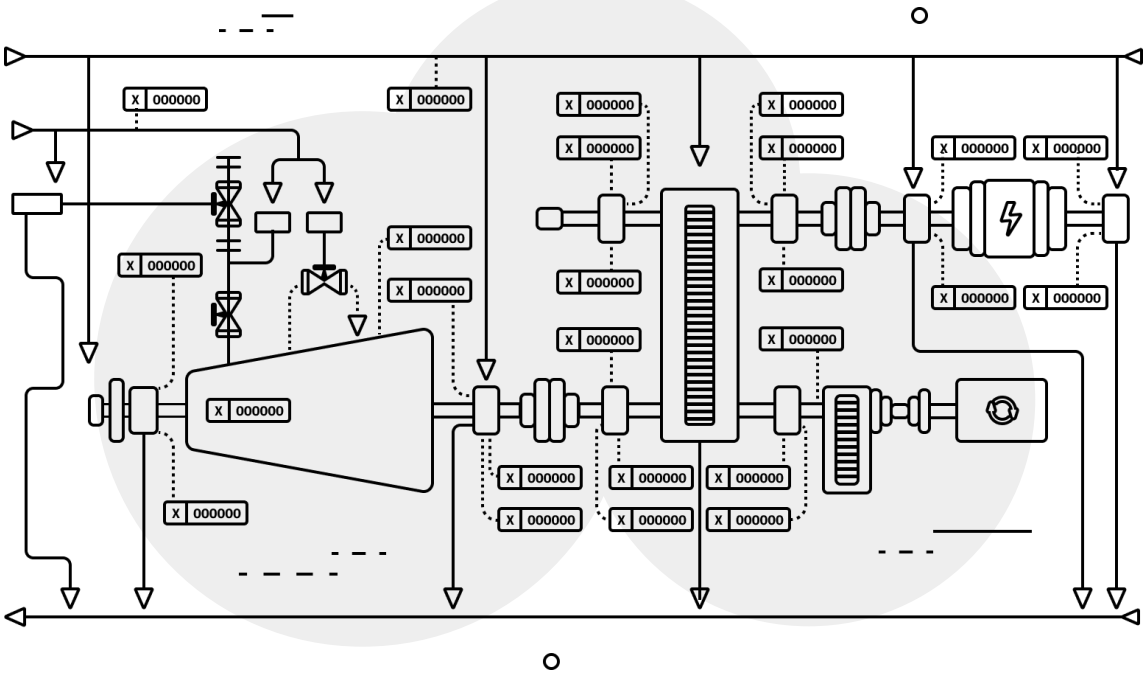
Архитектура высоконагруженных систем

Второе издание

Системы сбора информации
Распределенные системы управления
Системы реального времени



4CIO

АРПП
Отечественный софт

Москва, 2024

Вадим Подольный

В. П. Подольный

Архитектура высоконагруженных систем

Системы сбора информации

Распределенные системы управления

Системы реального времени

Второе издание

Москва

2024

УДК 004.6

ББК 16.35

П44 Подольный В.

Архитектура высоконагруженных распределенных систем

Издание второе. – М.: ООО «САМ Полиграфист» («OneBook»), 2024. –261[1] стр, ил.

Книга дает представление о том, как проектируют высоконагруженные распределенные системы управления. Книга адресована всем, кто хочет разобраться, как устроены и создаются такие системы.

УДК 004.6

ББК 16.35

ISBN 978-5-00227-164-1

[1] 246 стр. для электронной версии

Оглавление

Благодарности

Автор выражает искреннюю признательность коллегам за помощь в подготовке книги.

Прежде всего хотелось бы поблагодарить Клуб топ-менеджеров «Клуб директоров ИТ — 4 СИО.Ру», редакционную коллегию учебников по цифровой трансформации «4CIO» и «4CDTO», персонально Сергея Кирюшина, Феликса Карасева, Евгения Борисова, Владимира Определенова, Дмитрия Северова, Алексея Кравченко, Евгения Колесникова за полученный опыт разработки глав учебника 4CIO/4CDTO, мотивацию на создание собственной книги, конструктивную критику и отзывы.

Автор благодарит за активное участие в обсуждении книги, критику и ценные замечания профильных специалистов Антона Жбанкова, эксперта в области создания аппаратной инфраструктуры высоконагруженных систем, и Сергея Шелудько, эксперта в области разработки распределенных высоконагруженных программных систем.

Автор выражает благодарность опытному старшему товарищу, автору книг[2] по цифровизации АО «Концерн Росэнергоатом» «Цифровая трансформация», «Цифровой двойник», «Центры обработки данных» и «Искусственный интеллект» Александру Прохорову за замечания, которые позволили сделать книгу более качественной.

[2] Digital Atom — книги, #A1

Об авторе

Вадим Подольный с 2000 года работает в области разработки программного обеспечения промышленных распределенных высоконагруженных систем управления. За более чем 20 лет Вадим принял участие во многих крупных инфраструктурных проектах, связанных с обеспечением технологической независимости ключевых отраслей промышленности. В должности главного конструктора он руководил разработкой Российской программной платформы системы верхнего уровня АСУ ТП АЭС класса DCS/SCADA (ПО Распределенных Технологий Автоматизации Лицензированное, ПОРТАЛ) в ГК «Росатом». В настоящее время на базе ПОРТАЛ создано и успешно введено в эксплуатацию множество систем верхнего уровня АСУ ТП на современных энергоблоках АЭС с реакторами ВВЭР-1000, ВВЭР-1200 и БН-800. Вадим был главным конструктором и руководителем разработки специальной операционной системы «Заря», обеспечивающей выполнение задач сопряжения автоматизированных систем управления Вооруженными силами РФ и разрабатываемой в Центральном научно-исследовательском институте экономики, информатики и систем управления (ЦНИИ ЭИСУ). Как эксперт Вадим участвовал в разработке и обеспечении кибербезопасности крупных распределенных автоматизированных систем ГК «Ростех», ОАО «РЖД» и ряда других коммерческих структур. В настоящее время Вадим руководит несколькими проектами в области разработки промышленных распределенных высоконагруженных систем управления, обработки и хранения данных. Он является членом совета «Клуба Топ-менеджеров 4CIO», постоянным спикером многих отраслевых конференций, таких как «HighLoad++», «ИБ КВО», «4CIO» и др.

Вадим окончил факультет технической физики Национального исследовательского ядерного университета МИФИ (Московский инженерно-физический институт) по специальности «Ядерные реакторы и энергетические установки», аспирантуру Всероссийского научно-исследовательского института по эксплуатации АЭС (ВНИИАЭС), получил дополнительное профессиональное образование в области автоматизированных систем управления технологическими процессами, разработки программного обеспечения и индустриальной кибербезопасности.

Связаться с Вадимом: vadim@podolny.ru (email), @vpp01(tg)

О чем эта книга

Идея создания книги появилась после долгого обсуждения с техническими заказчиками подходов к разработке высоконагруженных распределенных систем. Всегда возникал один и тот же вопрос: «Как будем делать эту систему?», — а за ним один и тот же ответ: «Да вот так и будем!» Заказчик уходил думать, пытаясь «сколхозить» решение собственными силами, а когда в очередной раз не получалось, возвращался и говорил: «Вот еще требования добавились, как делать-то будем?». Да все так же! И это могло повторяться снова и снова достаточно долго. Бывало так: приходишь, а там уже другой человек, и все начинается сначала. Мне это надоело, и я решил, что вместо объяснений буду вручать эту книгу. Конечно, это шутка, но в каждой шутке есть доля правды.

Эта книга не претендует на звание универсального свода знаний о высоконагруженной обработке данных или создании систем реального времени. Она лишь отражает некоторый опыт в этой области. Мой опыт в основном касается создания распределенных систем управления промышленными критическими информационными системами. В таких системах присутствуют сотни тысяч источников изменений данных и их потребителей. Сценарии управления зависят от характера и интенсивности этих изменений. Возможно, экспертам в области корпоративных ИТ-систем используемая терминология покажется несколько непривычной, но она появилась из-за сильно отличающихся от ИТ и отчасти завышенных требований к промышленным системам.

Возникает вопрос: как обуздать этот хаос? Нужен *ordo ab chao*[3]. А что, если из хаоса достаточно создать порядок, но не полный? Как вы обычно просите своего ребенка, чтобы привел в порядок свою комнату? Каковы метрики требуемого результата? С какого момента беспорядок можно считать в большей степени порядком, чем беспорядком? Так происходит и с данными

[3] Порядок из хаоса

в нагруженных системах. Чтобы с ними можно было работать, данные должны быть целостными. А какими должны быть метрики целостности (консистентности) данных, позволяющие однозначно сказать, готовы данные к обработке или нет? Ведь если управлять, например, опасным производством, основываясь на неконсистентных (устаревших) данных, может случиться авария.

Книга дает представление о том, как проектируют высоконагруженные распределенные системы управления и адресована всем, кто хочет разобраться, как устроены и создаются такие системы.

Вадим Подольный

Предисловие ко второму изданию

Первое издание успешно нашло своего читателя. Было получено множество положительных отзывов и критических замечаний, которые были учтены в настоящей редакции.

Второе издание книги существенно дополняет первое. Особое внимание уделено вопросу формирования кэша, консистентности, масштабирования распределенных высоконагруженных систем. Добавился раздел о применении разнообразных подходов и практик, используемых в архитектуре распределенных систем. Добавился раздел с некоторыми соображениями относительно кибербезопасности распределенных систем управления.

Таким образом, второе издание более целостно рассматривает проблематику архитектуры высоконагруженных, распределенных систем.

Обращение к читателю

Чаще всего в наше время понятие «высоконагруженные системы» используют применительно к компьютерной инфраструктуре массового предоставления услуг в Интернете 2.0. Причём массово используемые компьютерные системы — это и простейшие сим-карты размером с ноготь мизинца (да, сим-карты — это тоже компьютеры), и мощнейшие компьютерные установки-здания всем известных интернет-гигантов. Только энергопотребление в этих системах отличается в миллиард раз. При этом кажется, что нагрузить до предела возможностей можно любую компьютерную систему. Тем сложнее составить собственное представление о высоконагруженных компьютерных системах и связанных с ними понятиях.

Автор помогает читателю охватить широкий спектр понятий, связанных с высоконагруженными компьютерными системами, выступая как создатель и пользователь систем, которые успешно управляют работой энергоблоков атомных электростанций, обслуживают и защищают инфраструктуру крупнейших предприятий страны. Специалисты оценят емкость изложения тем, на которых автор предлагает сфокусировать внимание. Любознательные читатели в книге объемом всего лишь 250 страниц обнаружат большинство сочетаний терминов, связанных с понятием высоконагруженных компьютерных систем. Надеюсь, что книга подтолкнет читателя к более глубокому знакомству с высоконагруженными компьютерными системами, где и когда это будет ему наиболее интересно.

Дмитрий Станиславович Северов,
экс-заместитель Министра связи и массовых коммуникаций РФ,
член совета Клуба топ-менеджеров
«Клуб директоров ИТ — 4 СИО.Ру»[4]

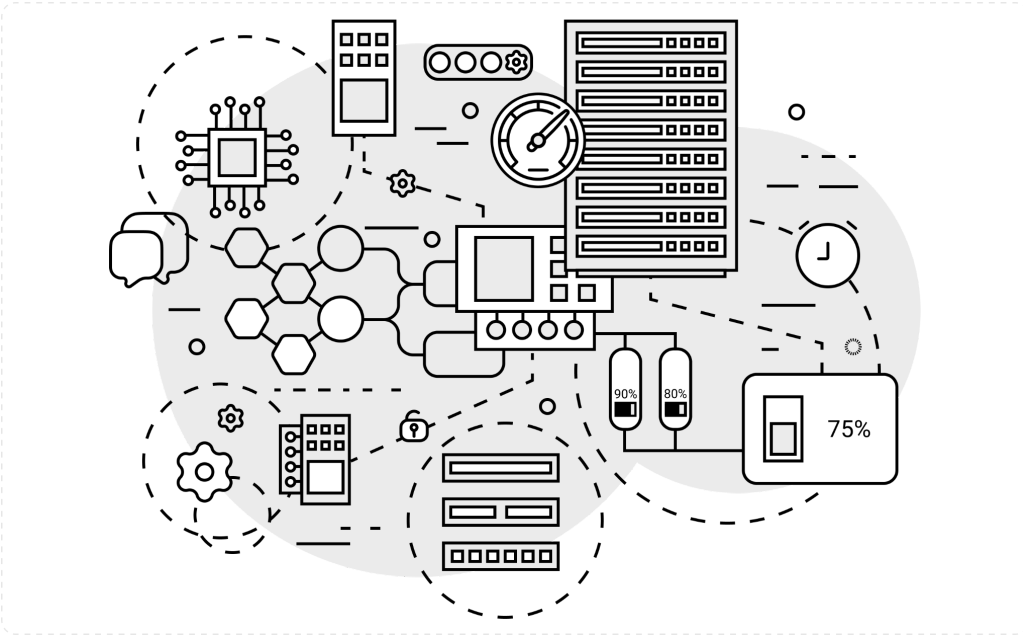
[4] 4CIO, #A2

Приветственное слово

Современные тенденции развития программного обеспечения неразрывно связаны с необходимостью создания эффективной программной архитектуры. Наиболее чувствительной задачей, важной для экономики РФ, является разработка программных продуктов, направленных на решение задач построения распределенных систем управления объектами критической информационной инфраструктуры, таких как крупные промышленные предприятия, банки, сервисы массового обслуживания клиентов. В современных условиях необходимо уметь разрабатывать такие системы силами отечественных разработчиков. Именно перед российскими разработчиками стоит задача создания программных решений для обеспечения технологической независимости страны. Опыт, накопленный автором, в рамках решения указанных задач, консолидирует теоретические основы разработки распределенных высоконагруженных систем управления, которые будут полезны коллегам и начинающим разработчикам.

Ренат Леонидович Лашин,
исполнительный директор, Ассоциации Разработчиков Программных Продуктов (АРПП) «Отечественный софт»[5]

[5] АРПП, #А3



Глава 1. Введение

Что такое высоконагруженная система (ВНС)? Очевидно, ВНС — это система, которая обрабатывает высокие нагрузки. Как обрабатывает? На чем обрабатывает? Насколько эффективно обрабатывает? На все эти вопросы есть ответы. Любая ВНС должна обрабатывать данные эффективно. Эффективно, то есть задействуя все доступные ресурсы аппаратного обеспечения, такие как вычислительная мощность всех ядер процессоров, оперативная память, сетевые устройства. Не должно быть такого, чтобы из-за низкой эффективности (производительности) сетевых карт процессоры оставались загруженными не полностью. Такая ситуация означает, что аппаратное обеспечение подобрано неэффективно с точки зрения решения конкретной задачи обработки данных. Задачи обработки данных бывают самыми разными, их можно классифицировать. Для каждого класса задач обработки существует своя эффективная (оптимальная) аппаратная конфигурация. С экономической точки зрения

аппаратную конфигурацию рекомендуется подбирать таким образом, чтобы при максимальной загрузке все аппаратные компоненты использовались с максимальной эффективностью, а отдельные узлы и компоненты не простаивали.

Современные задачи требуют серьезных вычислительных ресурсов. Какова бы ни была конфигурация оборудования даже самого мощного сервера, который только можно собрать, одного такого устройства никогда не будет достаточно, чтобы решить задачу обработки высоких нагрузок, обусловленных наличием множества источников и потребителей данных. В ВНС обработку данных принято осуществлять на многочисленных объединенных в сеть устройствах, распределяя нагрузки между ними.

Оптимальное разделение нагрузки между распределенными устройствами — это сложная нетривиальная задача, которая напрямую определяет конечную стоимость аппаратного обеспечения. Она является ключевым показателем эффективности бизнеса — в особенности если таких устройств тысячи, десятки тысяч и более.

Существуют задачи, которые можно разделить (распределить) между совокупностью узлов: например, обслуживание сайтов в центрах обработки данных (ЦОД). За клиентом закрепляют определенный набор серверов (узлов), участвующих в его обслуживании. В рамках такого подхода в современных системах принято и рекомендовано использовать гиперконвергентные решения, которые представляют собой совокупность маломощных недорогих серверов. При этом вся работа, связанная с надежностью, выполняется за счет средств виртуализации и связанных средств резервирования. В таких случаях обслуживание клиента осуществляется на закрепленном за ним узле без особого взаимодействия с окружающими узлами. Естественно, средства виртуализации «съедают» часть ресурса оборудования, однако при этом они обеспечивают высокую бизнес-эффективность.

Существуют задачи обработки данных, в которых одни изменения влекут за собой созависимые каскадные изменения. В задачах автоматизированных систем управления технологическими процессами (АСУ ТП) изменения одних переменных оказывают влияние на другие переменные и требуют мгновенной реакции (отклика): например, перерасчета переменных. В таких задачах не всегда можно использовать гиперконвергентные подходы. Рекомендуется использовать принцип «быть ближе к железу», исключив все лишние прослойки, приводящие к потере производительности и эффективности использования аппаратных ресурсов.

ВНС, обеспечивающую выполнение проектных характеристик, таких как выполнение заданного числа операций в заданную единицу времени и время отклика операций, можно считать системой реального времени (СРВ). СРВ в заданный интервал времени обеспечивает выполнение запланированного числа операций. Любую ВНС можно проектировать как СРВ или хотя бы как систему псевдореального времени. Как минимум, в ВНС стоит заложить некоторые полезные архитектурные свойства, присущие СРВ.

СРВ совсем не обязательно должна быть высоконагруженной, однако при решении современных задач большинство СРВ представляют собой подмножество ВНС. Например, умный датчик (устройство IoT) вполне может работать в режиме реального времени (РВ), но при этом не быть ВНС. Технологически такие компоненты обычно являются частью более крупных распределенных ВНС, о которых идет речь в этой книге.

Таковыми являются системы, в которых все компоненты должны работать максимально быстро и слаженно. Каналы связи должны обеспечивать высокую пропускную способность. Физические соединения и кабели должны быть произведены из лучших, эффективных по скорости прохождения импульсов и, следовательно, дорогостоящих материалов. В таких системах необходимо использовать быстродействующие микропроцессоры.

Должно быть обеспечено многократное резервирование компонентов. Использование таких ресурсов оправдано при решении ряда задач: например, таких как управление органами регулирования ядерного реактора и остальных систем управления атомной электростанцией, органами управления летательными аппаратами, системами управления вооружением, медицинскими системами. Такие системы принято относить к СРВ, а в некоторых случаях — к системам жесткого РВ, к компонентам которых предъявляют особенно высокие требования.

Для СРВ важнейшими понятиями являются дедлайн[6], представляющий собой предельную длительность выполнения операции, и латентность[7], представляющую собой время отклика (задержки) операции. Поэтому СРВ и некоторые ВНС принято называть низколатентными или ультра-низколатентными[8] системами.

В области знаний о СРВ и их описаниях часто возникают путаница с терминами и споры об их использовании. С точки зрения РВ существуют системы как критичные по быстродействию, так и не очень. Некоторые эксперты пользуются связкой терминов «системы жесткого РВ» (критичные) и «системы РВ» (не очень критичные). Другие пользуются связкой «системы РВ» (критичные) и «системы псевдо-РВ» (не очень критичные).

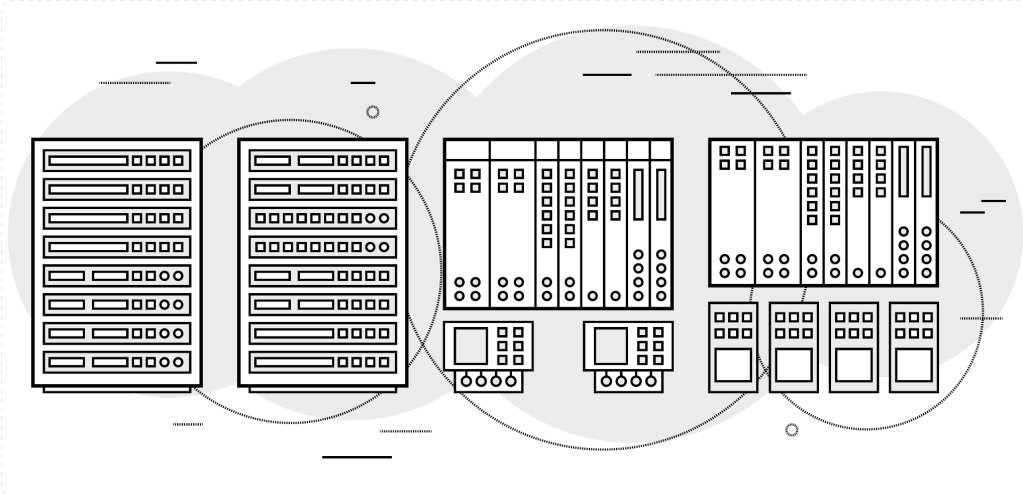
Системы псевдореального времени наиболее широко применяют в условиях, при которых небольшое нарушение заданных дедлайнов является нормой, однако некоторое их превышение может привести к немедленному отказу системы. На практике такие системы создают иначе, нежели СРВ. Это связано с тем, что требования к ним несколько упрощают. Обычно проектируют некую систему, оценивают ее возможности, а затем масштабируют и оптимизируют в узких местах. Такой подход выходит дешевле создания канонической СРВ и тем более систем

[6] Deadline, #1-01
[7] Latency, #1-02 ▸
[8] Low latency, ultra-low latency, #1-03

жесткого РВ, которые проектируют с большей ответственностью, предъявляя более жесткие требования к верификации и валидации.

Системы обычного времени (нормального времени) — это системы, в которые не закладывают требования к режимам работы РВ. Тем не менее такие системы могут быть эффективно и качественно разработаны, они могут обладать исключительными характеристиками производительности. Однако если в такую систему заранее не заложить требования режимов работы РВ, то при высоких нагрузках и перегрузках такая система с высокой вероятностью поведет себя непредсказуемо. Не всегда будет сразу понятно, что с ней не так, где именно возникли проблемы.

Самая большая проблема для лиц, принимающих решения, возникает в тот момент, когда появляется путаница в терминах. Разработчики навязывают систему, критичную к быстродействию, но реальная задача позволяет обойтись и не очень критичной. В таком случае возникает перерасход на разработку, закупку оборудования, пусконаладку, начинаются проблемы, связанные с обслуживанием серьезной, критичной к быстродействию системы. А такая дорогостоящая и «капризная» система нужна далеко не всегда.



Глава 2. Аппаратная архитектура

В ВНС и СРВ важнейшей составляющей является аппаратная инфраструктура. В этой книге не будут подробно описаны, но будут выделены несколько моментов, которые в конечном итоге влияют на надежность, эффективность и стоимость инфраструктуры целевой системы. При проектировании аппаратной инфраструктуры для ВНС РВ следует учитывать несколько факторов.

2.1. Выбор типа железа

При проектировании рассматриваемых систем всегда возникает вопрос выбора типа используемого железа: использовать меньше серверов, но крутых, мощных и дорогих с резервированием компонентов или больше, но дешевых и слабых без резервирования компонентов? На этот вопрос можно ответить, рассмотрев задачу как с точки зрения аппаратного обеспечения (АО), так и в совокупности с программным обеспечением (ПО), потому что именно оно влияет на итоговую эффективность работы железа.

При выборе между «круто и дорого» и «завалить кучей дешевого железа» есть один правильный алгоритм действий: необходимо тщательно оценить технические требования, географию, условия размещения оборудования, доступность

технического персонала на местном рынке труда и т. д. В абсолютном большинстве случаев этот выбор является надуманным. Если пройти все этапы проектирования системы, то либо многообразие вариантов сократится до одного безальтернативного, после чего останется только провести тендер на поставку, либо существенно сократится число вариантов выбора оборудования.

2.2. Место размещения оборудования

Место размещения оборудования почти всегда обусловлено правильно сформулированными при проектировании бизнес-требованиями и техническими требованиями. География распределенных кластеров напрямую зависит от показателей RPO/RTO (разд. 7.2.2) и требований по защите от катастроф (разд. 7.2.3). Выбор между коммерческим ЦОД и локальным размещением на площадке предприятия вытекает из требований к удаленности, безопасности, защищенности и резервированию каналов связи.

Проектирование обычно осуществляться сверху вниз путем декомпозиции от общего к частному. Важно своевременно учитывать возможности самого нижнего уровня, чтобы компенсировать его недостатки на верхнем. При отсутствии подходящих аппаратных решений и/или площадок для размещения оборудования можно снизить требования к нижнему уровню (площадке) за счет использования эффективной программной архитектуры и реализации общего уровня защищенности на более высоких уровнях. Важно помнить, что требования архитектуры целевой системы — это способ реализации основной бизнес-задачи, а не истина в последней инстанции. Всегда существует несколько способов реализации таких требований.

2.3. Оценка конфигурации оборудования при нагрузке

Необходима определенная аккуратность в оценке комплектации оборудования. Например, может так случиться, что при максимальной нагрузке на сетевые устройства процессоры

серверов не будут эффективно загружены. В таких случаях можно оптимизировать соответствующие аппаратные ресурсы, увеличить мощность сетевого оборудования или уменьшить мощность процессоров, при этом даже немного сэкономив на них.

Требования к конфигурации оборудования должны исходить из следующих целевых показателей:

- производительность и загрузка при установленной нагрузке;
- производительность и загрузка при пиковой нагрузке;
- уровень резервирования;
- размеры доменов единичного и двойного отказа;
- стандартизация конфигураций;
- доступность оборудования и конфигурации (комплектующих) на рынке, планы по поддержке производителем;
- доступность квалифицированного персонала на локальном рынке труда.

Зачастую имеет смысл разместить достаточно мощное оборудование с большим запасом по производительности и резервированию на удаленных площадках для максимизации времени безостановочной работы без обслуживания при возникшем сбое. При условии логистической доступности и наличия большого склада ЗИП[9] в крупном центре можно обойтись меньшим уровнем резервирования, исходя из статистики отказов и своевременного пополнения ЗИП.

2.4. Зависимость от вендора или технологии

Необходимо оценить риски зависимости от одного вендора или конкретной технологии. Всегда лучше рассчитывать на то, что и вендора, и технологию в какой-то момент придется заменить. Или сам вендор может выбрать другую технологию для своего будущего оборудования, и ваши системы будут несовместимы. Например, бывает так, что программная инфраструктура

[9] Запасные части, инструменты и принадлежности

использует особую аппаратную технологию. В этой связи можно вспомнить классический пример того, как компания HP перестала поддерживать серверы AlphaServer с архитектурой RISC в начале 2000-х годов. Такие решения продолжают использовать и сейчас на множестве предприятий, и у многих из них не осталось резервов (ЗИП). Системы такого рода продолжают эксплуатировать, потому что разработчики ПО, занимавшиеся уже устаревшей архитектурой, уже на пенсии или недоступны, поэтому портировать системы на современную архитектуру уже некому. Модернизировать такие системы можно только целиком. Представьте себе, например, что это АЭС с несколькими энергоблоками. Такие ситуации сплошь и рядом возникают во всех отраслях и странах мира. Возникает вопрос организации технической и гарантийной поддержки большого разнообразия оборудования и решениями на основе одного вендора.

Крупные сервисные провайдеры и интеграторы самостоятельно осуществляют сборку оптимальных конфигураций оборудования для конкретных инфраструктурных решений, что довольно эффективно. Например, Google, Amazon, Яндекс собирают свои серверные и инфраструктурные решения, самостоятельно контролируя архитектуру, весь процесс производства и жизненный цикл.

2.5. Планирование масштабирования

Необходимо заранее планировать сценарий масштабирования ресурсов (Глава 10). Следует планировать возможность увеличения мощности серверов за счет добавления в них дополнительных процессоров, оперативной памяти, сетевых карт. Кроме того, следует планировать увеличение числа серверов. Бывают ситуации, когда программная инфраструктура рассчитана на некое предельное число узлов и просто не может работать при большем их числе. О таких ограничениях лучше знать заранее. Может случиться так, что условия лицензии на дополнительные узлы не устроят потребителя.

Глава 3. Программная архитектура

3.1. Данные и метаданные

Основная нагрузка в ВНС и СРВ связана с обработкой большого объема (потока) сообщений (данных, событий) из разнообразных внешних источников и внутренних составных компонентов системы. Принято выделять следующие типы событий:

- асинхронные — полностью непредсказуемые события, такие как оповещения, тревоги, команды управления;
- синхронные — предсказуемые события, случающиеся с определенной частотой, такие как сообщения, связанные с обменом и синхронизацией данных;
- изохронные — регулярные события (разновидность асинхронных), случающиеся в определенный промежуток времени, такие как диагностические сообщения о состоянии компонентов системы (разд. 7.1.5).

Не все данные следует обрабатывать в режиме РВ (данные РВ). В СРВ большая часть данных имеет временную привязку. Часть данных стоит отнести к вспомогательным — такие данные нужны для функционирования системы, но пользователю они незаметны. Их не всегда нужно обрабатывать в режиме РВ. Например, не следует обмениваться текстовыми переменными в режиме РВ, если переменные можно задать заранее и обмениваться их идентификаторами. Для облегчения работы с данными вводят дополнительные данные о содержании данных. Например, для фотоизображения вспомогательными данными могут служить дата, координаты, выдержка. Такие данные о данных принято называть метаданными.

Может быть и обратная ситуация, когда часть метаданных разумнее отнести к данным РВ в связи с высокой частотой их использования. Следовательно, такие данные необходимо обрабатывать соответствующим образом, при этом важно соблюдать баланс между объемом обрабатываемых данных РВ и метаданными. Если не оптимизировать формирование объема данных РВ, это может привести к излишним нагрузкам, которые могут повлечь за собой снижение эффективности системы. При этом производительность системы может и не снизиться. Системе не важно какие данные она обрабатывает: важные, вспомогательные или бесполезные, если только она не спроектирована с учетом соответствующих требований.

3.2. Способы обмена данными

В работе современных ВНС принимает участие множество узлов, выполняющих разнообразные задачи. Такие системы называют распределенными вычислительными системами (РВС), распределенными системами обслуживания (РСО), распределенными системами управления (РСУ[10]). В общем случае РВС и РСО чем-то управляют — хотя бы собственными вычислениями, обслуживанием потребителей.

[10] DCS, distributed control system, #3-01

Такое допущение позволяет классифицировать их как РСУ для удобства описания. Основными объектами внимания настоящей книги являются высоконагруженные РСУ (ВРСУ).

Узлы РСУ могут быть загружены по-разному. Они могут представлять собой серверы, рабочие станции, коммутаторы, умные устройства интернета вещей. Узлы с разной степенью активности (непрерывности работы и доступности) могут обмениваться сообщениями между собой в рамках определенной событийной модели.

Чтобы обеспечить эффективное взаимодействие систем, следует выбрать выгодный способ взаимодействия (обмена данными) между узлами. В СРВ не принято терять время на установку TCP-соединений: если таких соединений требуется много, то суммарное время, затраченное на установку множества соединений, может повлиять на эффективность работы всей системы. TCP преимущественно является *unicast*-протоколом, поэтому каждое TCP-соединение будет обработано отдельно.

В РСУ возникает необходимость доставки одинаковых сообщений более чем на один узел одновременно — следовательно, множество (пул) TCP-соединений оказывается неэффективным. В таких случаях рекомендуется использовать протокол UDP в режиме широковещательных сообщений типа *multicast* или режиме *broadcast*. TCP можно использовать в режиме *broadcast*, однако при этом могут возникать некоторые задержки в связи с циркулирующими подтверждающими пакетами, которые создают лишнюю нагрузку. Бывают ситуации, когда выгодно установить пул устойчивых TCP-соединений и осуществить имитацию широковещательных сообщений поверх совокупности таких соединений. Обработка каждого следующего соединения может происходить с некоторой задержкой. В случае использования пула TCP-соединений латентность системы будет выше, чем при использовании протокола UDP (рис. 3-01).

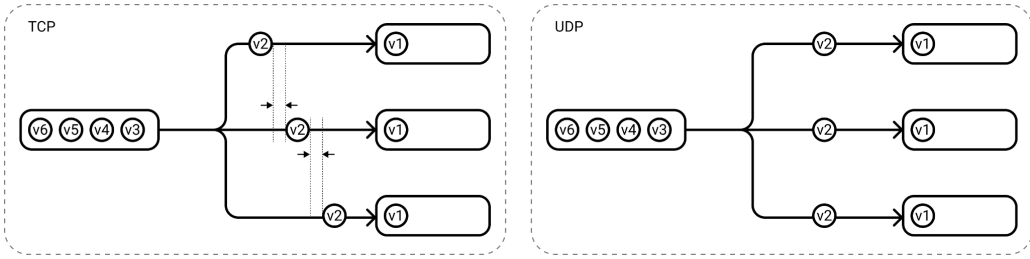


Рис. 3-01. Разница в задержках: UDP, TCP

Иногда возникают неприятные ситуации: новые сообщения могут устаревать в процессе получения адресатами. Следовательно, у части адресатов будут неактуальные данные, что может быть весьма неприятно в контексте задач управления. В СРВ невыгодно применять TCP-соединения в связи с тем, что на базе TCP сложнее реализовать пул асинхронных сообщений. В реальном мире события чаще происходят в асинхронном режиме. Кроме того, использовать UDP-соединения в асинхронном режиме выгоднее. Разработчик может самостоятельно контролировать обмен диагностическими сообщениями, которые могут содержать диагностические метрики. Такие метрики могут описывать показатели качества обмена сообщениями: например, метрики подтверждения доставки сообщений согласно требованиям разработчика, состояние отдельных компонентов и процессов системы (разд. 7.1).

3.3. Транзакции

В ВНС использование транзакций допустимо, а в СРВ — нет. Следует вспомнить принцип ACID[11], описывающий требования к транзакциям:

- *Atomicity* — атомарность гарантирует, что каждая транзакция будет выполнена полностью или не будет выполнена совсем, промежуточные состояния не допускаются;
- *Consistency* — консистентность (согласованность, разд. 6.6) означает, что транзакция, достигающая своего нормального

[11] ACID, #3-02

завершения (EOT — *end of transaction*) и тем самым фиксирующая результаты, сохраняет согласованность данных;

- *Isolation* — изолированность означает, что во время выполнения транзакции параллельные транзакции не должны влиять на результат;
- *Durability* — надежность транзакции означает, что, независимо от проблем на нижних уровнях (например, обесточивание системы или сбои в оборудовании), изменения, возникшие в результате успешно завершенной транзакции, должны остаться сохраненными после возвращения системы в работу. Если пользователь получил от системы подтверждение, что транзакция выполнена, он может быть уверен, что внесенные им изменения не будут утрачены из-за какого-либо сбоя.

В СРВ не принято[12] использовать транзакции. Это связано с тем, что процесс обработки транзакции чаще является синхронным. Когда узлов, участвующих в обработке транзакции, становится больше одного, транзакция становится распределенной (рис. 3-02).

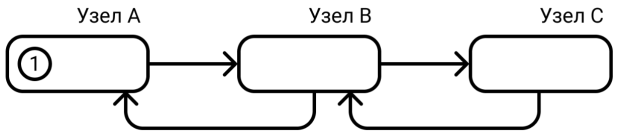


Рис. 3-02. Распределенная транзакция

Таким образом, узел *A* (инициатор транзакции) ждет, пока транзакция 1 будет обслужена всеми участвующими в процессе узлами. Результат выполнения транзакции вернется со всеми вытекающими задержками и блокировками. В таком случае задержки являются лишь частью проблемы. В ВНС транзакции обрабатываются очередями (разд. 5.4.1). При этом крайне выгодно задействовать все ресурсы вычислительного оборудования. Если спроектированное ПО является эффективным, задержка ожидания

[12] В ВНС тоже не принято, но есть решения, такие как 2PC, разд. 11.2.8

выполнения сложных распределенных транзакций может быть скомпенсирована: например, их количеством. Обработка новых транзакций из очередей может быть начата до завершения активной транзакции (разд. 11.1.3). В таком случае распределенным транзакциям придается некое свойство асинхронности обработки, а блокировки ожидания выполнения транзакций влияют только на их инициаторов. Такой механизм асинхронности позволяет эффективно задействовать значительное число распределенных узлов для децентрализации, распределения сколь угодно сложной логики (рис. 3-03) (разд. 11.2.4).

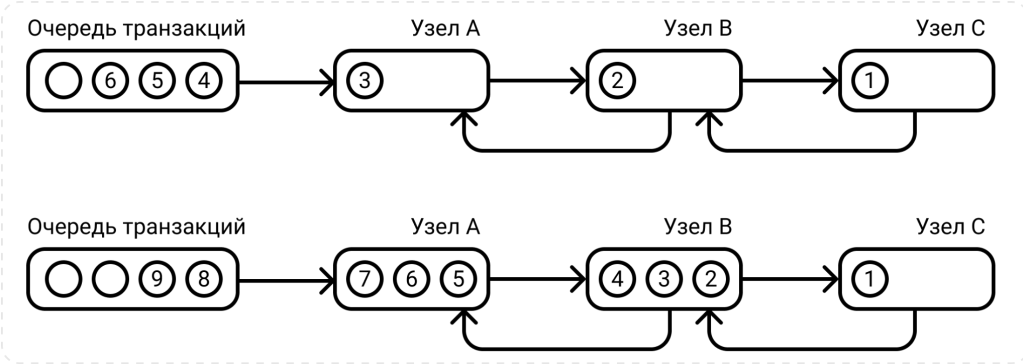


Рис. 3-03. Распределенная транзакция асинхронная

Такой процесс, называемый асинхронной обработкой распределенных транзакций, вместо обеспечения эффективности их обработки может привести к дисбалансу. В связи с тем, что обработка новых транзакций из очереди начинается до завершения уже начатой обработки предыдущих транзакций, среднее время обработки транзакции определенно будет увеличиваться. Такую систему тяжело балансировать из-за неизвестных времени задержки и длительности блокировки, в особенности если целевая система позволяет использовать непредсказуемо сложную логику, задаваемую пользователем, а не жестко реализованную в системе.

3.4. Распределенность

Узлов-инициаторов, обработчиков и верификаторов транзакций в РСУ может быть множество. Такую систему необходимо непрерывно балансировать (Глава 8), чтобы обеспечить слаженность работы всех распределенных узлов (планировать их нагрузку). При условии, что обрабатывающие транзакции узлы не оказывают влияния на вычислительные процессы друг друга, балансировать такую конструкцию достаточно просто. Вычислительные алгоритмы, выполняемые в форме распределенных транзакций, могут использовать данные узлов, которые участвуют в обработке транзакций. Алгоритмически зависимые узлы могут непредсказуемо влиять на общую нагрузку, поэтому балансировать такую систему еще сложнее (рис. 3-04).

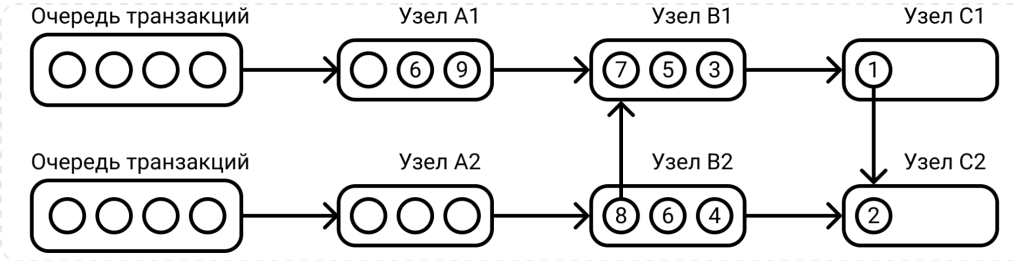


Рис. 3-04. Распределенная транзакция с зависимостью

Этот вопрос хорошо отработан в реляционных системах управления базами данных (СУБД), которые представляют собой специальные инструменты обработки распределенных транзакций[13]. Эффективность вычислений обеспечивают оптимизация структур обрабатываемых данных и оптимизация конфигурации СУБД, однако такая оптимизация практически не приближает обработку данных к режиму РВ. Скорость обработки в таких СУБД существенно ниже, чем в специализированных системах. Такого рода специальные системы становятся все более востребованными на современном рынке. Большинство реляционных СУБД осуществляют хранение данных на жестких дисках. Универсальный табличный (реляционный) формат требует постоянной

[13] Каскадные транзакции, #3-03

дополнительной обработки данных. Следовательно, необходимо разбираться, откуда прочитать, куда записать, как найти, где обработать и куда сохранить индекс. Современные реляционные СУБД (РСУБД) позволяют хранить как данные, так и индексы в оперативной памяти, но сложность обработки реляционных структур накладывает свои ограничения. Транзакции, построенные на базе РСУБД, не отличаются особым быстродействием, которое необходимо СРВ и большинству ВНС. В СРВ стараются отказываться от транзакций в классическом понимании, используя более эффективные механизмы обработки данных и балансировки (планирования) распределенных взаимодействий. Для этого создана масса специализированных решений класса NoSQL, NewSQL — каждое из них имеет свои преимущества и недостатки. Естественно, разнообразные решения, эффективные для реализации целевых требований, принято комбинировать в единую взаимодействующую систему, интегрируя актуальные свойства каждого из них.

3.5. Ограничения распределенных систем

В любой распределенной системе действует принцип теоремы CAP[14], которая гласит, что при реализации распределенных вычислений можно обеспечить не более двух из трех следующих требований:

- согласованность (консистентность) данных (***c****onsistency*) — во всех не отказавших вычислительных узлах в один момент времени данные не противоречат друг другу, то есть являются одинаковыми;
- доступность (***a****vailability*) — любой запрос к не отказавшим узлам возвращает корректный отклик, при этом нет гарантий, что ответы всех узлов системы совпадут;
- устойчивость к разделению (***p****artition tolerance*) — разделение распределенной системы на несколько изолированных сегментов в связи с нестабильностью соединения или его

[14] CAP theorema, #3-04, #3-05 ▸

полным отказом. В этом случае система должна продолжать функционировать, то есть перебои связи не должны приводить к отказу системы.

Типичные РСУ построены на сочетаниях требований CAP-теоремы (рис. 3-05):

- согласованные РСУ со всегда доступными узлами, но неустойчивые к разделению (*CA*);
- устойчивые к разделению РСУ со всегда с доступными узлами, но не согласованные (*AP*);
- согласованные и устойчивые к разделению РСУ, но с не всегда доступными узлами (*CP*).

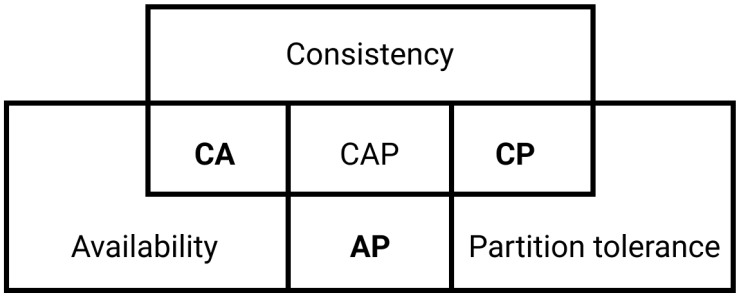


Рис. 3-05. CAP-теорема

В реальных задачах часто требуется, чтобы одновременно работали все три свойства теоремы. Возникает вопрос: как этого достичь, какие допущения и упрощения можно сделать, чтобы приблизиться к выполнению всех трех требований CAP? Опытным профильным архитекторам и разработчикам секрет давно известен: он заключается именно в свойствах консистентности, которые могут быть самыми разнообразными. Консистентность может быть как строгой, так и слабой, как длительной, так и периодической (разд. 6.6).

Например, если рассматривать РСУ в составе АСУ ТП, становится очевидно, что в таких системах недопустимо разделение узлов — может произойти потеря управления важной подсистемой.

Совершенно недопустима несогласованность данных. Например, если операторы будут работать с отличающимися данными, их действия будут несогласованными. Такая ситуация может привести к катастрофе. Естественно, в таких системах допустима потеря связи между узлами. В серьезных АСУ предусмотрена возможность автономной работы отдельных подсистем и даже автономного управления. Оборудование не будет повреждено даже при разделении целостной системы на подсистемы, однако такая АСУ долго не проработает в таком режиме. Существуют специальные регламенты, определяющие, сколько времени можно проработать в режиме автономности подсистем. После исчерпания регламентного времени технологический процесс следует остановить, что зачастую бывает очень дорого и небезопасно. Так, например, остановка ядерного реактора АЭС стоит десятки миллионов долларов.

Рассматриваемые системы, а именно ВНС, СРВ, РС(У), обладают различными свойствами, выполняют самые разнообразные задачи. ВНС часто являются распределенными (ВН РС), существуют ВНС, работающие в режиме РВ (ВНС РВ). Встречаются и РС(У), работающие в режиме РВ (РС РВ). Реже встречаются ВН РС, работающие в режиме РВ (ВН РС РВ): например, масштабные АСУ ТП. Как правило, СРВ является составной частью ВНС, которая, в свою очередь, может быть частью РС(У) (рис. 3-06).

Рис. 3-06. Связь типов систем

Хороший пример противоположности принципов и задач, выполняемых такими системами, можно продемонстрировать,

сравнив, к примеру, системы промышленной автоматизации с банковскими системами. Например, если некий вычислитель вовремя не получит значение температуры реактора от одного или нескольких датчиков или получит его, но не вовремя, может возникнуть серьезная проблема. А вот если клиент банка получит сообщение о зачислении зарплаты немного позже, чем ожидал, или даже совсем его не получит, с ним ничего не случится — он зайдет в приложение и увидит, что деньги пришли. Ну или с банком поругается немного, не катастрофа. Обычно в зарплатные дни банковские системы испытывают высокие нагрузки: банки тщательно обрабатывают все транзакции и контролируют, чтобы все получили сообщения о начислении зарплаты и не звонили с вопросами. Банк не направляет клиенту более одного сообщения о начислении заработной платы, иначе клиент может обрадоваться раньше времени. В промышленной автоматизации, напротив, в АСУ ТП может поступать множество повторных сообщений о значениях температуры и ее изменениях. Если часть сообщений потеряется, ничего страшного не произойдёт. В критически важных компонентах располагают множество избыточных резервных датчиков. Таким образом, подтверждающая информация о доставке значений температур с этих датчиков становится не совсем обязательной. При подаче команд управления, естественно, необходимо подтверждение их выполнения. В таких случаях команда управления обычно запускает параллельные каскадные изменения ряда параметров, которые формируют множество обратных связей. Множество таких связей могут привести к частому обновлению состояния переменных. Изменения в трафике, касающиеся команд, составляют долю процента от общего объема трафика. Следовательно, часть данных в АСУ ТП можно потерять. Вопрос о том, какой объем данных можно потерять, — это вопрос конкретных условий целевой задачи. Такое допущение позволяет «нарушить» CAP-теорему, и «выполнить» все три ее условия одновременно, так как потеря части данных не повлечет за собой утрату консистентности АСУ ТП. В правильно спроектированных РСУ АСУ ТП сообщения приоритизируются по

значимости, при этом допускается потеря наименее значимых данных, а наиболее приоритетные сообщения обрабатываются с подтверждением, почти как транзакции.

3.6. Единый формат данных

Основу быстродействия любой системы составляет отсутствие задержек при лишних преобразованиях форматов данных. Это значит, что в идеальной ситуации все узлы РСУ взаимодействуют между собой в едином формате в рамках формализованного протокола. Чтобы можно было быстро и эффективно обрабатывать, считывать и записывать (перезаписывать) данные, структура их хранения должна быть максимально упрощенной и соответствовать выбранному протоколу.

Данные, участвующие в обмене между узлами, могут быть отделены от метаданных, кроме отдельных специфических случаев. В реальных РСУ редко возникают ситуации, при которых в нагрузке изменяется структура данных: обычно меняются только значения переменных. Как правило, структура данных системы статична. Конфигурация структуры может быть загружена из файла или получена из РСУБД (RDBMS[15]) при старте узлов или в режиме минимальной нагрузки. Конфигурация структуры данных РСУ обычно находится в составе метаданных. Все сложные связи, определяющие структурную принадлежность, иерархичность и связность, могут находиться в метаданных. Все, что можно убрать из трафика обмена данными между узлами в режиме эксплуатации под нагрузкой, рекомендуется определять как метаданные.

3.7. Операции с данными

В высоконагруженных системах, в особенности в системах РВ, операции CRUD[16] принято приоритизировать по значимости. Наиболее частыми (востребованными) являются операции чтения

[15] Relational Database Management System, #3-06
[16] CRUD — операции Create, Read, Update, Delete, #3-07

(*read)* и перезаписи (*update)*. Операции создания сущности (*create*), как правило, сопровождаются выделением динамической памяти, а операции удаления (*delete*) — высвобождением динамической памяти. Данные операции не рекомендуется выполнять в режиме РВ, так как они являются затратными с точки зрения ресурсов. Если *create* — достаточно безопасная операция, то *delete* таковой совсем не является, поэтому в СРВ рекомендуется отказаться от этой операции и вместо физического удаления пометить структуру как удаленную. Физически операцию удаления рекомендуется проводить в режиме сервисного обслуживания или хотя бы в период, когда система достаточное время не будет под нагрузкой. В РСУ рекомендуется избегать любых динамических операций с данными, способных привести к нарушению целостности.

3.8. Распределенные операции с данными

Операции CRUD следует рассматривать с точки зрения распределенности узлов. Какой бы ни была распределенная структура узлов, все распределенные операции должны выполняться с учетом того факта, что узлов более одного. Все распределенные операции должны осуществляться с требуемой эффективностью, с минимальными задержками и, возможно, даже с поддержкой параллелизма.

Если в распределенной системе выполняется операция *create*, значит, должно появиться требуемое число копий сущностей, определяемых конфигурацией целевой системы. Следовательно, при выполнении операции *delete* эти копии должны быть уничтожены. Если осуществляется операция *read*, считывание данных должен выполнять узел, ближайший к источнику запроса. В некоторых случаях при новом запросе ячейка чтения может быть не заполнена, а в случае долгого отсутствия запроса она может перестать обновляться. Возникает дилемма: что выгоднее — поддерживать в актуальном состоянии ячейку данных или обновлять ее при запросе, при определенной частоте запросов или, например, по расписанию?

Операцию *update* в распределенной системе следует рассмотреть подробнее. По сути, *update* отвечает именно за поддержание актуальности той самой ячейки памяти. Однако поддержание ее актуальности может осуществляться с различным приоритетом в зависимости от значимости обрабатываемой переменной. В CPB рекомендуется выделять три вида таких операций:

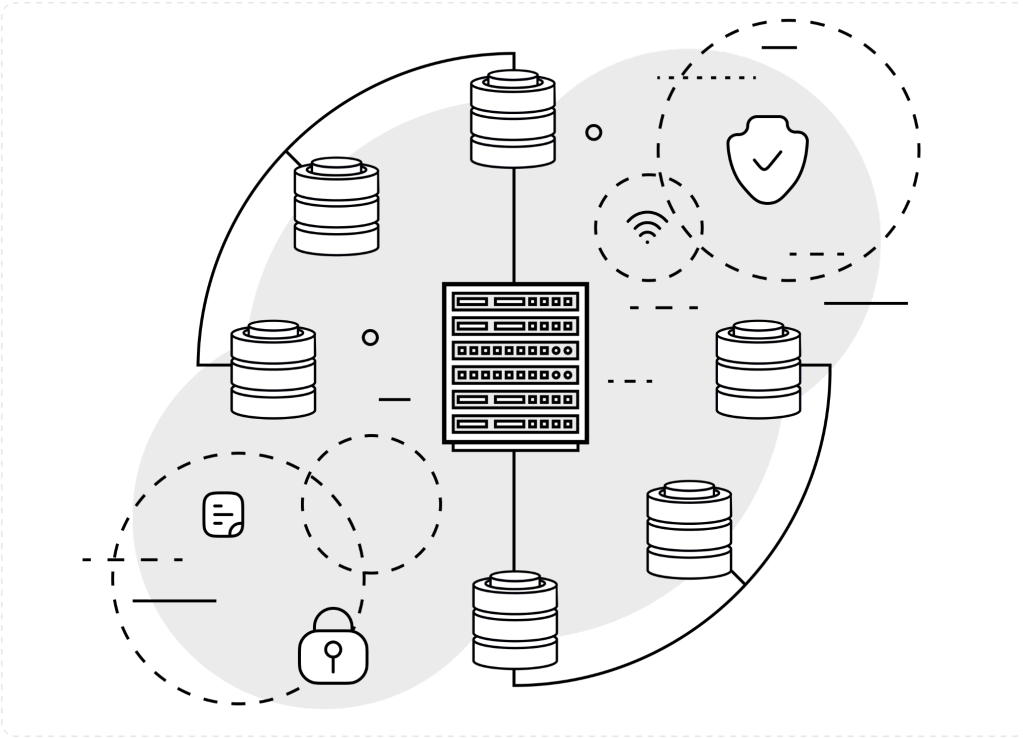
- негарантированный *fast update*[17];
- гарантированный *strict update*;
- гарантированный с подтверждением *hard update*.

Простой негарантированный *update* не гарантирует доставку на целевые узлы и не предполагает подтверждения операции. Это самый быстрый способ распределенной модификации ячейки данных, применяемый для переменных, потеря значений которых может быть скомпенсирована следующим циклом обновления значений.

Гарантированный *update* гарантирует распространение изменений по всем распределенным целевым узлам. Гарантированный *update* с подтверждением обеспечивает самый надежный способ обновления данных с подтверждением. Выполнение такой операции будет происходить медленнее остальных. Ее следует использовать для работы с переменными, отвечающими за критически важные данные: например, когда нужно передать команду управления. В связи с тем, что такие операции являются наиболее ресурсозатратными, рекомендуется свести к минимуму их применение.

Перечисленные выше распределенные операции представляют собой механизмы распространения данных в распределенной системе, обеспечивающие их системную избыточность. Именно такие операции обеспечивают поддержание данных в актуальном состоянии при их изменении, по сути, осуществляя синхронизацию изменений (разд. 6.5).

[17] Best effort #3-08



Глава 4. Принципы распределения данных

4.1. Распределенное хранение данных

В современных системах, сколь бы мощными ни были ресурсы узла (сервера), они в какой-то момент заканчиваются. В серьезных системах данные целесообразно обрабатывать на совокупности распределенных узлов. Это значит, что сначала данные необходимо распределить таким образом, чтобы их можно было удобно и эффективно обработать в распределенном режиме. Каждый такой узел обладает определенным объемом оперативной и постоянной памяти, определенными мощностями процессоров и пропускной способностью сетевых карт. В СРВ не принято задействовать постоянную память (ПЗУ), какими бы производительными ни были устройства.

Распределение данных между узлами может осуществляться в зависимости от их аппаратных конфигураций:

- совокупность узлов с одинаковой конфигурацией;
- совокупность относительно равных по конфигурации узлов;
- совокупность групп, относительно равных по конфигурации узлов;
- совокупность неравных по конфигурации узлов.

Наиболее эффективным для систем является первый вариант — по разным причинам, которые будут рассмотрены далее в разделах 4.3, 4.4. Жизненный цикл любой системы включает периодическое масштабирование (разд. 10.1), модернизацию парка оборудования и его компонентов, поэтому второй вариант встречается чаще. Если речь идет о задачах, в которых встречаются граничные (*edge*[18]) и туманные (*fog*[19]) вычисления, то такие системы будет описывать третий вариант. Если в системе встречается множество разнообразных источников и потребителей, например, устройств интернета вещей (IoT), то такую систему будет описывать четвертый вариант.

Распределение данных между узлами может осуществляться с формированием избыточности. Принцип избыточности широко известен и применяется в различных информационных системах. Например, в системах хранения данных (СХД) для надежности используют разнообразные избыточные массивы независимых дисков (RAID[20]). Существует целый ряд конфигураций таких массивов, в которых используются разнообразные способы организации избыточности. RAID эффективно работают, когда устройства хранения данных (ПЗУ, HDD, SSD), на базе которых они созданы, одинакового размера. В распространенных решениях СХД RAID-массивами управляют один или два узла, работающие в режиме *active-active* или *active-passive*. Решения с более чем двумя управляющими узлами (контроллерами) RAID встречаются редко

[18] Edge compute, #4-01
[19] Fog compute, #4-02
[20] Redundant Array of Independent Disks, #4-03

в связи с их сложностью и дороговизной. Обычно это два сервера, подключенные по резервируемым портам к специальным устройствам (полкам[21]) с дисковыми массивами. Современные RAID-системы, основанные на технологии All Flash на базе NVMe-OF[22], могут быть созданы дезагрегированными. А именно: они могут быть разнесены по нескольким узлам группы серверов, каждый из которых управляет своими сегментами (разд. 4.4) NVMe[23] SSD. Для этого может быть использован специальный протокол RoCE[24]. Такие решения следует считать более близкими к программно-определяемым.

Существуют решения, которые представляют собой программно-определяемые хранилища, устроенные иначе, нежели классические RAID. Такие программно-определяемые хранилища могут быть созданы на базе узлов с разнообразной аппаратной конфигурацией. Конструкции подобного рода принято называть избыточными массивами независимых узлов (RAIN[25]).

В РСУ активно применяют как классические СХД, так и программно-определяемые СХД на базе RAIN. В любом случае до того, как данные попадут на физические носители таких решений, они сначала окажутся в оперативной памяти. Вопрос обмена данными между оперативной памятью и массивами дисков не является задачей РВ и осуществляется в режиме ограничения скоростей, доступных в соответствующих решениях.

РСУ могут использовать для обработки данных оперативную память распределенных узлов как общее пространство. Такая конструкция, предназначенная для формирования общей оперативной памяти узлов, представляет собой особый вид RAIN. Задачи создания общей оперативной памяти лежат в области асимметричного и симметричного мультипроцессирования (AMP[26],

[21] JBOD, JBOF, FBOF, #4-04
[22] NVMe over Fabric, #4-05
[23] Non-Volatile Memory Host Controller Interface Specification, #4-06
[24] RDMA over Converged Ethernet, #4-07
[25] Redundant Array of Independent Nodes, #4-08
[26] Asymmetric multiprocessing

SMP[27]), аналогично задаче превращения распределенных процессоров в единый вычислительный модуль (кластер). AMP и SMP — это технологии объединения находящихся на одной материнской плате процессоров, задачей которых является обеспечение разграничения доступа к оперативной памяти.

Сейчас, в эпоху тотальной виртуализации, задачу объединения ресурсов распределенных узлов формулируют как задачу виртуализации SMP[28]. Задача объединения ресурсов распределенных узлов в единую конструкцию (разд. 4.2), по сути, является задачей, обратной классической виртуализации, в рамках которой решается задача разделения большего ресурса на меньшие с возможностью программного управления их активностью..

Именно такую конструкцию предлагается рассмотреть для управления распределенной обработкой данных в РСУ. Такая конструкция по своей сути является распределенной разделяемой памятью.

4.2. Распределенная разделяемая память

Эффективным механизмом организации распределенного хранения данных является организация распределенной разделяемой памяти (РРП, DSM[29]). РРП представляет собой разновидность технологии программно-определяемой памяти (SDM[30]). РРП образует виртуальное адресное пространство, разделяемое всеми узлами РСУ (программно-определяемое). Узловые сервисы (программы, сценарии) в РСУ получают доступ к данным в РРП. РРП может быть организована статически, когда данные РРП не перемещаются между узлами, или динамически, когда они перемещаются между узлами (разд. 4.2.2).

[27] Symmetric multiprocessing, #4-09
[28] Следует обратить внимание на vSMP от ScaleMP, #4-10
[29] Distributed shared memory, #4-11
[30] Software defined memory

При организации РРП необходимо решить несколько основных вопросов:

- поддержки актуального состояния (целостности, разд. 6.6) структуры распределения данных;
- снижения латентности при доступе и обработке данных;
- обеспечения доступности данных на нескольких узлах одновременно для повышения производительности.

С точки зрения избыточности данных РРП может работать в нескольких вариантах режимов:

- данные на узлах не повторяются;
- данные повторяются на части узлов;
- данные повторяются на всех узлах.

В первом и втором вариантах данные могут быть распределены по узлам в зависимости от задачи системы и требуемой логики обработки данных. В отличие от первого варианта, второй подразумевает (одновременно обеспечивает) резервирование данных (разд. 6.1). Такой способ распределения данных называют сегментированием (разд. 4.4). Третий вариант обеспечивает многократное (кратное числу узлов) резервирование данных в системе. Это подразумевает наибольшие затраты на аппаратную конфигурацию узлов, при этом узлы должны иметь одинаковую конфигурацию. Обычно такие конструкции применяют для построения кластерных систем обработки данных (Глава 9).

Сегментирование РРП реализует прикладные цели:

- обеспечение эффективного резервирования данных, чтобы при отказе узлов данные не пропали;
- обеспечение оптимального использования (минимизации) аппаратных ресурсов при синхронизации сегментов;
- обеспечение большей доступности данных для узлов, осуществляющих их обработку, или иных потребителей.

Существует целый ряд алгоритмов организации РРП, основанных на следующих структурах данных:

- РРП на основе разделяемых страниц (блоков данных);
- РРП на основе разделяемых переменных;
- РРП на основе разделяемых объектов (структур данных).

Иных конструкций организации РРП не существует. Не имеет значения, какая логика взаимодействия задана на базе приведенных возможных архитектур РРП. Как теоретически, так и практически логика распределения данных может быть изолирована от логики целевых задач РСУ. Следовательно, на базе РРП можно одинаково успешно реализовать распределенную (*key-value),* или графовую СУБД, или иную конструкцию (Глава 12).

4.2.1. Алгоритм с центральным узлом или без

Алгоритм с центральным узлом представляет собой самый простой способ организации РРП. Центральный узел обеспечивает координацию всех запросов клиентов РРП (*single-leader*). В этом случае центральный узел является слабым звеном («узким местом»). Чтобы повысить эффективность такого алгоритма, целесообразно разделить функцию центрального узла между несколькими корневыми узлами (*multi-leader*, разд. 5.1) или выбрать способ координации без центрального узла (*leaderless*, разд. 5.3).

Когда данные распределены по нескольким узлам, необходимо уметь определять, на каких именно узлах находятся необходимые потребителю данные. Потребитель может осуществлять итеративный перебор узлов запросами, пока не получит доступ к нужным данным. Клиент может одновременно послать несколько запросов на все узлы, однако это может привести к перегрузке сети. Более эффективным решением является ведение адресации РРП и последующее предоставление доступа потребителям с опорой на адресацию. При наличии нескольких центральных узлов необходимо обеспечить поддержку актуальности адресации на всех узлах, чтобы не нарушить целостность системы.

4.2.2. Миграционный алгоритм

В отличие от алгоритма с центральными узлами, в котором запрос клиента в итоге адресуется узлу, обеспечивающему хранение необходимых клиенту данных, миграционный алгоритм обеспечивает перемещение данных на тот узел, в котором возник соответствующий запрос. Такой алгоритм позволяет взаимодействовать с локальными сервисами (программами, сценариями), интегрированными непосредственно в узлы. Он позволяет избежать перемещения клиентов между узлами. Миграционный алгоритм, подразумевает, что один узел в отдельный момент времени имеет возможность обратиться только к одному элементу данных.

При миграции данных обычно перемещается целый сегмент, содержащий выбранный элемент данных. По этой причине разумнее обеспечить уменьшение размера сегментов данных (разд. 4.4). Постоянное перемещение избыточных данных может привести к перегрузке системы. Такая перегрузка обусловлена пробуксовкой[31].

Пробуксовка возникает в случаях, когда объем данных, требуемых для работы системы, существенно ниже, чем объем данных, используемый при миграции целых сегментов в действительности. То есть, по сути, частый свопинг данных между узлами мешает работе системы, перегружая ее ресурсы. Проблему пробуксовки могут решить аппаратные средства, поддерживающие технологию RDMA[32], которая обеспечивает доступ к необходимым элементам удаленной памяти без необходимости перемещения, однако система может воспринять это как миграцию. В этом случае любой отказ узла повлечет за собой утрату данных.

[31] Trashing, #4-12
[32] Remote Direct Memory Access, #4-13

4.2.3. Алгоритм размножения для чтения

В отличие от миграционного алгоритма, который обеспечивает перемещение сегментов памяти между узлами, алгоритм размножения[33] не перемещает данные от узла к узлу, а размножает (создает реплику) их на требуемый узел. Обычно размножение происходит путем копирования целого сегмента данных (переменной или объекта), как в случае с миграционным алгоритмом. При этом данные становятся доступны с нескольких узлов.

Такой алгоритм можно сравнить с кэшированием (разд. 6.2) на ближайшем к потребителю корневом узле при условии, что такой кэш будет использован только для чтения. В самом простом исполнении модификация данных может быть осуществлена только на исходном узле (рис. 4-01).

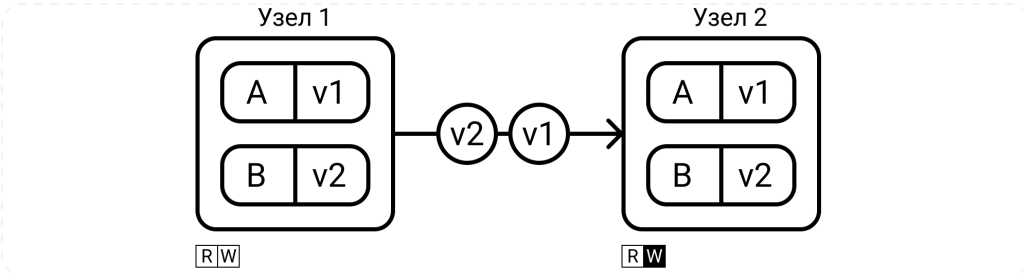


Рис. 4-01. Алгоритм размножения

В более сложном варианте алгоритм должен быть реализован с блокировками записи размножаемых переменных. Модификация может быть осуществлена только исходной переменной на исходном узле. На других узлах допускается исключительно чтение (рис. 4-02). В таком случае серьезной проблемой станет утрата узла источника. На всех оставшихся узлах переменные будут разблокированы для записи. Естественно, можно применить алгоритмы выбора нового лидера, назначить один из доступных узлов главным для модифицируемых переменных и разблокировать переменные на запись.

[33] Replication algorithm

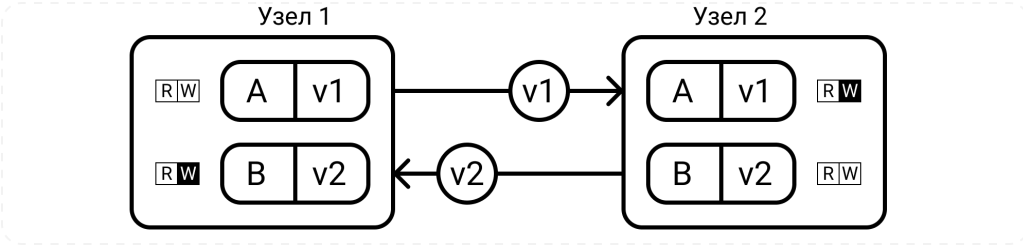


Рис. 4-02. Алгоритм размножения

Производительность системы пропорционально увеличивается за счет возможности организации одновременного доступа в режиме чтения с нескольких узлов. Однако в режиме записи имеют место существенные затраты на коррекцию (перезапись, обновление) всех устаревших блоков данных и обеспечение их консистентности (разд. 6.6). Такой вариант алгоритма называют алгоритмом полного размножения.

4.2.4. Алгоритм полного размножения

Алгоритм полного размножения представляет собой более сложный вариант алгоритма размножения для чтения. Такой алгоритм требует эффективного управления коллизиями. Может случиться так, что на двух узлах в размноженную переменную одновременно будут записаны разные значения. Необходимо будет принять решение, какое изменение считать корректным (рис. 4-03).

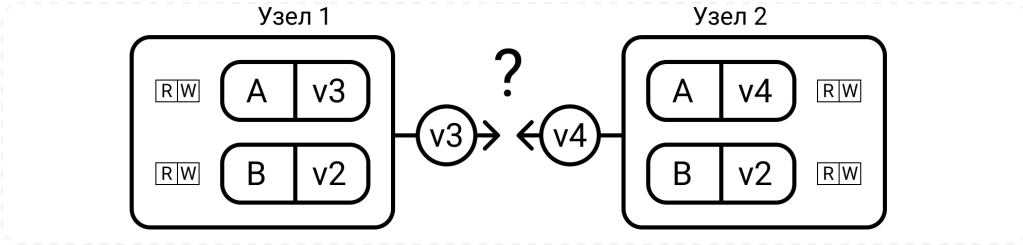


Рис. 4-03. Коллизия

Особенно острым вопрос может стать в случае если такие переменные отвечают за управление критически важными системами. Подобные проблемы решают путем превентивного выбора допустимых источников изменений в проектной логике целевой РСУ.

4.3. Избыточность данных

Избыточность данных (*redundancy*) обусловлена их размножением. Она является важным свойством РСУ, обеспечивающим эффективную (повышенную[34]) доступность данных, скорость их обработки и возможность резервирования.

Избыточные данные необходимы для повышения эффективности работы систем. Например, один узел способен обработать определенное количество запросов в единицу времени, но нужно обрабатывать больше или быстрее. В этом случае можно скопировать часть обрабатываемых данных на второй узел, обеспечить поддержку актуальности этих данных (как-то синхронизировать с первым узлом, разд. 6.5). Далее обработку данных можно частично переключить на второй узел (рис. 4-04).

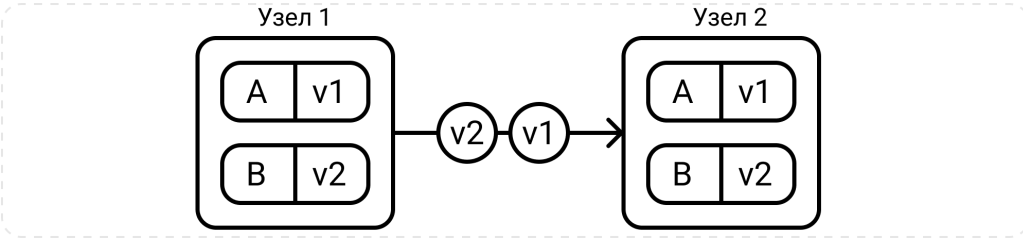


Рис. 4-04. Избыточность данных

Очевидно, что процесс синхронизации данных вносит свой вклад в нагрузку на аппаратные ресурсы системы. Следовательно, необходимо планировать, как эту нагрузку выделить в отдельную сеть, как нагрузить отдельные ядра процессора, — тогда эффективность такой конструкции увеличится. Если узлов больше двух и используются широковещательные механизмы обмена данными, то общая эффективность конструкции становится выше (рис. 4-05).

[34] По сравнению с единичным узлом

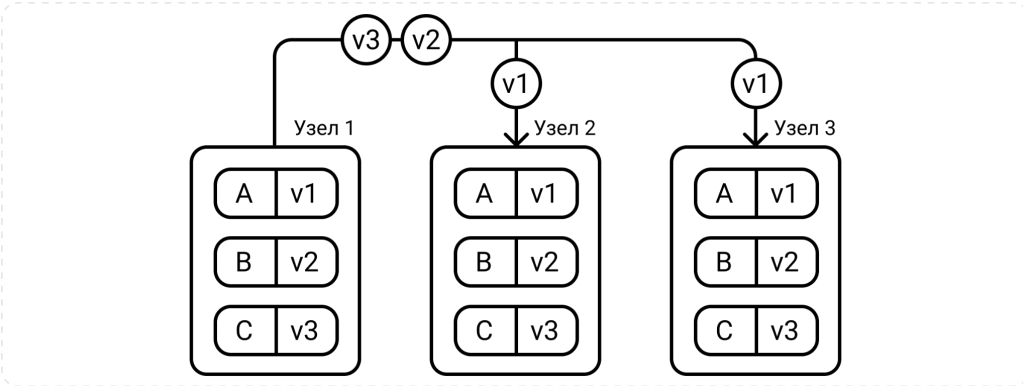


Рис. 4-05. Широковещательный обмен

4.4. Сегментирование данных

Сегментирование (шардинг[35]) путем создания избыточности в РРП является важной оптимизационной задачей. Схемы сегментирования могут быть заданы вручную или автоматически. Вручную можно явно указать узлы, на которых будут представлены необходимые сегменты (переменные, объекты, страницы) (рис. 4-06).

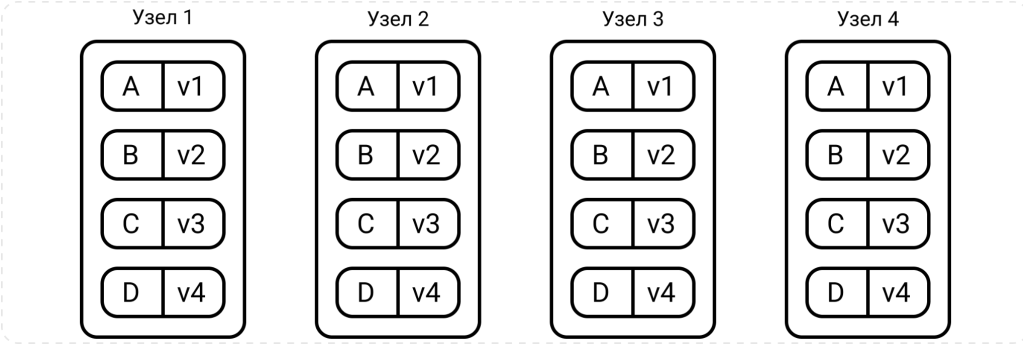


Рис. 4-06. Избыточность данных

Важным параметром сегментирования является фактор избыточности — он определяет, сколько раз сегмент будет представлен (реплицирован, разд. 6.4) в РРП.

Схема сегментирования может быть определена автоматически в соответствии с заданной целевой задачей.

[35] Sharding, #4-14, #4-15 ▸

Схема может динамически изменяться в зависимости от ситуации. Например, автоматическое сегментирование может определять планировщик нагрузки (Глава 8), формируя оптимальную схему распределения сегментов в зависимости от близости узлов к источнику и/или потребителям данных (рис. 4-07).

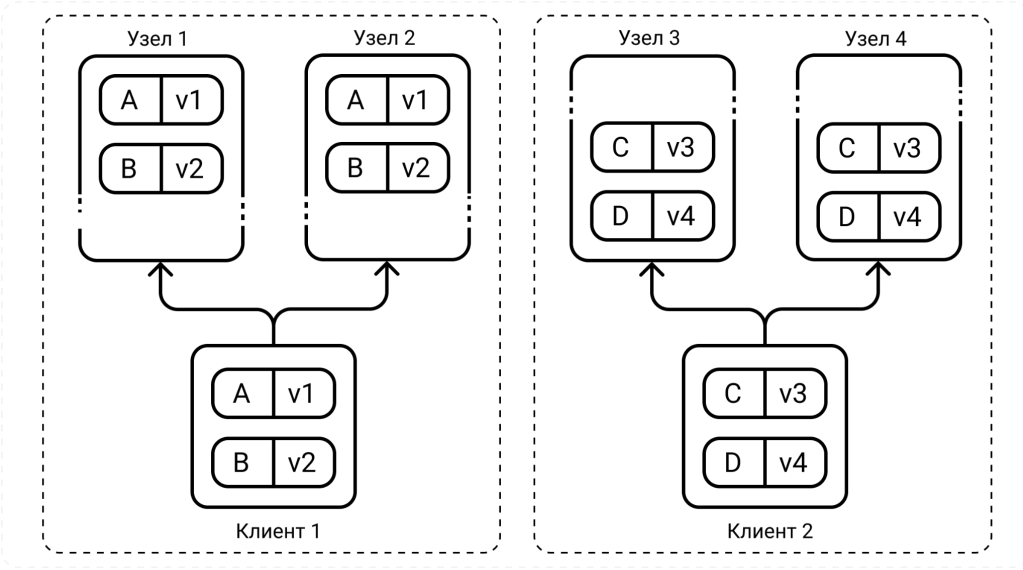


Рис. 4-07. Сегментирование данных

Переменные *v1*, *v2*, необходимые для работы клиенту 1, сегментированы и представлены на узлах 1 и 2. Переменные *v3*, *v4*, необходимые для работы клиенту 2, сегментированы и представлены на узлах 3 и 4. В случае отказа одного узла (например, узла 1), нужные клиенту 1 копии (реплики) сегментов (данных) остаются в единственном экземпляре на узле 2. При отсутствии в системе горячего резерва для дальнейшего обеспечения резервирования данные могут быть скопированы на узел 3 (ближайший), на котором будут находиться данные, необходимые клиенту 1 в случае утраты узла 2 (рис. 4-08).

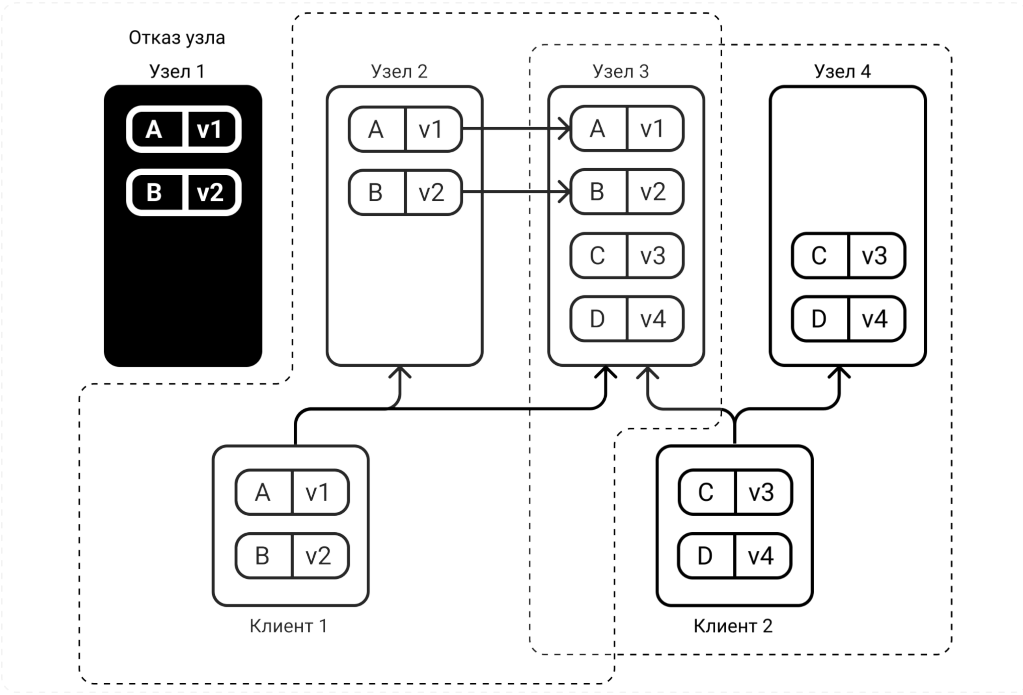
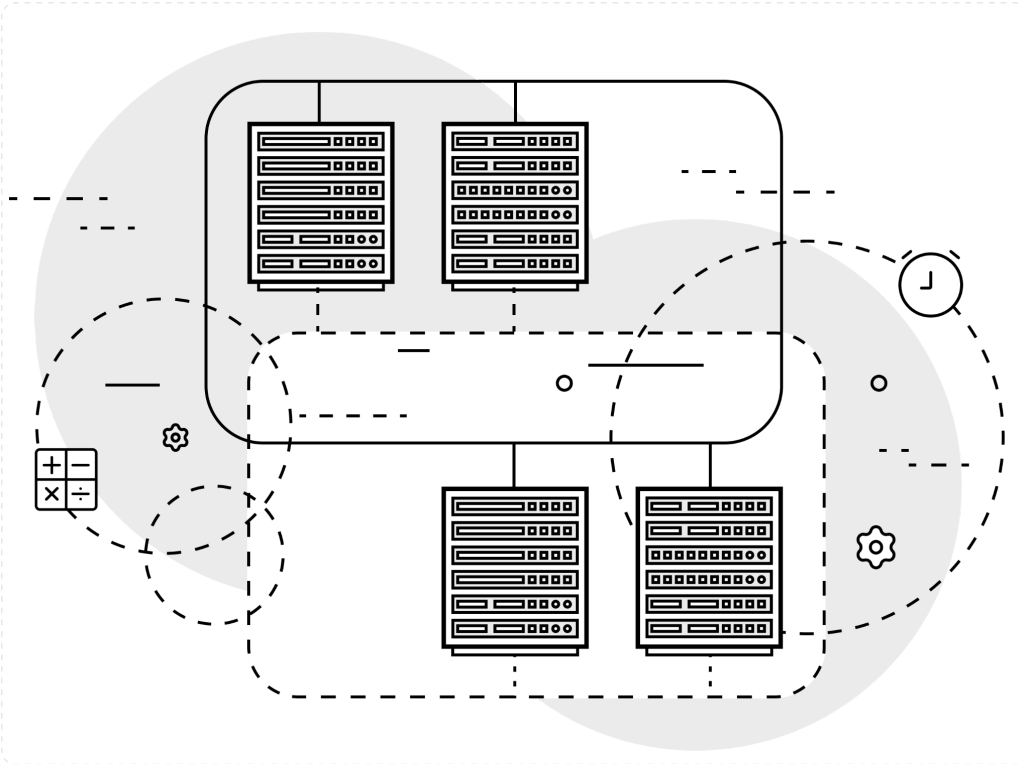


Рис. 4-08. Восстановление избыточности

4.5. Консолидация данных

При сегментировании данных с избыточностью необходимо отслеживать актуальное состояние избыточности сегментов. Может возникнуть такая ситуация, что при утрате узлов будут утрачены и избыточные данные, не говоря уже о данных в единственном экземпляре. При сегментировании данных необходимо оценивать фактор избыточности сегмента. В случае утраты требуемой избыточности необходимо ее восстановить.

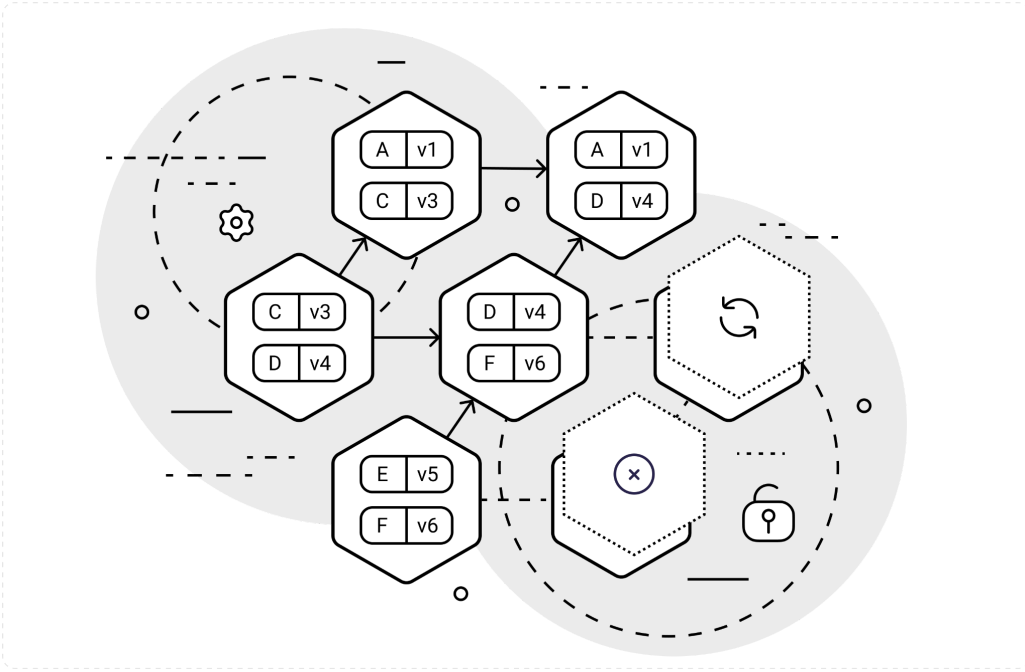
Совокупность всех сегментов данных без избыточности образует консолидацию. Если каждый сегмент имеет единичную избыточность, по сути, возникает полная реплика данных в системе. Таким образом, формируются две консолидации, при этом определяется фактор избыточности консолидации. При утрате узлов необходимо следить за консолидацией и восстанавливать ее при необходимости — в зависимости от фактора консолидации (разд. 6.4).



Глава 5. Распределенная обработка данных

Большинство современных ВНС представляют собой распределенные системы. Например, обширный класс распределенных ВНС, СРВ можно отнести к распределенным системам управления (РСУ). Современные вычислительные комплексы расчетного моделирования физических процессов построены как распределенные кластеры, а системы операторского сервисного обслуживания построены как распределенные системы обслуживания.

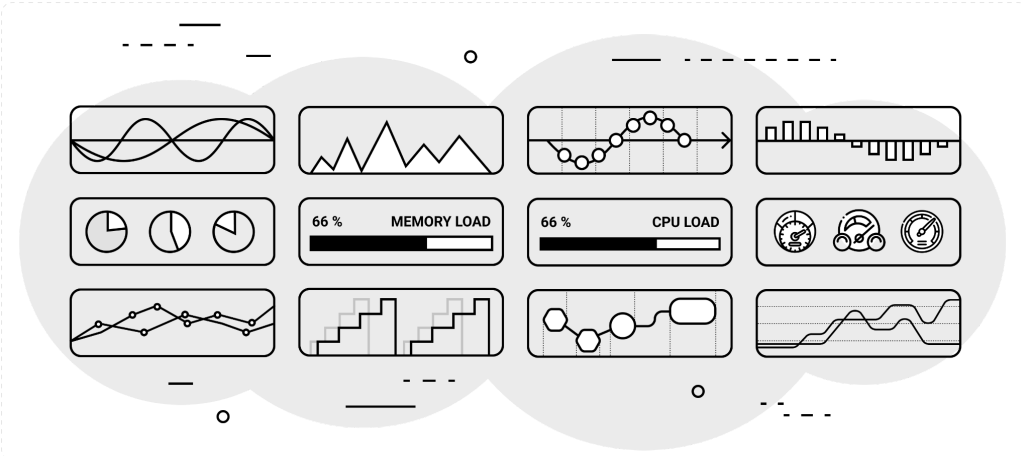
Такие системы отличаются друг от друга классом задач, возможностью смены конфигураций выполняемых задач «на лету» и целевыми потребителями (пользователями).



Глава 6. Надежная распределенная архитектура

Надежная архитектура подразумевает избыточность (разд. 4.3) узлов и данных. Чтобы получить избыточность, необходимо выполнить ряд действий. Можно создать резервную копию на отключаемом ресурсе. В этой ситуации появляется некоторая избыточность данных, но после отключения ресурса работать с такой резервной копией уже не представляется возможным.

Избыточность возникает, когда на пути от источника к потребителю данные проходят через промежуточный узел, где происходит кэширование (разд. 6.2). Такие кэшированные данные вполне можно использовать как резервные. Чтобы намеренно создать избыточность данных одного узла на другом, следует провести принудительную репликацию данных (разд. 6.4). Репликация обычно представляет собой одноразовый процесс, если не предполагается создание полностью персистентной системы, в которой будут сохранены все изменившиеся со временем реплики.



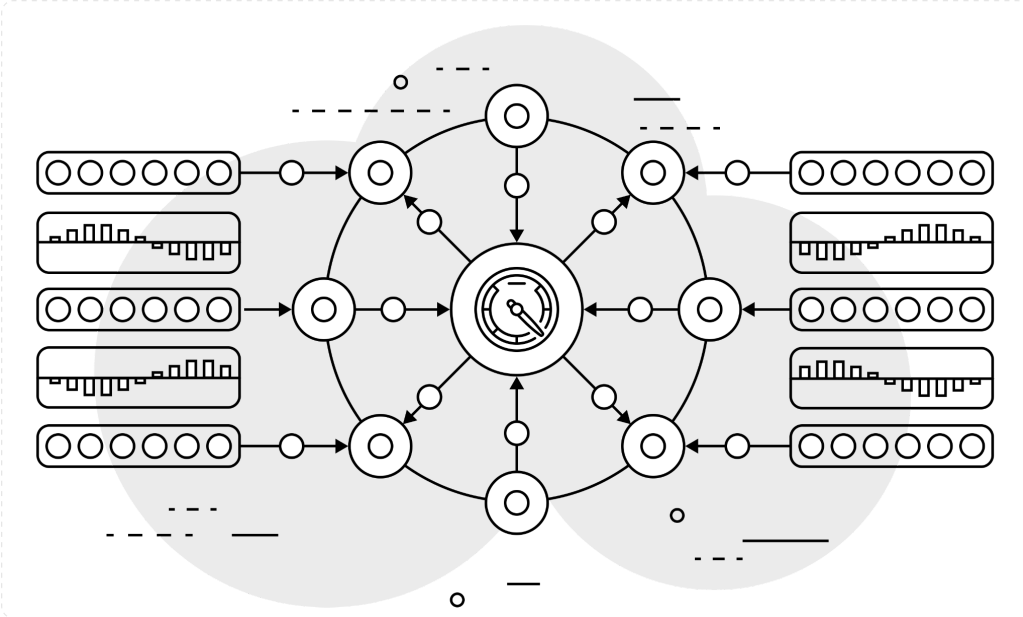
Глава 7. Отказоустойчивость и высокая доступность

7.1. Диагностика и качество сервиса (QoS)

Обеспечение качества сервиса QoS[67] (обслуживания) является важнейшей задачей любой ВНС, в особенности СРВ. Существует множество разных определений QoS — например, «принцип предоставления различным классам трафика различных приоритетов в обслуживании», или «вероятность того, что сеть связи соответствует заданному соглашению об уровне обслуживания (SLA[68])», или «обозначение вероятности прохождения пакета между двумя точками сети».

РСУ обычно взаимодействуют не с трафиком низкого уровня, а с протоколами высокого уровня: начиная с протоколов транспортного уровня модели OSI[69] активно применяют протоколы сеансового уровня. В этом случае перечисленные выше определения можно свести к следующему: QoS — принцип

[67] Quality of Service, #7-01
[68] Service Layer Agreement, #7-02
[69] OSI, #7-03



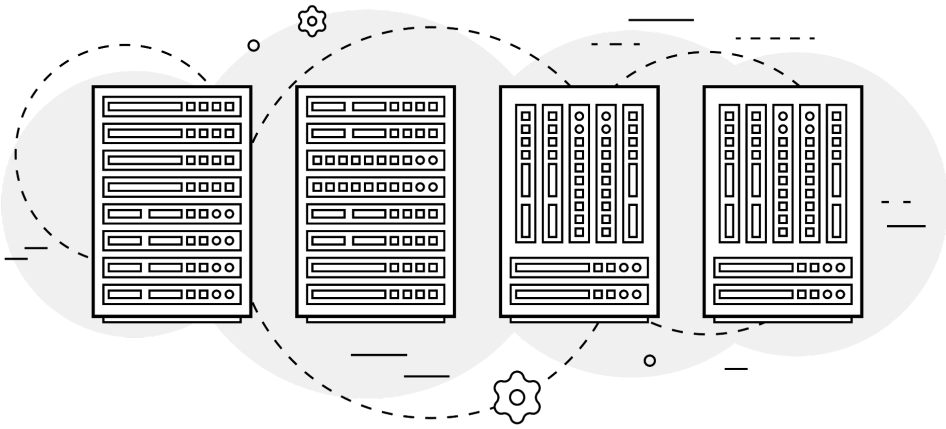
Глава 8. Балансировка нагрузки

Балансировка нагрузки[88] (планирование нагрузки) РСУ является одной из сложнейших задач современных инфраструктур. Балансировка нагрузки — это метод распределения заданий между несколькими распределенными узлами. Целью балансировки являются:

- оптимизация использования ресурсов, сокращение времени отклика;
- горизонтальное масштабирование (разд. 10.2) узлов РСУ (динамическое добавление/удаление устройств);
- предотвращение перегрузки узлов;
- обеспечение отказоустойчивости (резервирования).

Архитектура схем и алгоритмов балансировки нагрузки весьма разнообразна, она зависит от целевой архитектуры системы. Например, архитектура балансировки нагрузки на узлы

[88] Load balancing, #8-01

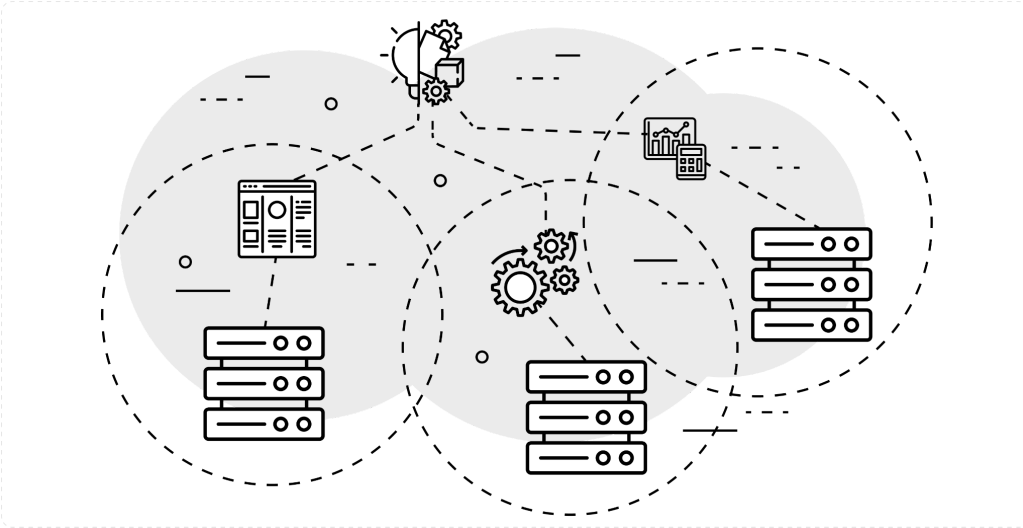


Глава 9. Кластеризация

В некоторых задачах РСУ, СРВ возможным и эффективным решением является объединение корневых узлов в единый вычислительный кластер. Кластерные вычисления необходимы для выполнения самых разнообразных задач, прежде всего для ускорения вычислений путем распределения задач по узлам.

9.1. Принципы построения

Вычислительным кластером называют объединение нескольких вычислительных узлов с помощью высокоскоростных каналов связи, с точки зрения пользователя представляющее собой единый аппаратный ресурс. Кластеризация подразумевает ряд жестких ограничений, применяемых к аппаратной составляющей задействованных узлов. Все узлы должны иметь идентичную аппаратную конфигурацию. Кластер всегда предполагает наличие выделенного интерконнекта, соединяющего кластерные (корневые) узлы. РСУ можно считать кластером, пока обеспечивается кворум кластера, то есть пока РСУ содержит достаточное число активных узлов, позволяющих кластеру выполнить поставленную задачу. Выделяют несколько разновидностей кластеров на основании доступности и типов выполняемых ими задач.



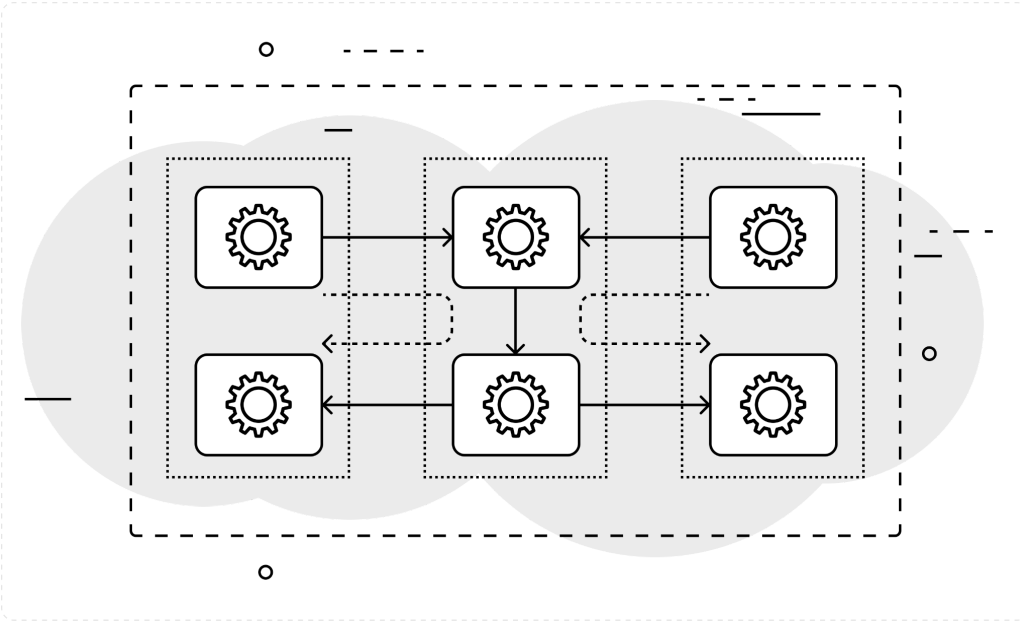
Глава 10. Масштабирование

Серьезная РСУ представляет собой большой и дорогостоящий инфраструктурный объект, который проектируют на десятилетия. Не исключено, что со временем система будет модернизирована, появятся новые дополнительные устройства, контроллеры и прочее оборудование. Это значит, что нагрузка на эксплуатируемую систему будет выше.

Необходимо заранее подготовиться к возможному масштабированию системы. Обычно масштабирование бывает необходимо в следующих случаях:

- увеличивается общий объем данных;
- увеличивается количество источников и разнообразие данных;
- усложняется логика обработки данных.

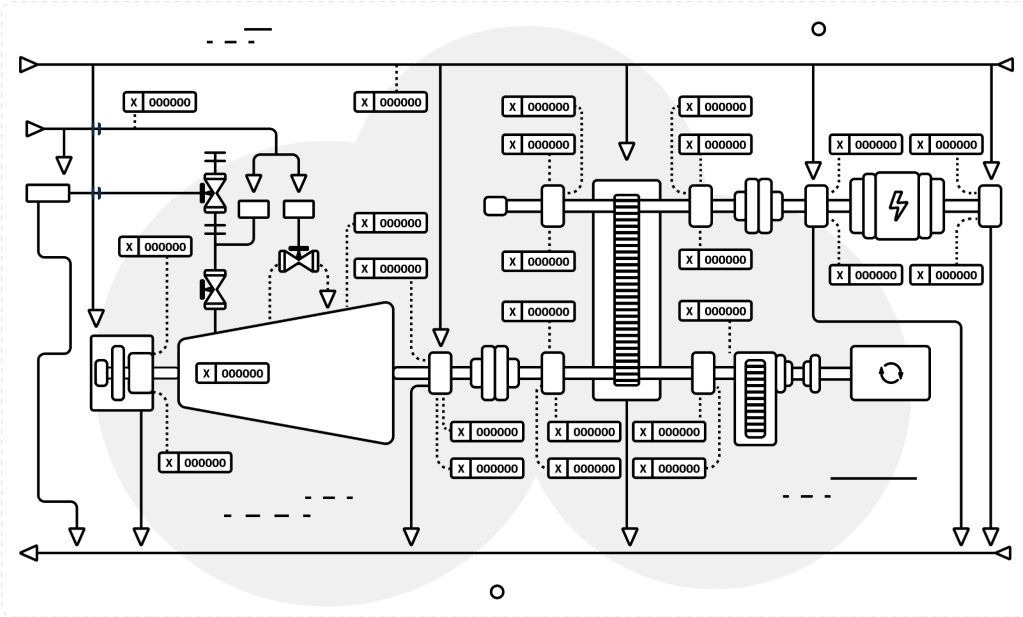
Во всех случаях масштабирование осуществляют путем добавления в систему новых узлов. Может случиться так, что



Глава 11. Архитектурные шаблоны

От архитектуры разрабатываемой платформы и реализации задач целевой системы зависит очень многое: возможности эффективной эксплуатации, диагностики, обслуживания, адаптации к новым условиям, модернизации, тестирования.

Важно определить задачи системы и приоритизировать их, определить, что важнее — надежность или скорость обработки, целостность данных или их доступность, как будет осуществляться взаимодействие с клиентами, что собой представляют клиенты системы. Например, клиентами IoT платформы будет большое число небольших устройств, каждое из которых обеспечивает достаточно скромную нагрузку на систему. В качестве противоположного примера можно привести СХД: чем выше ее уровень, тем, как правило, серьезнее клиенты. В роли клиентов выступает ограниченное число серверов (например, серверы ВНС).



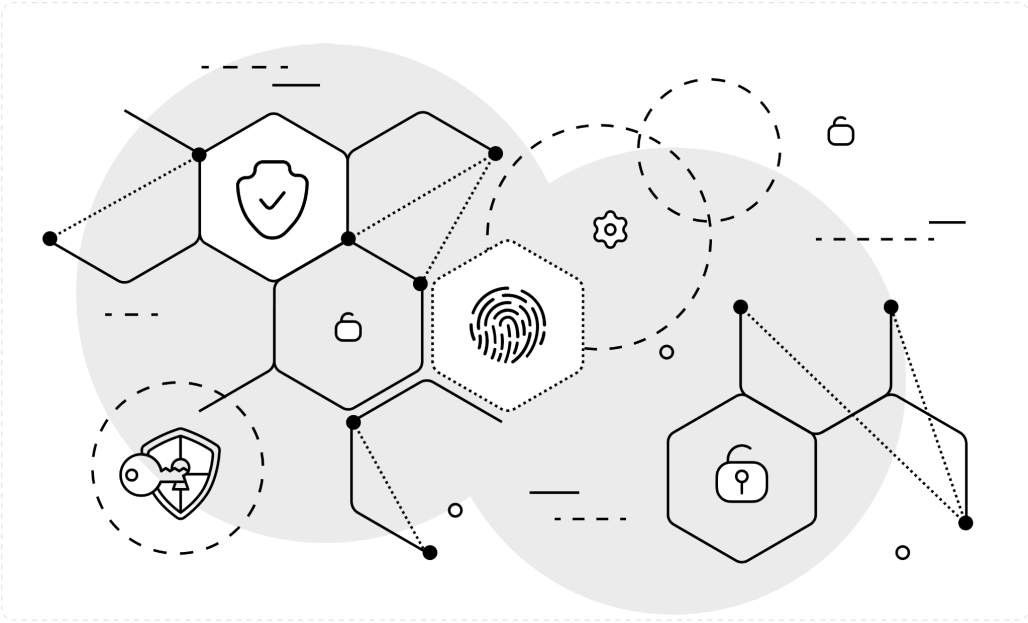
Глава 12. Приемы реализации

12.1. Распределенные СУБД

В этой книге не будут рассмотрены механизмы организации локального хранения данных на жестких дисках — подразумевается, что архитектор найдет эффективный способ. Системы распределенного хранения и доступа к данным наиболее эффективно реализуют в архитектуре *in-memory*. Таким образом, при создании распределенной системы, на узлах которой будут задействованы жесткие диски, мы будем считать, что было обеспечено надлежащее кэширование данных в оперативную память для организации эффективной работы узла.

12.1.1. Ключ-значение

Эффективным способом организации хранения распределенных данных является конструкция типа ключ-значение (*key-value*), которая представляет собой реализацию типичной РРП (разд. 4.2)



Глава 13. Кибербезопасность РСУ

Термин «кибербезопасность» не закреплен в каких-либо официальных документах. В настоящем тексте данный термин следует понимать как подмножество дисциплин, изучаемых наукой «информационная безопасность (ИБ)», гармонизированное и не противоречащее требованиям промышленной и функциональной безопасности. Эти требования очень важны для РСУ технологических и производственных процессов. Западные издания пользуются терминами *cybersecurity* и *computer security*, которые наиболее точно отражают смысл термина КБ.

Разделяют подходы к обеспечению КБ в области информационных и операционных технологий (*IT* и *OT cybersecurity*). Важно отметить, что РСУ однозначно следует в большей степени отнести к операционным технологиям, чем к информационным, даже в том случае, когда РСУ является составной частью ИТ инфраструктуры (а не промышленной системы), например, сервисной платформой, обслуживающей ИТ.

Приложение 1. Принятые сокращения

Сокр.	Расшифровка	Альт.
ACID	atomicity, consistency, isolation, durability	
AMP	asymmetric multiprocessing	
BASE	basically available, soft state, eventually consistent	
CAP	availability, consistency, partition tolerance	
CRDT	Conflict-Free Replicated Data Type	
CRUD	create, read, update, delete	
DBMS	database management system	СУБД
DCS	distributed control system	РСУ
DDoS	distributed DoS	
DoS	denial of service	
DRP	disaster recovery plan	
DRS	disaster recovery solution	
DSM	distributed shared memory	РРП
FIFO	first in, first out	
GbE	gigabit ethernet	
HDD	hard disk drive	
HLS	high load system	ВНС
IIoT	industrial IoT	
IoT	internet of things	
ITIL	IT infrastructure library	
LFU	least-frequently used	
LIFO	last in, first out	

Сокр.	Расшифровка	Альт.
LRU	least recently used	
MPS	messages per second	
MQCA	multi queue caching algorithm	
MRU	most recently used	
MSA	microservices architecture	MCA
MTBF	mean time between failures	
NUMA	non-uniform memory access	
NVMe	non-volatile memory express	
OTBF	operating time between failures	
RAID	redundant array of independent disks	
RAIM	redundant array of independent memory	
RAIN	redundant array of independent nodes	
RDMA	remote direct memory access	
RoCE	RDMA over converged ethernet	
RPO	recovery point objective	
RT	real time	PB
RTO	recovery time objective	
RTS	real time system	CPB
SCADA	supervisory control and data acquisition	
SDM	software defined memory	
SDN	software defined network	
SLA	service level agreement	
SLI	service level indicators	
SMP	symmetric multiprocessing	

Приложение 1. Принятые сокращения

Сокр.	**Расшифровка**	**Альт.**
SOA	service-oriented architecture	COA
SSD	solid-state storage device	
SQL	structured query language	
QoS	quality of service	
АО	аппаратное обеспечение	HW
АСУ	автоматизированные системы управления	ACS
АПК	аппаратно-программный комплекс	
АЭС	атомная электростанция	
ВНС	высоконагруженная система	HLS
ЗИП	запасные части, инструменты и принадлежности	
ИБ	информационная безопасность	
ЖЦ	жизненный цикл	
ЗИП	Запасные части, инструменты и принадлежности	
КИИ	критическая информационная инфраструктура	
МСА	микросервисная архитектура	MSA
ОЗУ	оперативно-записывающее устройство	
ОС	операционная система	OS
ПАЗ	противоаварийная защита	SIS
ПАК	программно-аппаратный комплекс	
ПЗУ	постоянно-записывающее устройство	RAM
ПЛК	программно-логический контроллер	PLC
ПТК	программно-технический комплекс	
ПО	программное обеспечение	SW
ППИ	прикладной программный интерфейс	API

Сокр.	**Расшифровка**	**Альт.**
РВ	реальное время	RT
РРП	распределенная разделяемая память	DSM
РС	распределенная система	
РСУ	распределенная система управления	DCS
РСУБД	распределенная СУБД	
СЗИ	средства защиты информации	
СКЗИ	средства криптографической защиты информации	
СРВ	система реального времени	RTS
СОА	сервис-ориентированная архитектура	SOA
СУБД	система управления базами данных	DBMS
СУИБ	система управления информационной безопасностью	SIEM
СХД	система хранения данных	
ТП	технологический процесс	
ФГУ	функционально-групповое управление	
ЦД	цифровой двойник	
ЦОД	центр обработки данных	DC

Приложение 2. Перечень шаблонов проектирования

Перечень упомянутых в книге шаблонов проектирования.

Шаблон	Pattern	Раздел
Амбассадор	Ambassador	11.1
Брокер очередей	Queue broker	11.1.4
Брокер сообщений	Message broker	11.1
Ведущий-ведомый	Leader-follower	11.1
Выбор лидера	Leader election	11.1
Выделенное ядро	Dedicated core	11.1.6
Двухфазная фиксация	Two-phase commit	11.2.8
Единая точка входа	Common entry point	11.1
Избыточность	Redundancy	11.1
Издатель-подписчик	Publisher-subscriber	11.1.7
Интерконнект	Interconnect	11.2.1
Каталог данных	Data catalog	10.4
Каналы и фильтры	Pipes & filters	11.1.1
Клиент-сервер	Client-server	11.1.2
Конвейер	Pipeline	11.1.1
Кэш на стороне	Cache-aside	11.1
Микросервисная архитектура	Micro Service Architecture (MSA)	11.1.4
Многоуровневая архитектура	Tiered architecture	11.1.2
Многослойная архитектура	Layered Architecture	11.1.5

Шаблон	**Pattern**	**Раздел**
Общие данные	Shared data	11.1
Одноранговая сеть	Peer to peer	11.1.2
Оркестратор	Orchestrator	11.2.5
Очередь сообщений	Message queue	11.1
Поточная очередь	Thread pool	11.1
Прицеп	Sidecar	11.1
Сага	Saga	11.2.9
Сервис-ориентированная архитектура	Service-Oriented Architecture (SOA)	11.2.3
Событийно управляемая архитектура	Event-driven architecture (EDA)	11.2.2
Суперконвейер	Super pipeline	11.1.3
Разделение ответственности команд и запросов	Command Query Responsibility Segregation (CQRS)	11.2.7
Разделяемая база данных	Shared database	11.1
Распределенное ядро	Distributed core	11.2.6
Реплицированные сервисы с распределением нагрузки	Replicated Load-Balanced Services (RLBS)	11.2
Фабрика данных	Data fabric	10.4
Фабрика сервисов	Service fabric	10.4
Хореография	Choreography	11.2.5
Шардинг	Sharding	11.1

Приложение 3. Список литературы

1. Вентцель Е. С., Овчаров Л. А. Теория вероятностей. М.: Наука, 1969. 368 с.

2. Воеводин В. В., Воеводин Вл. В. Параллельные вычисления. СПб.: БХВ-Петербург, 2002. 608 с.

3. Елизаров И. А., Мартемьянов Ю. Ф. Технические средства автоматизации. Программно-технические комплексы и контроллеры: учебное пособие. М.: «Издательство Машиностроение-1», 2004. 180 с.

4. Ивченко Г. И., Каштанов В. А., Коваленко И. Н. Теория массового обслуживания. Учебное пособие для вузов. М.: Высшая школа, 1982. 256 с.

5. Оленев Н. Н. Основы параллельного программирования в системе MPI. М.: ВЦ РАН, 2005. 80 с.

6. Петров О. Внутреннее устройство баз данных — глубокое погружение в то, как работают распределённые системы баз данных. O'Reilly, 2019. 349 с.

7. Подольный В. П. Архитектура высоконагруженные систем. Системы сбора информации, распределенные системы управления, системы реального времени. М.: OneBook, 2022. 160 с.

8. Согомонян Е. С., Слабаков Е. В. Самопроверяемые устройства и отказоустойчивые системы. М.: Радио и связь, 1989. 207 с.

9. Танненбаум Э. Архитектура компьютера. 5-е изд. СПб.: Питер, 2013. 884 с.

10. Танненбаум Э., ван Стеен М., Распределенные системы. Принципы и парадигмы. СПб.: Питер, 2003. 877 с.

11. Фаулер М., Садаладж П. Дж. NoSQL. новая методология разработки нереляционных баз данных. М.: «Вильямс», 2013. 192 с.

12. Sunshine CA, editor. Computer Network Architectures and Protocols. Springer Science & Business Media, 2013. 542 p.

13. Severance C, Dowd K. High Performance Computing (RISC Architectures, Optimization & Benchmarks), 2nd ed. O'Reilly Media, 1998. 466 p.

14. McCreary D, Kelly A. Making Sense of NoSQL: A guide for managers and the rest of us. Manning Publications, 2013. 312 p.

15. Dhamdhere DM. Operating Systems. McGraw-Hill Science/Engineering/Math, 2008. 864 p.

16. Dubrova E. Fault-Tolerant Design. New York: Springer New York, 2013. 185 p.

17. Foster I, Kesselman C. The Grid: Blueprint for a New Computing Infrastructure. San Francisco, CA: Morgan Kaufmann Publishers, 1998. 562 p.

18. Hohpe G, Woolf B. Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions. Pearson Education, 2012. 736 p.

19. Davies H, Bressan B, editors. A History of International Research Networking: The People who Made it Happen. John Wiley & Sons, 2010. 347 p.

20. Huang D, Wu H. Mobile Cloud Computing: Foundations and Service Models. Morgan Kaufmann, 2017. 317 p.

21. Kshemkalyani AD, Singhal M. Distributed Computing: Principles, Algorithms, and Systems. Cambridge University Press, 2011.

22. Molyneaux I. The Art of Application Performance Testing: Help for Programmers and Quality Assurance. O'Reilly Media, 2009.

23. Curé O, Blin G. Chapter 2. Database Management Systems. In: RDF Database Systems: Triples Storage and SPARQL Query Processing. Elsevier Science, 2014. 256 p.

24. Patterson DA, Hennessy JL. Computer Architecture: A Quantitative Approach. 4th ed. Burlington, Massachusetts: Morgan Kaufmann, 2006. 704 p.

25. Chevance RJ. Server Architectures: Multiprocessors, Clusters, Parallel Systems, Web Servers, Storage Solutions. Digital Press, 2004. 784 p.

26. Sadalage PJ, Fowler M. NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence. Addison-Wesley Professional, 2012. 192 p.

27. Tiwari Sh. Professional NoSQL. John Wiley & Sons Inc, 2011. 384 p.

28. Solihin Y. Fundamentals of Parallel Multicore Architecture. Boca Raton, Florida: Chapman and Hall/CRC, 2015. 468 p.

29. Sorin DJ, Hill MD, Wood DA. A Primer on Memory Consistency and Cache Coherence. Morgan & Claypool, 2011. 210 p.

30. Sosinsky B. Cloud Computing Bible. Wiley, 2010.

31. Stallings W. Operating Systems — Internals and Design Principles. 7th ed. Prentice Hall, 2011. 816 p.

Приложение 4. Ссылки

В книге приведены ссылки на источники (статьи, прочие материалы), начинающиеся с символа «#». Символом «▸» отмечены ссылки на рекомендуемые видеоматериалы.

Ссылки на источники можно найти на странице книги по следующей ссылке в QR-коде.

Вадим Подольный

Архитектура высоконагруженных систем

Издание второе

Литературный редактор – Надежда Тихомирова

Корректоры – Максим Кузнецов

Иллюстратор – Анастасия Балякно

Верстка – Любовь Подольная

Подписано в печать 02.02.2024 г.
Формат 60x90/8. Бумага офсетная. Печать цифровая.
Заказ No 25000. Тираж 100 экз.
Отпечатано в типографии «OneBook.ru», www.onebook.ru
ООО «Сам Полиграфист»
109316, г. Москва, Волгоградский проспект, дом 42, корп. 5,
«Технополис Москва».