображается следующим образом:. Однако это полезно лишь в случае сервис-ориентированной архитектуры приложения, поскольку монолитное приложение нельзя горизонтально масштабировать, а следовательно, в большинстве случаев для такого проекта можно обойтись обычной реляционной СУБД, такой как MySQL или PostgreSQL.

**Key words part:** 0.5172413793103449

=================================

**RUBERT\_page\_rank\_Without\_ST/:** Вертикальное – увеличение производительности системы за счет установки более мощного аппаратного обеспечения. Таким образом, в итоге придется масштабировать приложение горизонтально, то есть разносить функционал на модули, которые будут выполняться на отдельных серверах. Поиск среднего значения. Удаление элемента по индексу. При таком подходе квалификация специалиста имеет огромное значение.

**Key words part:** 0.4137931034482759

=================================

**RUBERT\_page\_rank\_With\_ST/:** При промышленном – отдельно. Преимущества подхода:. Результаты эксперимента в таблице 2:. Поиск среднего значения. Результаты эксперимента в таблице 3:.

**Key words part:** 0.3103448275862069

=================================

**RUSBERT\_KMeans\_Without\_ST/:** Условно их называют промышленный и эвристический . При эвристическом подходе средства разрабатываются совместно с бизнес-логикой. В случае, когда у компании достаточное количество временных и денежных ресурсов лучший вариант – промышленный подход к реализации высоконагруженной системы, поскольку есть возможность купить дорогостоящее аппаратное обеспечение и потратить много времени на разработку API. Однако это полезно лишь в случае сервис-ориентированной архитектуры приложения, поскольку монолитное приложение нельзя горизонтально масштабировать, а следовательно, в большинстве случаев для такого проекта можно обойтись обычной реляционной СУБД, такой как MySQL или PostgreSQL.

**Key words part:** 0.5862068965517241

=================================

**RUSBERT\_KMeans\_With\_ST/:** Масштабируемость[7] — способность системы справляться с увеличением рабочей нагрузки при добавлении ресурсов (обычно аппаратных). Рисунок 3 - График зависимости среднего времени выполнения запроса (в секундах) от количества элементов в базе данных при выборке элементов:. Таблица 3 указывает на то, что вычисление среднего значения для целочисленного значения в MongoDB происходит быстрее, чем в MySQL. Однако это полезно лишь в случае сервис-ориентированной архитектуры приложения, поскольку монолитное приложение нельзя горизонтально масштабировать, а следовательно, в большинстве случаев для такого проекта можно обойтись обычной реляционной СУБД, такой как MySQL или PostgreSQL.

**Key words part:** 0.7931034482758621

=================================

**RUSBERT\_page\_rank\_Without\_ST/:** При промышленном – отдельно. Недостатки данного подхода:. Преимущества подхода:. Поиск среднего значения. Удаление элемента по индексу.

**Key words part:** 0.3103448275862069

=================================

**RUSBERT\_page\_rank\_With\_ST/:** Существует два подхода к реализации сервис-ориентированной архитектуры[5]. Преимущества подхода:. Благодаря этому использование аппаратного обеспечения происходит более эффективно. Существует два вида масштабирования системы: горизонтальное и вертикальное. Поиск среднего значения.

**Key words part:** 0.3793103448275862

=================================

**Simple\_PageRank/:** Во-первых, много времени уходит на разработку общего API – набора классов, функций и констант, предоставляемых приложением для использования во внешних программных продуктах. Таким образом, для разработчиков новых приложений, все методы будут реализованы в API, обращаясь к которому уже не стоит беспокоиться о решении проблемы высокой нагрузки. В данном эксперименте сравнивалась производительность основных операций с базами данных, таких как вставка, выборка, удаление и поиск среднего значения. В случае, когда у компании достаточное количество временных и денежных ресурсов лучший вариант – промышленный подход к реализации высоконагруженной системы, поскольку есть возможность купить дорогостоящее аппаратное обеспечение и потратить много времени на разработку API. Кроме того, крупным компаниям предпочтительнее вложить заранее известную сумму в стоимость аппаратной части и получить рабочее приложение, способное выдерживать высокие нагрузки. В ходе эксперимента, приведенного в статье, было доказано, что не реляционная СУБД справляется с основными простейшими операциями на порядок быстрее MySQL, и если специфика проекта не подразумевает осуществление сложных запросов к базе, например, таких как JOIN, то лучше использовать не реляционную базу данных.

**Key words part:** 0.8275862068965517

=================================

**TextRank/:** Преимущество данного подхода заключается в том, что отсутствует огромная нагрузка на главную шину, по которой в случае промышленного подхода пропускаются все данные и проходят запросы. В данном эксперименте сравнивалась производительность основных операций с базами данных, таких как вставка, выборка, удаление и поиск среднего значения. Рисунок 3 - График зависимости среднего времени выполнения запроса (в секундах) от количества элементов в базе данных при выборке элементов:. Если имеется достаточное количество разработчиков, обладающих знаниями в области производства высоконагруженных систем, то можно проектировать сервис-ориентированное приложение используя эвристический подход. В случае, когда у компании достаточное количество временных и денежных ресурсов лучший вариант – промышленный подход к реализации высоконагруженной системы, поскольку есть возможность купить дорогостоящее аппаратное обеспечение и потратить много времени на разработку API. В ходе эксперимента, приведенного в статье, было доказано, что не реляционная СУБД справляется с основными простейшими операциями на порядок быстрее MySQL, и если специфика проекта не подразумевает осуществление сложных запросов к базе, например, таких как JOIN, то лучше использовать не реляционную базу данных.

**Key words part:** 0.7586206896551724

=================================

**TF-IDF\_KMeans/:** Преимущества подхода:. Таким образом, для разработчиков новых приложений, все методы будут реализованы в API, обращаясь к которому уже не стоит беспокоиться о решении проблемы высокой нагрузки. Вертикальное – увеличение производительности системы за счет установки более мощного аппаратного обеспечения. В случае же сервис-ориентированной архитектуры в целях обеспечения устойчивости данных к разделению лучше использовать NoSQL базы данных. Результаты эксперимента в таблице 2:. В цикле происходила запись в базу 250 000 элементов и через каждые 5 000 элементов происходило вычисление среднего значения поля всей базы. Для проведения опыта по удалению элементов из базы был разработан PHP скрипт, запускающий запрос на удаление записи из базы данных 250 000 раз. Такой подход на первых этапах будет быстрее, чем промышленный, поскольку не будет тратиться время на разработку общего API.

**Key words part:** 0.6551724137931034

=================================

**Текст:** Существует два подхода к реализации сервис-ориентированной архитектуры[5]. Условно их называют промышленный и эвристический . Разница между подходами заключается в способе разработки средств масштабирования. При эвристическом подходе средства разрабатываются совместно с бизнес-логикой. При промышленном – отдельно.. Рассмотрим более подробно промышленный подход (Рисунок 1), поскольку его используют большинство крупных проектов: Facebook, Yandex, Google.. Рисунок 1. Промышленный подход к разработке архитектуры. В приложении, спроектированном промышленным подходом, имеется большая шина (API), на которую отправляются все запросы и от которой приходит обработанный результат. Недостатки данного подхода:. Во-первых, много времени уходит на разработку общего API – набора классов, функций и констант, предоставляемых приложением для использования во внешних программных продуктах.. Во-вторых, из-за высокой нагрузки на API, для его корректной работы необходимо высокопроизводительное аппаратное обеспечение.. Преимущества подхода:. Основным преимуществом данного подхода является быстрая разработка приложений. Поскольку к моменту, когда все API реализовано, приложениям, остается лишь обращаться к его методам и получать необходимые данные.. Второе неоспоримое преимущество – возможность найма разработчиков, далеких от создания сервисов с высокой нагрузкой. Основное API сервисов разрабатывается командой инженеров, специализирующихся на создании высоконагруженных систем. Таким образом, для разработчиков новых приложений, все методы будут реализованы в API, обращаясь к которому уже не стоит беспокоиться о решении проблемы высокой нагрузки.. Обратный подход используется в разработке ВКонтакте. Когда в этой компании разрабатывается какой-либо сервис, разработчику отдают все полномочия и оговаривается, что сервис должен быть масштабируем [6]. Такой подход называется эвристическим.. Основным его недостатком является предъявление высоких требований к квалификации разработчика, поскольку помимо бизнес-логики сервиса он должен позаботиться и о масштабировании своего продукта.. Преимущество данного подхода заключается в том, что отсутствует огромная нагрузка на главную шину, по которой в случае промышленного подхода пропускаются все данные и проходят запросы. В эвристическом подходе каждый сервис борется с высокой нагрузкой самостоятельно. Благодаря этому использование аппаратного обеспечения происходит более эффективно.. Еще одним немаловажным преимуществом сервис-ориентированного подхода является возможность его горизонтального масштабирования.. Масштабируемость[7] — способность системы справляться с увеличением рабочей нагрузки при добавлении ресурсов (обычно аппаратных). Существует два вида масштабирования системы: горизонтальное и вертикальное. Вертикальное – увеличение производительности системы за счет установки более мощного аппаратного обеспечения. Горизонтальное – увеличение за счет количества серверов.. Вертикально масштабировать проект значительно проще, достаточно лишь снабдить сервера более мощными процессорами или добавить оперативной памяти. Проблема такого подхода в том, что на определенном этапе не будет существовать подходящего аппаратного обеспечения, способного справится с текущими нагрузками. Таким образом, в итоге придется масштабировать приложение горизонтально, то есть разносить функционал на модули, которые будут выполняться на отдельных серверах.. Поскольку монолитное приложение представляет из себя единую структуру, его нельзя декомпозировать на модули, а это означает, что горизонтальное масштабирование применить невозможно.. Теорема Брюера[8] утверждает, что в любой реализации распределенных вычислений можно обеспечить не более двух из трех свойств: согласованность данных, доступность, устойчивость к разделению. В случае с монолитными приложениями все приложение находится в одном месте и устойчивостью к разделению данных можно пренебречь. По этой причине в приложениях с такой архитектурой принято применять реляционные системы управления базами данных, например MySQL или PostgreSQL.. В случае же сервис-ориентированной архитектуры в целях обеспечения устойчивости данных к разделению лучше использовать NoSQL базы данных. Кроме того, они оптимизированы для операций поиска и добавления нового элемента. Это полезно для анализа статических элементов в больших объемах данных. В качестве примера такой базы данных можно привести MongoDB[9].. В данном эксперименте сравнивалась производительность основных операций с базами данных, таких как вставка, выборка, удаление и поиск среднего значения. При проведении тестов использовался язык PHP версии 5.4.9 и персональный компьютер со следующими характеристиками: процессор Intel Core i5 2.80 GHz, ОЗУ 8GB, ОС Windows 7.. Вставка элементов:. Для проведения эксперимента по замеру скорости вставки элементов был написан PHP-скрипт, создающий в базе 250 000 записей. Каждые 5 000 записей замерялось среднее время выполнения запроса. Зависимость времени выполнения операции (в секундах) от количества элементов в базе показана в таблице 1.. Таблица 1 показывает, что время выполнения запроса в MongoDB для операции вставки на порядок ниже, чем для СУБД MySQL.. Рисунок 2 показывает график по всем результатам эксперимента вставки:. . Рисунок 2.. Выборка элементов:. Для проведения эксперимента по замеру средней скорости выборки элементов был разработан скрипт, создающий 250 000 записей в базе и проводящий выборку всех элементов базы через каждые 5 000 записей. Аналогично прошлому эксперименту, считалось среднее время выполнения запроса на интервале через 5 000. Результаты эксперимента в таблице 2:. Таблица 2. Рисунок 3 - График зависимости среднего времени выполнения запроса (в секундах) от количества элементов в базе данных при выборке элементов:. . Рисунок 3.. Как видно из графика, при малом количестве данных в базе (до 50 000 записей) разница практически не ощутима. В середине графика (150 000 записей) для обоих СУБД заметен скачок, предположительно, его причина появления обусловлена задержкой при считывании данных с диска.. Поиск среднего значения. Для данного эксперимента был разработан PHP скрипт, делающий выборку из базы среднего значение одного из полей целочисленного типа. В цикле происходила запись в базу 250 000 элементов и через каждые 5 000 элементов происходило вычисление среднего значения поля всей базы. Результаты эксперимента в таблице 3:. Таблица 3 указывает на то, что вычисление среднего значения для целочисленного значения в MongoDB происходит быстрее, чем в MySQL. Графически зависимость отображается следующим образом:. . Рисунок 4.. Удаление элемента по индексу. Для проведения опыта по удалению элементов из базы был разработан PHP скрипт, запускающий запрос на удаление записи из базы данных 250 000 раз.. Результат работы скрипта в таблице 4. Как видно из полученной таблицы при удалении элементов из таблицы производительность MongoDB выше, чем у MySQL.. Полный результат опыта по удалению записей из таблицы в графическом представлении на рисунке 5:. . Рисунок 5.. Разработка программных систем с высокой нагрузкой на сегодняшний день достаточно актуальная тема. Но универсального ответа на вопрос «Как лучше проектировать высоконагруженное приложение» не существует.. Если стоит задача в сжатые сроки произвести основной функционал приложения, к примеру, для показа прототипа инвестору, то логичнее проектировать монолитное приложение, поскольку это позволить уменьшить срок разработки. В случае успеха проекта, для удобства поддержки и поддержания масштабируемости его следует привести к сервис-ориентированной архитектуре.. Если имеется достаточное количество разработчиков, обладающих знаниями в области производства высоконагруженных систем, то можно проектировать сервис-ориентированное приложение используя эвристический подход. Такой подход на первых этапах будет быстрее, чем промышленный, поскольку не будет тратиться время на разработку общего API. Каждый сервис будет масштабироваться отдельно. Проблемы возникнут в дальнейшем, когда необходимо будет расширить проект новыми разработчиками. При таком подходе квалификация специалиста имеет огромное значение.. Так же эвристический подход удобно применять, когда недостаточно средств на мощное аппаратное обеспечение (требуемое при промышленном подходе).. В случае, когда у компании достаточное количество временных и денежных ресурсов лучший вариант – промышленный подход к реализации высоконагруженной системы, поскольку есть возможность купить дорогостоящее аппаратное обеспечение и потратить много времени на разработку API. Кроме того, крупным компаниям предпочтительнее вложить заранее известную сумму в стоимость аппаратной части и получить рабочее приложение, способное выдерживать высокие нагрузки.. Выбор системы управления базами данных зависит от реализации архитектуры приложения и специфики проекта. В ходе эксперимента, приведенного в статье, было доказано, что не реляционная СУБД справляется с основными простейшими операциями на порядок быстрее MySQL, и если специфика проекта не подразумевает осуществление сложных запросов к базе, например, таких как JOIN, то лучше использовать не реляционную базу данных.. Дополнительным преимуществом MongoDB является относительная простота горизонтального масштабирования, по сравнению с MySQL. Однако это полезно лишь в случае сервис-ориентированной архитектуры приложения, поскольку монолитное приложение нельзя горизонтально масштабировать, а следовательно, в большинстве случаев для такого проекта можно обойтись обычной реляционной СУБД, такой как MySQL или PostgreSQL.