Особенности разработки высоконагруженных систем

С.Н. Амиров

***Аннотация* — Статья раскрывает особенности проектирования высоконагруженных систем. Проанализированы источники последних лет, рассмотрен международный опыт создания информационных систем, сформулировано понятие высокой нагрузки, перечислены наиболее распространенные практики и технологии. На основе ряда источников по данной тематике было выявлено, что конкретные решения при разработке инфраструктуры приложения подбираются индивидуально, в зависимости от особенностей проекта и решаемых задач. В то же время, несмотря на разнообразие технологий, базовые требования к системам, будь то надежность, возможность масштабирования или удобство сопровождения, и применяемые в проектировании архитектурные паттерны универсальны для сферы информационных систем в целом. В статье обобщается новая информация, включая международный опыт в этой сфере, а также делаются выводы по поводу наиболее распространенных практик при проектировании высоконагруженных систем.**

***Ключевые слова* — программное обеспечение, информационные системы, высокая нагрузка, архитектура системы, паттерны проектирования.**

I. Введение

Цель исследования: выявить и сформулировать актуальные технологии и практики при проектировании высоконагруженных информационных систем.

В работе были использованы такие научные *методы*, как сравнительный анализ, синтез, описание, классификация, моделирование и конкретизация.

Задачи: изучить как местные, так и зарубежные успешные практики, исследования и работы по разработке систем, высоконагруженных данными; сформулировать на основе полученной информации понятие «высоконагруженные системы»; выявить требования, проблемы и ограничения, возникающие на начальных этапах проектирования системы и обусловленные решаемыми задачами; описать эффективные решения и подходы при разработке аппаратной и программной части; на основе изученной информации сделать выводы об универсальных практиках проектирования высоконагруженных систем.

Статья отправлена 26 мая 2020.

С.Н. Амиров – магистрант Дагестанского Государственного Университета, г. Махачкала, р. Дагестан, место работы - руководитель Центра цифрового образования при ГБУ ДО РД «Малая академия наук РД» (e-mail: s.amirov90@mail.ru)

.

II. Определение Понятия "Высоконагруженные Системы"

На сегодняшний день существует огромное количество проектов, высоконагруженных данными (data-intensive system). Технологии, реализуемые в подобных проектах, важны во многих прикладных областях: от прогнозирующей аналитики до мониторинга окружающей среды, от электронного правительства до умных городов [1]. Кроме того, научные области (биоинформатика, исследования климата, геология, радиоастрономия, астрофизика и т. д.) также сталкиваются с проблемами, связанными с эффективным манипулированием большими объемами данных, включая сбор, генерацию и распространение сложных, неоднородных данных, помощь в принятии решений, моделирование, прогнозирование, визуализацию результатов и т.д. [2].

В одной из зарубежных статей приводится следующее определение, которое мы примем за основу в текущем исследовании: «Системы, высоконагруженные данными — это системы, которые могут принимать, обрабатывать и генерировать большие объемы данных разной природы и из разных источников с течением времени, организованные с помощью различных технологий и с целью извлечения ценности из данных для различных типов предприятий» [3].

В русскоязычном сегменте для обозначения таких систем чаще используют термин хайлоад. Хайлоад (highload) это нагрузка, с которой не справляется аппаратное обеспечение [4]. Он возникает в тот момент, когда мы достигаем каких-либо технических ограничений: сеть, память, CPU, хранилище. Дополнительно стоит упомянуть сопряженные с этим термином проблемы — недоиспользование оборудования, трудности масштабирования.

На начальных этапах функционирования высоконагруженный сайт или приложение осуществляет соединение пользователей с удаленными ресурсами посредством интернета (доступ и, возможно, изменение данных). Но по мере развития возникает все большая необходимость в технологиях, направленных на оптимизацию как аппаратной, так и программной стороны системы.

Аппаратная сторона (если речь о небольших приложениях) может содержать один веб-сервер и базу данных; при малых объемах данных у нас нет необходимости задумываться об оборудовании как о компоненте нашего приложения, но по мере того, как

проект масштабируется, оборудование становится все более важной частью общего дизайна. При проектировании программной стороны приложения необходимо сформулировать, как мы храним данные, как получаем к ним доступ и модифицируем (бизнес-логика) и как представляем данные пользователям (логика взаимодействия).

Реализация подобного функционала может показаться тривиальной, однако необходимо спроектировать сервис с высокой степенью настройки и оптимизации под конкретные потребности, чтобы заставить этот функционал работать для миллионов пользователей, не расходуя слишком много средств на оборудование.

III. Требования к Системам

Высоконагруженные системы обрабатывают большие объемы данных и формируют за счет этого свою высокую ценность для бизнеса, поэтому сбои и другие проблемы с качеством результата обходятся компаниям крайне дорого. Так, Gartner в своей статье за 2014 год приводит следующую цифру по убыткам компаний: потери крупных онлайн-сервисов в среднем достигают $300 000 в час в случае сбоя сети [5].

Как пишет в своей статье Д. Обухов: «Все проблемы, которые происходят с высокой нагрузкой, сводятся к одному термину — архитектурные проблемы» [4]. Ключевым источником проблем высоконагруженных приложений является объем данных, их сложность и скорость изменения. Поэтому важно, чтобы общая архитектура большого приложения разрабатывалась как с точки зрения программных компонентов, так и аппаратной части, на которой они функционируют. Более того, при проектировании информационной системы необходимо четко понимать, какие установлены сроки поставки, законодательные ограничения, опыт специалистов, участвующих в конструировании, осознавать сопутствующие риски и их приемлемость для компании.

Также необходимо ответить на ряд вопросов: каким образом будет обеспечена правильность и полнота данных, даже в случае возникновения сбоев; как сохранить высокую производительность для пользователей системы; как провести масштабирование в случае роста нагрузки; и т. п.

В изученной нами литературе внимание акцентируется на различных аспектах качества системы при ее проектировании. Так, одни статьи считают ключевыми масштабируемость хранилища данных, низкую задержку запросов/загрузок, достоверность данных, управляемость (система должна быть проста в обслуживании), экономическую эффективность [6]. В то время, как другие источники упоминают надежность, эффективность использования ресурсов, эффективность финансовых затрат, безопасность [1].

При разработке крупномасштабных информационных систем (в первую очередь в сфере веб-технологий) важно учесть ряд принципов:

A. Доступность (availability)

Время безотказной работы прямо коррелирует с репутацией и функционированием многих компаний.

B. Производительность (performance)

Скорость веб-сайта влияет на удовлетворенность пользователей сервисом, а также на рейтинг в поисковых выдачах (что отражается на посещаемости).

C. Надежность (reliability)

Запрос всегда должен возвращать пользователям одни и те же данные, чтоб пользователи были уверены — если какие-то данные записаны/внесены в систему, при последующем извлечении можно рассчитывать на их неизменность и сохранность.

D. Масштабируемость (scalability)

Речь может идти о различных параметрах системы: сколько дополнительного трафика она может обрабатывать, насколько просто увеличить емкость хранилища данных, сколько транзакций может быть обработано сверх текущих возможностей.

E. Управляемость (manageability)

Разработка системы, которая проста в эксплуатации, крайне важна на более поздних этапах развития проекта (простота диагностики и понимания сути проблем, когда они возникают, легкость обновлений или модификаций).

F. Стоимость (cost)

Подразумевает затраты на аппаратное и программное обеспечение. Важно учесть и другие аспекты, необходимые для развертывания и обслуживания системы: количество времени, затрачиваемого разработчиками на сборку системы, объем усилий, необходимых для запуска системы, подготовку, обучение кадров и т. д.

Перечисленные выше принципы могут идти вразрез друг другу, и достижение одной цели будет осуществляться за счет другой. Например, повышение количества серверов (масштабируемость) достигается ценой управляемости (вам придется работать с дополнительными серверами) и стоимостью (расход на приобретение серверов) [6].

По мнению авторов статьи «Architecting Data-Intensive Software Systems» проблемы при проектировании систем с интенсивным использованием данных возникают в следующих сегментах: объемы данных, распространение данных, коррекция данных, использование программного обеспечения с открытым исходным кодом, поиск, обработка и анализ данных, информационное моделирование [2].

О. Хаммель в своем исследовании обозначил ряд проблем обеспечения качества [7]:

- сложность визуализации и объяснения предоставляемых результатов,
- неинтуитивное, «ослабленное» (с целью повышения производительности) понятие согласованности,
- комплексная обработка данных и различные понятия корректности, усложняющие тестируемость подобных систем,
- высокие требования к оборудованию,

используемому для тестирования,
- трудность генерации адекватных, высококачественных данных,
- сложность методов отладки, ведения журнала и отслеживания ошибок,
- обеспечение качества данных.

Рассмотрим часто упоминаемые требования к информационным системам.

Надежность. Термин надежность (reliability) для программного обеспечения включает следующие пункты [8]:
- приложение выполняет ожидаемый пользователем функционал,
- приложение устойчиво к ошибочным действиям пользователя, а также использованию программного обеспечения непредусмотренным путем,
- производительность приложения сохраняется на должном уровне при предполагаемом сценарии использования, ожидаемых нагрузке и объемах данных,
- любая попытка несанкционированного доступа к данным или функционалу, компрометации информации и неправильной эксплуатации предотвращается системой.

Надежность подразумевает возможность системы продолжать работать нормально даже в случае возникновения проблем. Возникающие проблемы называются сбоями, а системы, созданные в расчете на них, называются устойчивыми к сбоям.

Хорошо проработанная документация по управлению сбоями должна включать простое пошаговое руководство по восстановлению системы после практически любого возможного сбоя. Так, автор книги «Building Scalable Web Sites» упоминает следующие вопросы, на которые необходимо располагать ответом: что необходимо делать, когда возникает сбой или заполнение диска, поврежден индекс базы данных, сервер достигает предела пропускной способности ввода-вывода или выходит из строя и т. д. В конечном счете назначение документации состоит в том, чтоб сделать решение всех распространенных проблем тривиальным [9].

Когда речь заходит о крупных центрах обработки и хранения данных, известно, что аппаратные сбои (будь то отключение питания, сбой винчестеров или ОЗУ и т. п.) происходят постоянно. Одним из путей решения проблемы является создание архитектуры без общего доступа. Благодаря такой архитектуре отсутствует центральный сервер, управляющий и координирующий действия других узлов, и, соответственно, каждый узел системы может работать независимо друг от друга. В подобных системах нет единой точки отказа, поэтому они гораздо более устойчивы к сбоям. Другим методом предотвращения сбоев является повышение избыточности (redundancy) отдельных компонентов системы, направленное на снижение частоты сбоев (резервное электропитание, RAID — избыточный массив дисков и т. п.). Когда один из компонентов выходит из строя, запасной компонент берет на себя его функционал. Подобным образом нельзя полностью избежать сбоя, однако вариант вполне приемлем в большинстве случаев, т. к. есть возможность восстановить систему из резервной копии в короткие сроки [6].

В зависимости от типа оборудования и того, как спроектирована ваша система, резервные части могут быть "холодными", "теплыми" или "горячими":

1) Холодный резерв (cold spare) — в случае, если один из компонентов сервера выходит из строя, нам необходимо взять запасной, подключить все оборудование к нему, продублировать конфигурацию и включить.
2) Теплый резерв (warm spare) — это часть аппаратного обеспечения, в которой изначально все настроено для использования, и специалисту нужно просто включить его (физически или программно), чтобы ввести в эксплуатацию в рамках системы.
3) Горячий резерв (hot spare, наиболее предпочтительный режим) — когда один из компонентов выходит из строя, другие используемые одновременно с ним компоненты автоматически заменяют его, направляя весь трафик на себя. Обнаружение нерабочего компонента и переключение происходят без какого-либо вмешательства пользователя.

Если говорить о глобальном резервировании, то все правила взаимодействия между серверами распространяются и на дата-центры – у нас должен быть запас прочности (емкость, вычислительные мощности и т. п.), чтоб продолжить работать при потере одного дата-центра без ощутимого урона качеству предоставляемых услуг.

Для корректной обработки ошибок избыточность должна распространяться и на данные, и на предоставляемые сервисы. Например, если сервер хранит только одну копию файла, его отключение приводит к ограничению доступа к файлу. Именно поэтому практикуется создание нескольких избыточных копий одних и тех же данных либо сервисов.

Программные сбои, порождаемые ошибкой в системе, в силу проблемности предотвращения, а также влияния на несколько узлов сразу, вызывают гораздо больше отказов системы, чем аппаратные. По мнению многих специалистов обеспечение качества в процессе разработки ПО для высоконагруженных приложений все еще находится в зачаточном состоянии [1].

Частыми являются сбои, связанные с предельной нагрузкой одного из ресурсов системы (пространства в хранилище, CPU, оперативной памяти и т. п.), сбои в работе сервиса, обуславливающего функционал системы (задержка или прекращение ответов и т.п.), программные ошибки, возникающие при «плохих» вводных данных, а также небольшие сбои в одном компоненте, которые провоцируют серию последовательных сбоев по цепи (каскадные сбои). Случаются перебои, обусловленные качеством предоставляемых провайдерами услуг — провайдеры на своих серверах запускают устаревшие сервисы,

настроенные для конкретного типа хранилища, формата данных и используемой политики управления. Для преодоления возникающего разрыва в подобных случаях в архитектуре информационной системы необходима реализация промежуточного ПО, которое, с одной стороны, занимается доставкой данных и метаданных, соответствующих общему в рамках системы API, а с другой взаимодействует с существующими внутренними серверами и приложениями [2].

Для обеспечения надежности системы рекомендуется воспользоваться следующими подходами:

- Отдалить те части системы, которые влияют на работоспособность системы, от частей, наиболее подверженным человеческим ошибкам.
- Реализовывать все формы тестирования, будь то модульное тестирование, комплексное тестирование системы, ручные тесты.
- Предоставить инструменты по восстановлению системы в случае сбоя в кратчайшие сроки, чтобы свести к минимуму последствия.
- Внедрить систему метрик, мониторинга и логирования как инструментов диагностики ошибок и причин сбоев.

Также всё чаще можно встретить следующую рекомендацию: выстроить систему подготовки, обучения персонала, а также внедрения лучших практик управления.

Масштабируемость. Рост нагрузки является одной из наиболее распространенных причин падения эффективности, например, в случае если количество активных пользователей выросло на порядок. Под масштабируемостью (scalability) подразумевается способность системы выдерживать растущую нагрузку.

Объем дисков в современных серверных установках, как правило, измеряется в сотнях ГБайт – единицах ТБайт, и масштабирование такой системы данных в десять раз обойдется сравнительно недорого. Однако по мере роста потребностей переход на сотни ТБайт не только значительно увеличивает расходы на оборудование, но и усложняет сопровождение и управление системой с позиции архитектуры.

Расходование ресурсов на масштабирование до того, как в этом возникла необходимость, как правило, является неоправданным с точки зрения бизнеса; однако предусмотренные заранее в архитектуре системы решения могут существенно сэкономить ресурсы и время на более поздних этапах [6].

По мнению авторов «Building Scalable Web Sites» у любой масштабируемой системы должны быть учтены три характеристики [9]:

- готовность к росту хранимых данных,
- готовность к росту нагрузки/запросов,
- ремонтопригодность.

Для определения того, к чему приведет рост нагрузки, следует акцентировать внимание на двух моментах:

1) Каким образом увеличение параметра нагрузки повлияет на производительность системы в условиях неизменных значений ее ресурсов (скорость сети, CPU, оперативная память)?
2) В случае роста нагрузки каким образом стоит изменить ресурсы системы для сохранения ее производительности на текущем уровне?

Простым для масштабирования элементом в высоконагруженных системах являются сети в силу того, что сетевые технологии и протоколы изначально хорошо адаптированы под масштабирование. При масштабировании особенно важной характеристикой сети является объем данных, который мы можем передать прежде, чем достигнем предела емкости; он зависит от таких параметров, как тип трафика, скорость устройств, скорость канального уровня, есть ли на пути между системами какое-либо сетевое оборудование и т. п.

В случае с быстрорастущими системами распространено явление, когда каждое возрастание нагрузки на порядок вынуждает пересматривать архитектуру. Более того, по словам Кола Хэндерсона, автора книги «Building Scalable Web Sites», при масштабировании рано или поздно будет переработана каждая функция сервиса [9].

Практикуется два подхода в масштабировании аппаратной части: вертикальное и горизонтальное. В первом случае машина заменяется на более производительную, в то время как во втором возросшая нагрузка решается посредством добавления в систему новых машин.

Вертикальное масштабирование подразумевает увеличение ресурсов на одном сервере (добавление жестких дисков либо замена на более емкие, увеличение ОЗУ, установка более мощного ЦП), т. е. в итоге отдельный сервер способен обрабатывать большее количество данных самостоятельно.

При горизонтальном масштабировании, в то время как аппаратное обеспечение будет линейно расти, в производительности подобного роста может не наблюдаться, т. к. программное обеспечение (ПО) вынуждено обмениваться сообщениями со всеми узлами системы, объединять результаты вычислений всех узлов кластера. По этой причине рост производительности может быть нелинейным; более того, производительность при добавлении нового сервера может уменьшаться. В какой-то момент мы сталкиваемся с порогом, после которого каждое следующее добавление узлов одного ранга не будет приводить к росту. В подобных случаях стоит внимательно отслеживать все ситуации нелинейного роста, и, оценив перспективы с позиции затрат и выгод, начать вертикальное масштабирование, заменяя текущие узлы на более мощные. Подобное сочетание плюсов горизонтального и вертикального масштабирования считается наиболее эффективным, именно поэтому сегодня наиболее популярен комбинированный подход.

Хотя на начальном этапе приобретение оборудования кажется затратным шагом, по мере развития проекта стоимость разработки ПО становится несравненно больше. Именно поэтому в современных системах стремятся развивать архитектуру так, чтоб масштабирование практически не требовало разработки ПО – приобретение, установка и настройка

оборудования обходится намного дешевле.

Приложение масштабируется так эффективно, как самый слабый его компонент, поэтому нельзя забывать выявлять подобные места, проектировать с учетом будущего масштабирования и отслеживать всё происходящее с оборудованием посредством качественно реализованной системы мониторинга компонентов системы.

Удобство сопровождения (maintainability). Стоимость программного обеспечения состоит по большей части из затрат не на изначальную разработку, а на текущее сопровождение — исправление ошибок, поддержание работоспособности его подсистем, расследование отказов, адаптацию к новым платформам, модификацию под новые сценарии использования и добавление новых возможностей. Стоит уделить особое внимание нескольким принципам проектирования программных систем [8]:

- Удобство эксплуатации. Облегчает обслуживающему персоналу поддержание беспрепятственной работы системы.
- Простота. Облегчает понимание системы новыми инженерами путем максимально возможного ее упрощения (один из лучших инструментов — абстракция, она позволяет скрыть большую часть подробностей реализации за аккуратным и понятным фасадом).
- Возможность развития. Упрощает разработчикам внесение в будущем изменений в систему, ее адаптацию для непредвиденных сценариев использования при смене требований.

IV. Оборудование, Аппаратная Сторона Системы

На начальных этапах аппаратная сторона приложения составляет большую часть всех расходов на запуск системы. Затраты на разработку программной стороны (зарплаты разработчикам, специалистам, обслуживающим систему и т. д.) значительно выше (как было указано выше), но за оборудование приходится платить в самом начале и сразу всей суммой. Речь идет не о каких-то конкретных типах процессоров, марках оборудования или архитектурах, а, в целом, о совокупности составляющих элементов (аппаратное и программное обеспечение) приложения. Оборудование может быть представлено одним типом вычислительных машин с одинаковыми поставленными операционными системами (ОС) и наборами утилит, однако чаще всего это парк из различных классов оборудования с различными ОС.

При хранении больших объемов данных используются специализированные подходы [2]:

- Организация хранения файлов на основе метаданных (разделение по каким-то определенным атрибутам).
- Репликация серверов управления файлами для разделения общего хранилища данных и их доступа, поиска, обработки и т. п.
- Отбор технологий распространения данных с учетом всех ограничений и преимуществ.
- Дополнительные возникающие задачи поиска и обработки данных.

Перечислим наиболее распространенные методы размещения оборудования приложений.

Совместно используемое оборудование (shared hardware)

Как правило, арендуется у крупных провайдеров (хостинги и др.), ресурсы серверов эксплуатируются совместно с другими пользователями. У пользователя нет возможности конфигурации сервера под себя и установки дополнительных модулей. Подобные платформы подходят для этапа разработки, размещения прототипа и раннего запуска приложения.

Выделенное оборудование (dedicated hardware)

Оборудование, необходимое для запуска приложения, вы арендуете у поставщика. Он со своей стороны владеет и занимается обслуживанием данного оборудования, поэтому вам не придется менять выходящие из строя сервера, диски хранения данных, стойки и т. п. – только вход в систему и обмен данными по защищенному каналу. Выделенное оборудование по возможностям разнится от неуправляемого (нам предоставляется только логин пользователя) до полностью управляемого нами (получаем удаленный доступ к консоли для последующей самостоятельной настройки). Плюсом является отсутствие необходимости наличия системных администраторов в штате.

Совместно размещенное оборудование (co-located hardware)

Вы предоставляете свое оборудование и обслуживание системы, а поставщик совместного размещения обеспечивает вас площадкой, пропускной способностью сети и электричеством. Предоставляются такие функции, как мониторинг сети, перезагрузка сервера в случае сбоя и т. п. Некоторые поставщики предоставляют услуги по мониторингу работы серверов, диагностике и оповещению в случае сбоев, однако этот функционал может отсутствовать, либо за него будет браться отдельная плата.

Самостоятельный хостинг (self-hosting)

В случае, если ваш проект разросся до нескольких тысяч серверов, возникает смысл создания собственных дата-центров. Однако надо быть готовым к тому, что вам придется создать собственную площадку, выстроить в соответствии с потребностями серверную архитектуру, нанять DevOps-специалистов, а также обслуживающий персонал, обеспечить системой пожарной безопасности, резервными подключениями к электросети и интернету, источником бесперебойного питания (UPS) и т. д.

Узким местом веб-приложений является пропускная способность базы данных, что вызвано низкой скоростью ввода и вывода информации на диске. Полученные объемы данных после нескольких этапов обработки необходимо переносить в итоговую область хранения данных, что требует высокой пропускной способности сети, соответствующей по скорости генерации данных, и задействование таких протоколов передачи данных, которые в полной мере используют эту полосу пропускания. Проблема отставания

пропускной способности сети от скорости генерации данных и роста объемов хранимой информации актуальна и сегодня, поэтому одной из задач является разработка технологий, способствующих максимальной скорости передачи информации, будь то создание нескольких параллельных потоков данных, декомпозиция передаваемых файлов, настройка буфера и т.п.

Автор «Building Scalable Web Sites» Кол Хэндерсон пишет: "Планирование мощностей — довольно точная наука" [9]. Его рекомендации по поводу начального дизайна аппаратной части приложения выглядят так:

- Приобретать распространенное, представленное на рынке оборудование.
- Использовать готовые сборки ОС и проверенные дистрибутивы.
- Использовать по возможности готовое программное обеспечение.

V. Паттерны Проектирования, Программная Сторона Системы

При разработке высоконагруженного данными приложения (DIA) необходимо реализовать возможности, обеспечивающие регулярно требующуюся функциональность. Например, стандартными считаются такие возможности: хранение данных для дальнейшего использования их текущим либо внешним приложением — база данных; поиск и фильтрация данных по ключевым словам и другими способами — поисковый индекс; сохранение наиболее часто проводимых операций для ускорения возврата результата — кэш; переработка больших пакетов данных — пакетная обработка; отправка сообщения между процессами для асинхронной обработки — потоковая обработка.

Вопрос «на каком языке лучше писать highload-проект» ошибочен в силу того, что отказоустойчивость, высокие нагрузки и т. п. завязаны не на технологиях, а архитектуре, и от того, как будет спроектирована архитектура, зависит, как будет работать проект, выдержит ли высокую нагрузку, будет ли возможность масштабировать и какими путями. Соответственно, нет определенных языков, технологий или баз данных, которые лучше или хуже подходят для высоконагруженного данными проекта.

При проектировании следует воспользоваться следующим алгоритмом: получить подробную информацию о своих данных; построить по полученной информации схему использования и движения данных; подобрать архитектурный паттерн, разработать архитектуру под каждое конкретное использование; подобрать наиболее уместные инструменты и технологии под каждую выбранную архитектуру.

Если стоит задача получить подробную информацию о данных, то необходимо ответить на следующие вопросы: какие есть данные в проекте, как они связаны между собой, какое количество данных каждого типа, какие минимальные, максимальные и средние размеры у данных каждого типа, какое количество актуальных данных, каким образом определяется актуальность данных, какими темпами растет объем данных, какова частота чтения данных [10].

С выбором методологий всё неоднозначно – классическая методология водопад (waterfall) достаточно хорошо работает на больших, монолитных проектах, однако зачастую разработка приложений выигрывает от иного подхода, с быстрыми короткими итерациями (напр., Scrum, Agile). Каждый сделанный вами шаг, каждое принятое решение должно быть легко обратимым в случае, если мы обнаружим что ошиблись.

Если мы говорим о готовых программных решениях, то использование ПО с открытым исходным кодом (open-source software) является распространенной практикой среди высоконагруженных проектов. Значимая часть крупнейших IT-компаний были разработчиками либо активно участвовали в развитии open source библиотек и фреймворков (Facebook, Google, Amazon, Microsoft и т. д.). Подобное программное обеспечение подразумевает под собой повторное использование разработанных компонентов в процессе реализации сервисов.

В случае с подбором архитектурных паттернов, необходимо выбрать наиболее актуальные из списка рекомендуемых к использованию: сервисно-ориентированная архитектура; отложенные вычисления; асинхронная обработка; конвейерная обработка; использование "толстого" клиента; функциональное разделение; шардинг (sharding); виртуальные шарды; центральный диспетчер; репликация; партиционирование (partitioning); кластеризация (clustering); денормализация (denormalization); введение избыточности; параллельное выполнение. При выборе паттернов необходимо ясно понимать плюсы и минусы каждого из них, границы применения, а также ознакомиться с уже реализованными примерами использования.

Начальным этапом в разработке большинства веб-проектов является разделение структуры на три части, каждая из которых отвечает за обработку или выполнение определенного класса задач: фронтенд (front-end, быстрая обработка легких данных), бэкенд (back-end, вычисление) и хранение данных.

Первым звеном выступает фронтенд, который предназначен для обработки легких данных, как правило, статического контента сайта, освобождая от этого массивный бэкенд. Ключевым моментом во фронтенде является то, какое количество ресурсов тратится на обработку одного запроса, поэтому для этого звена используются легковесные сервера, например, Nginx.

Второе звено это бэкенд. На нем, как правило выполняются вычисления, реализуется бизнес-логика, поэтому легковесные запросы обрабатывать тяжелым, высокопроизводительным бэкендом неэффективно.

Третье звено – хранение информации. В простейшем виде представляет из себя базу данных, но скорее это хранилище данных на серверах, расположенных в дата-центрах. В любом случае оно является ключевым элементом нашей системы – данные, которые лежат в основе остальных сервисов приложения.

Для эффективной работы системы с хранилищем данных ее разбивают на несколько уровней, и там, где раньше был большой монолитный блок кода, теперь присутствуют такие элементы, как бизнес-логика, логика взаимодействия и разметка/представление. Т. к. каждому из элементов системы приходится взаимодействовать с другими, разработчикам необходимо проделать дополнительную работу по реализации интерфейсов на границе взаимодействия уровней.

Бизнес-логика (business logic)

Располагается непосредственно над хранилищем, и в ней заключается ключевой функционал приложения, то, чем оно отличается от других подобных сервисов. Уровень бизнес-логики — это единственный путь получения доступа к хранилищу данных, т. к. поведение нашей системы напрямую определяется правилами, регулирующими доступ к данным, и особенностями их манипуляции (как хранятся и обрабатываются).

Логика взаимодействия (interaction logic)

Следующий после бизнес-логики уровень, описывает то, каким образом данные изменяются, что и как отображается пользователю. В зависимости от производимых действий меняется отражаемая информация, однако основной функционал приложения остается без изменений.

Разметка/представление (интерфейс) (markup, interface)

Располагается на самом верхнем уровне, предоставляет пользователям возможности для доступа на более нижние уровни и демонстрирует, что на них находится.

В вебе для разработки высоконагруженной системы часто используются такие технологии как распараллеливание, предобработка, конвейер, отложенные вычисления, поиск, кэширование различных типов, прокси-сервер, индекс, балансировка, очередь. Рассмотрим ниже некоторые из них.

Поиск (search) является одной из неотъемлемых частей систем, высоконагруженных данными. Он принимает запросы и возвращает в качестве ответа множество результатов, которые впоследствии могут быть использованы для предоставления конечным пользователям, обработки данных для последующего анализа и т.п. Поисковый сервис представляет из себя распределенную структуру, которая хранит каталоги, наполненные информацией о хранимых данных (индексы и метаданные) для их каталогизации. Архитектура поисковых сервисов включает в себя обработчик запросов, индексатор, диспетчер схемы (отвечает за предоставление метаданных для запросов), преобразователь (адаптирует данные для добавления в каталог, извлекает метаданные и передает в индексатор), брокер хранилища (отвечает за сохранение информации).

Кэш (cache) напоминает кратковременную память – работает намного быстрее, чем источник данных, хранит информацию, к которой недавно обращались, однако ограничен в объеме, и при покупке оборудования цена за единицу информации у кэша обходится дороже, чем у стандартных носителей (диски и т. п.). Как правило, кэш размещается на ближайшем к представлению, интерфейсу уровне, т. к. предназначен для быстрого возврата информации без обращения нижним уровням системы. Если данных в кэше нет, то искомая информация запрашиваются непосредственно с хранилища данных. Благодаря своей эффективности кэш активно применяется на каждом уровне системы, будь то аппаратное обеспечение, операционные системы, браузеры или приложения.

Существуют такие формы кэша, как глобальный (global cache) и распределенный кэш (distributed cache). В случае глобального кэша узлы системы все запросы обращают к одному пространству кэша, и каждый из этих узлов запрашивает кэш тем же образом, что и локальный; в силу разнообразности запросов глобальный кэш очень легко переполнить. Технология распределенного кэша подразумевает, что каждый узел хранит в себе часть кэшированных данных, и благодаря хэшированию (технология вычисления функции для быстрого перехода к информации) система запросов очень быстро определяет, где именно могут быть расположены данные для последующей проверки их наличия [9].

Прокси-сервер (proxy server) представляет из себя промежуточный элемент, получающий запросы от клиентов и передающий их на внутренние сервера, назначением которого является регистрация, фильтрация и модификация запросов, в том числе, шифрование. Прокси-сервера являются удобным инструментом по оптимизации приходящего трафика, координации запросов, идущих от нескольких серверов, возможности объединения одинаковых или подобных запросов в один с последующим возвратом результата соответствующим пользователям. Помимо этого, есть возможность объединять запросы по близости их хранения непосредственно на диске, что тоже значительно снижает нагрузку.

Индекс (index) является инструментом для оптимизации скорости доступа к данным. Более того, индексы можно использовать для создания нескольких разных представлений одной информации в зависимости от поставленных задач, избегая создания дополнительных копий данных (напр., формирование различных фильтров данных, сортировок по различным признакам). В частности, данная возможность востребована у аналитиков, т. к. оптимизация данных для анализа отличается от оптимизации при стандартных методах чтения/записи.

Балансировщик нагрузки (load balancer) используется в информационных системах для равномерного распределения нагрузки между массивом узлов, обрабатывающих запросы. Т. е. ключевой смысл балансировщика нагрузки это возможность масштабировать высоконагруженную систему (что позволит обслуживать большее количество запросов) простым добавлением новых узлов с помощью обработки всех соединений и их перенаправления на один из узлов. Реализация балансировщика может быть как на аппаратном, так и на программном уровне. Также балансировщики являются инструментом, позволяющим

справляться с нарушением работоспособности узла – если какой-то узел перегружен либо перестал отвечать на запросы, его можно удалить из пула запросов, перенаправив нагрузку на другие узлы. На сегодняшний день используется ряд алгоритмов реализации обслуживания запросов, например, циклический перебор, случайный выбор, выбор узла на основе какого-либо критерия (загрузка памяти, процессора и т. п.).

Очередь (queue) является популярным инструментом и применяется в тех ситуациях, когда обработка запроса (чтение, запись, вычисление и т. п.) занимает продолжительное время, предоставляя асинхронность выполнения: поступившая задача добавляется в конец очереди, в то время как воркеры (workers) проходятся по сформированному списку задач. Всё, что нужно клиенту в данной ситуации, это подтверждение получения системой запроса. Также данная технология предоставляет защиту от сбоев – в случае нарушения работы сервера очередь может повторно предоставить невыполненные запросы.

Перечисленные выше технологии имеют множество готовых программных реализаций, обладающих своими особенностями, но решающих одни задачи. Выбор в пользу того или иного решения зависит конкретных ситуаций и потребностей, возникающих при работе над системой.

VI. Заключение

По результатам исследования можно сделать следующие выводы: сфера высоконагруженных систем находится в стадии активного развития, и на рынке можно встретить огромное количество различных технологий и инструментов, способствующих решению конкретных задач. Поэтому нельзя говорить о каких-то универсальных наборах технологий, их стоит подбирать для своего проекта в каждой ситуации индивидуально. В то же время еще на стадии планирования специалистами применяются достаточно четко проработанные, стандартные практики, способствующие избежать проблем с надежностью системы и сложностями в масштабировании. Речь идет и о первоначальном сборе информации о требованиях к информационной системе, и о последовательности дальнейших действий, и классических архитектурных паттернах, использование которых в значительной степени и определяет выбор технологий для реализации этих паттернов. Только изучив этот базис, а также проанализировав текущую ситуацию в реализуемом проекте, можно переходить к выбору технологий и построению структуры приложения.

Библиография

[1] G. Casale, D. Ardagna, M. Artac, F. Barbier, E. D. Nitto, A. Henry, G. Iuhasz, C. Joubert, J. Merseguer, V. I. Munteanu, “Dice: quality-driven development of data-intensive cloud applications.” in Proceedings of the Seventh International Workshop on Modeling in Software Engineering. IEEE Press, 2015, pp. 78–83.

[2] C. A. Mattmann, D. J. Crichton, A. F. Hart, C. Goodale, J. S. Hughes, S. Kelly, L. Cinquini, T. H. Painter, J. Lazio, D. Waliser, “Architecting data-intensive software systems” in Handbook of Data Intensive Computing, Springer, 2011, pp. 25–57.

[3] M. Felderer, B. Russo, F. Auer, “On Testing Data-Intensive Software Systems”, 2019.

[4] Д. Обухов, Блог Highload Junior [Электронный ресурс], “Highload для начинающих”. URL: http://highload.guide/blog/highload-for-beginners.html

[5] A. Lerner, “The Cost of Downtime”, [Электронный ресурс], Gartner, 2014. URL: https://blogs.gartner.com/andrew-lerner/2014/07/16/the-cost-of-downtime/

[6] A. Brown, G. Wilson, “Scalable Web Architecture and Distributed Systems”, in The Architecture of Open Source Applications, Volume II, Mountain View, 2012, pp. 1-23.

[7] O. Hummel, H. Eichelberger, A. Giloj, D. Werle, K. Schmid, “A collection of software engineering challenges for big data system development”, in 2018 44th Euromicro Conferenceon Software Engineering and Advanced Applications (SEAA), IEEE, 2018, pp. 362–369.

[8] M. Kleppmann, “Designing Data-Intensive Applications”, 1st Edition. Sebastopol: O’Reilly Media, Inc, 2017.

[9] C. Henderson, “Building Scalable Web Sites”, 1st Edition, Sebastopol, O’Reilly Media, Inc, 2006.

[10] О. Бунин, “Вступительная статья”, [Электронный ресурс], Блог Highload Junior. URL: http://highload.guide/blog/what_data_we_have.html)

[11] О. Бунин, “Общая логика масштабирования”, [Электронный ресурс], Блог Highload Junior. URL: http://highload.guide/blog/scaling-logic.html

[12] B. Burns, “Designing Distributed Systems: Patterns and Paradigms for Scalable, Reliable Services”, 2018.

[13] M. de Bayser, L. G. Azevedo, R. Cerqueira, “The case for devops in scientific applications”, in 2015 IFIP/IEEE International Symposium on Integrated Network Management (IM), 2015, pp. 1398–1404.

[14] R.T. Fielding, “Architectural Styles and the Design of Network-based Software Architectures”, dissertation, 2000.

[15] D. Ford, “Availability in Globally Distributed Storage Systems” in 9th USENIX Symposium on Operating Systems Design and Implementation (OSDI), 2010.

[16] R. T. Kouzes, G. A. Anderson, S. T. Elbert, I. Gorton, D. K. Gracio, “The Changing Paradigm of Data-Intensive Computing” in Computer, vol.42, no.1, 2009, pp. 26–34.

[17] C. Lynch, “Big data: How do your data grow?”, Nature, 2008, pp. 28–29.

[18] C. Mattmann, “Software Connectors for Highly Distributed and Voluminous Data-Intensive Systems”, Ph.D. dissertation, University of Southern California, 2006.

[19] S. McConnell, “Software Estimation: Demystifying the Black Art”, 1st edition, Microsoft Pres, 2006.

[20] A. Perez "High-load services: Lecture course”, 2017.

[21] I. C. Richard, “How Complex Systems Fail”, Chicago, Cognitive technologies Laboratory, University of Chicago, 2000.

[22] T. White, “Hadoop: The Definitive Guide”, 2nd Edition, O’Reilly, 2010.

С.Н. Амиров – г. Махачкала, р. Дагестан, РФ, 12.07.1990. Магистрант факультета Математики и Компьютерных наук по направлению фундаментальная информатика и информационные технологии, Дагестанский Государственный Университет, г. Махачкала, р. Дагестан, РФ, год окончания 2020. Окончил бакалавриат по направлению психология образования, Дагестанский Государственный Педагогический Университет, 2018.

Работает Руководителем центра цифрового образования при ГБУ ДО РД "Малая академия наук РД". До этого работал веб-разработчиком, наставником курсов программирования, методистом технических курсов в Инжиниринговом Центре Микроспутниковых Компетенций при Дагестанском Государственном Техническом Университете. По окончании бакалавриата была защищена дипломная работа "Влияние информационных технологий на когнитивные процессы у детей". В данный момент готовится к защите магистерская диссертация "Упрощенные формы естественного языка и приемы их формализации".

Features of the Development of High Load Data Systems

Saeed N. Amirov

***Abstract* —The article develops the design features of data-intensive systems. Sources of recent years are analyzed, international experience in creating information systems is considered, the concept of high load is formulated and common practices and technologies are listed. Based on a number of sources on this topic, it has been revealed that specific solutions for developing the application infrastructure are selected individually, depending on the features of the project and the tasks to be solved. At the same time, the basic requirements are universal for the sphere of information systems: the reliability, the scalability or ease of maintenance and the architectural patterns used in the design. The article summarizes new information, including international experience in this area, and draws conclusions about the most common practices in the design of high-load systems.**

***Keywords* — software, data systems, high load, system architecture, design patterns.**

REFERENCES

[1] G. Casale, D. Ardagna, M. Artac, F. Barbier, E. D. Nitto, A. Henry, G. Iuhasz, C. Joubert, J. Merseguer, V. I. Munteanu, “Dice: quality-driven development of data-intensive cloud applications.” in Proceedings of the Seventh International Workshop on Modeling in Software Engineering. IEEE Press, 2015, pp. 78–83.

[2] C. A. Mattmann, D. J. Crichton, A. F. Hart, C. Goodale, J. S. Hughes, S. Kelly, L. Cinquini, T. H. Painter, J. Lazio, D. Waliser, “Architecting data-intensive software systems” in Handbook of Data Intensive Computing, Springer, 2011, pp. 25–57.

[3] M. Felderer, B. Russo, F. Auer, “On Testing Data-Intensive Software Systems”, 2019.

[4] D. Obuhov, Blog Highload Junior [Jelektronnyj resurs], “Highload dlja nachinajushhih”. URL: http://highload.guide/blog/highload-for-beginners.html

[5] A. Lerner, “The Cost of Downtime”, [Jelektronnyj resurs], Gartner, 2014. URL: https://blogs.gartner.com/andrew-lerner/2014/07/16/the-cost-of-downtime/

[6] A. Brown, G. Wilson, “Scalable Web Architecture and Distributed Systems”, in The Architecture of Open Source Applications, Volume II, Mountain View, 2012, pp. 1-23.

[7] O. Hummel, H. Eichelberger, A. Giloj, D. Werle, K. Schmid, “A collection of software engineering challenges for big data system development”, in 2018 44th Euromicro Conferenceon Software Engineering and Advanced Applications (SEAA), IEEE, 2018, pp. 362–369.

[8] M. Kleppmann, “Designing Data-Intensive Applications”, 1st Edition. Sebastopol: O’Reilly Media, Inc, 2017.

[9] S. Henderson, “Building Scalable Web Sites”, 1st Edition, Sebastopol, O’Reilly Media, Inc, 2006.

[10] O. Bunin, “Vstupitel'naja stat'ja”, [Jelektronnyj resurs], Blog Highload Junior. URL: http://highload.guide/blog/what_data_we_have.html)

[11] O. Bunin, “Obshhaja logika masshtabirovanija”, [Jelektronnyj resurs], Blog Highload Junior. URL: http://highload.guide/blog/scaling-logic.html

[12] B. Burns, “Designing Distributed Systems: Patterns and Paradigms for Scalable, Reliable Services”, 2018.

[13] M. de Bayser, L. G. Azevedo, R. Cerqueira, “The case for devops in scientific applications”, in 2015 IFIP/IEEE International Symposium on Integrated Network Management (IM), 2015, pp. 1398–1404.

[14] R.T. Fielding, “Architectural Styles and the Design of Network-based Software Architectures”, dissertation, 2000.

[15] D. Ford, “Availability in Globally Distributed Storage Systems” in 9th USENIX Symposium on Operating Systems Design and Implementation (OSDI), 2010.

[16] R. T. Kouzes, G. A. Anderson, S. T. Elbert, I. Gorton, D. K. Gracio, “The Changing Paradigm of Data-Intensive Computing” in Computer, vol.42, no.1, 2009, pp. 26–34.

[17] C. Lynch, “Big data: How do your data grow?”, Nature, 2008, pp. 28–29.

[18] C. Mattmann, “Software Connectors for Highly Distributed and Voluminous Data-Intensive Systems”, Ph.D. dissertation, University of Southern California, 2006.

[19] S. McConnell, “Software Estimation: Demystifying the Black Art”, 1st edition, Microsoft Pres, 2006.

[20] A. Perez "High-load services: Lecture course”, 2017.

[21] I. C. Richard, “How Complex Systems Fail”, Chicago, Cognitive technologies Laboratory, University of Chicago, 2000.

[22] T. White, “Hadoop: The Definitive Guide”, 2nd Edition, O’Reilly, 2010.