На пути к новой теории информационных систем

© С. В. Знаменский
Институт программных систем имени А.К. Айламазяна РАН
Переславль-Залесский
svz@latex.pereslavl.ru

Аннотация

Эволюционирующая информационная система, способная адаптироваться к изменяющимся требованиям и противостоять непредвиденным угрозам, должна сочетать высокие корректность и быстроту реакции даже при структурных перестройках, задержках и временных потерях связи. Известная теорема CAP Брювера отрицает возможность построения таких систем.

Новые основания теории информационных систем позволяют построить содержательный контрпример к теореме CAP Брювера. Для модельной задачи приведённое построение теоретически доказывает возможность практической реализации информационной системы с расчётными показатели качества в сто и более раз лучшими, чем у любой возможной реализации стандартными методами.

Приводится грубый набросок архитектуры информационной системы общего назначения с неограниченно изменяемыми функциональностью концептуальной моделью и физической реализацией.

Введение

«Dubito ergo cogito, cogito ergo sum»
Рене Декарт

Примерами систем, ориентированных на длительное существование, являются интернет в целом, социальные сети, Google Scholar, Википедия, Sourceforge, инфраструктура разработки ядра ОС Linux и её дистрибутивов и многие другие. Ориентация на длительное существование приобретает растущую популярность (см., например, [1])

Труды 14-й Всероссийской научной конференции «Электронные библиотеки: перспективные методы и технологии, электронные коллекции» — RCDL-2012, Переславль-Залесский, Россия, 15-18 октября 2012 г.

Длительно существующая компьютерная система сталкивается с новыми видами или разновидностями опаснейших угроз, которые не были предусмотрены при её создании. Нереально в деталях предусмотреть все возможные угрозы: хакерское вторжение, диверсию или террористический акт, грубую критическую ошибку персонала, смену технических политик, сдвиг потребностей пользователей, смену ориентиров руководства, стихийное бедствие, технократическую катастрофу, всплеск популярности.

Технологии защиты информации и предупреждения угроз здесь не обсуждаются. Речь идёт исключительно о способности программного обеспечения компьютерной системы обеспечить службе сопровождения возможности исследовать происходящее, встраивать защитные механизмы, устранять последствия поломок, по необходимости восстанавливать разрушенное, добавлять непредусмотренную функциональность и расширять востребованные сервисы.

Служба сопровождения здесь понимается в общем смысле, от системных сервисов мониторинга и самовосстановления до технических администраторов и разработчиков разного уровня.

Похожие свойства являются объектом широкого фронта исследований (см., например, [2,3]), оставляющих комплексную реализацию на крайне отдалённую перспективу.

Принципиальным препятствием к построению устойчивой отзывчивой системы видится *теорема CAP Брювера* [4], утверждающая невозможность сочетания трёх свойств:

Consistency (*Согласованность — ответы на запросы к системе логически непротиворечивы*).

Availability (*Доступность — ответ на любой запрос может быть получен незамедлительно*).

Partition-tolerance (*Устойчивость к разделению — при восстановлении потерянной на время связи функциональность и внутренняя целостность восстанавливаются*).

Не секрет [5], что формулировка теоремы CAP провоцирует на сомнительные популяризации.

Формальное доказательство этой теоремы в [6] основано на прозрачной идее:

$$\begin{pmatrix}\text{появление}\\ \text{различий}\end{pmatrix}\&\begin{pmatrix}\text{их}\\ \textit{доступность}\end{pmatrix}\Rightarrow\begin{pmatrix}\text{противо-}\\ \text{речие!!!}\end{pmatrix}$$

Практику очевидно, что после обрыва связи с филиалом компании работа *должна* быть продолжена с адекватными ситуации ограничениями и после восстановления связи система *должна* полностью восстановиться без потерь информации, то есть логика доказательства противоречит потребностям практики.

Неприемлемость строго доказанного результата в науке встречается. Достаточно вспомнить классические безупречные доказательства невозможности рационального корня из числа 2, корней из отрицательных чисел и недифференцируемости функции Хевисайда. Их неприемлемость привела в своё время к развитию и широкому применению новых теорий, ставших основой современной математики — теории вещественного числа, теории комплексных чисел и теории обобщённых функций.

1 Адекватное разделение ИС

Начнём с простейшей ситуации, в которой все отмеченные качества критичны — с организации совместного доступа к редактированию файла.

1.1 Неизбежность противоречий

Классические блокировки доступа к файлу ощутимо задерживают работу и делают документ недоступным при поломке или потере связи. Попытки этого избежать (например, предоставление пользователю возможности снимать блокировку другого пользователя) создают реальную угрозу потери введённых данных.

Выход казалось бы найден в декомпозиции редактирования документа на коммутирующие изменения (OT-operational transformations). Это дало эффектную иллюзию одновременной работы и стало общепринятым [7] современным средством организации совместной деятельности.

Однако такое редактирование исходного кода программы двумя пользователями затрудняет работу и повышает риск ошибки [8]. Например, пока открывающая скобка одним пользователем уже набрана, а закрывающая ещё нет, второй лишён возможности тестировать код и подсветка синтаксиса у него испорчена.

Есть и другая опасность:

Пример: *Вычитывая совместную статью на удалённом сервере, профессора A и B замечают ошибку в ключевой математической формуле. Профессор A исправляет формулу и закрывает окно с чувством полной уверенности в истинности формулы.. Одновременно профессор B так же поступает с другим местом той же формулы. Оба считают ошибку исправленной, обе части формулы изменены, . . . — но* формула вновь неверна!

Рассмотренные примеры показывают, что даже безупречно формально обоснованно информационные технологии не позволяют избежать конфликтов доступа. Такие конфликты приводит к иногда невосполнимым потерям пользовательских данных.

Этим конфликт напоминает дуэль.

1.1.1 Принцип Байрона-Галуа

Дуэли категорически недопустимы.

Побочным эффектом дуэлей стали невосполнимые для человечества потери.

Не улей и не муравейник должны стать образцом адаптивной системы, поскольку человеческий опыт организации общества обеспечил более чем тысячекратное ускорение эволюции по сравнению с биологической.

Сложившиеся нормы и принципы организации цивилизованного общества нуждаются в переосмыслении применительно к эволюционирующим компьютерным системам.

Для конфликтов это может быть сделано так:

1. *Логически обоснованные алгоритмы разрешения конфликтов должны использоваться везде, где они корректно применимы.*
2. *Конфликт не нашедший конструктивного решения, должен разрешаться на более высоком уровне организации.*
3. *Вердикт должен быть вынесен на основе рассмотрения всех обстоятельств и может предусматривать предупредительные меры — поиск или разработку подходящего конструктивного решения.*
4. *Если вердикт не разрешает конфликт содержательно, то правило действует рекурсивно до высшего уровня управления.*
5. *Сопутствующая информация обязательно должна сохраняться для будущих рассмотрений.*

Для информационных систем это означает особую заботу о сохранности всех входных данных и корректности их обработки, старательное исключение любых возможностей отката транзакций.

В ситуации совместно редактируемого файла отсюда вытекает важность сохранения документа таким, каким его видел пользователь и всей сопутствующей информации для возможного

пересмотра принятого решения и использованной логики.

Взяв на заметку, что доступность полной неподдельной истории происходившего существенно помогает правильному разрешению конфликтов, вернёмся к разделению информационной системы на части A и B.

Под устойчивостью информационной системы к разделению на части будем понимать следующее: Части A и B изменяются и связаны с потерями и задержками. Тогда о текущем значении данного, доступного из частей A и B говорить бессмысленно. Что действительно можно требовать от функционирующей при непредсказуемых задержках информационной системы, так это определить последние моменты прошлого, на который в системе в целом или локально было установлено последнее корректно обоснованное значение этого данного и выдать эти значения и моменты.

Использование старых данных опасно не потерей актуальности. В самом деле: если система работает быстро, то разница во времени ничтожна, а если время обработки значительно, то пока ответ по данным на момент запроса будет получен, момент запроса устареет на такое же время обработки.

Опасность в том, что обрабатывать старые данные можно лишь по их состоянию на *фиксированный* момент времени. Изменившееся значение противоречит прежнему.

Мы приходим к требованию *ретроспективной согласованности данных информационной системы*:

- В любой момент времени t в системе доступен прошедший *момент истины* $\mu(t) < t$, которым завершилась доступная история согласованных изменений данных системы.
- Для любого предыдущего момента $\tau < \mu(t)$ система возвращает согласованные реплики.

Разность $t - \mu(t)$ назовём *длительностью согласования*. При удачном разделении длительность согласования для части системы оказывается меньше, чем для целой системы.

Ретроспективная информационная система может не всегда иметь единую концептуальную модель. Более того, её концептуальная модель может изменяться не мгновенно и не одновременно во всех частях. Этим она принципиально отличается от *последовательных* информационных систем, основанных на моделировании последовательности изменений глобального согласованного состояния системы.

Практически все существующие информационные системы *последовательные*. Это косвенно подтверждается общепринятостью доказанности теоремы CAP.

Рассмотрим возможность и целесообразность построения системы, сочетающей ретроспективность с *адекватностью* — безупречностью реакции на запросы и соразмерностью последствий временных поломок и сбоев.

1.2 Пробная задача

Банк имеет тысячу клиентов в двух городах A и B. Расчётное максимальное время задержки передачи данных между городами $T = 0.25$ секунды.

Города расположены так, что устранить эту задержки технически невозможно[1]

Для поддержки переводов денег со счёта на счёт банк нуждается в качественной информационной системе с по возможности быстрым временем отклика τ, рассчитанную на обработку не более десяти запросов в секунду с одновременной поддержкой ста сессий:

1. Результаты любых запросов гарантированно должны быть корректны и согласованы.
2. При любом обращении к сервису задержка обработки не должна превосходить τ для любого из следующих запросов:
   (a) Просмотр состояния счёта и истории его изменений.
   (b) Перевод имеющейся суммы на другой счёт.
   (c) Получение отчётов.

Для предельного упрощения задачи полагаем, что банк с наличностью не работает, кредитованием не занимается, вне городов никого не обслуживает, пользователи заведены с неизменными личными данными и для простоты не переезжают, а потерями и задержками внутри городов можно пренебречь.

1.3 Последовательный подход

Информация о текущем состоянии каждого счёта должна где-то храниться. Запрос о переводе на этот счёт из другого города, должен дойти и после этого ответ должен вернуться обратно, чтобы пользователь смог знать, что запрос принят. Поэтому реакция системы не может быть гарантирована быстрее, чем в течение $\tau = 2T = 0.5$ секунды.

[1] При передаче со скоростью света через геостационарный спутник сигнал задерживается на $1/4$ секунды. При недоступности геостационарного решения неизбежны задержки при переключениях на очередной спутник.

1.4 Ретроспективный подход и сравнение

В мегаполисах размещаются идентичные серверы. Каждый из них ведёт счета пользователей из своего мегаполиса.

Сервер отказывается выполнить перевод если денег на счету отправителя недостаточно. Если на счету достаточно денег, то сервер немедленно изменяет состояние локальных счетов и надёжно запоминает своё обязательство по возможности быстро выполнить перевод.

После обработки серии запросов (например каждые пол-секунды) серверы передают друг другу полные списки накопившихся запросов и записей изменений в счетах и извещения о получении списков. Извещение о получении посылается только в том случае, когда пакет получен полностью и содержит момент времени, вплоть до которого все данные получены. Пакетная передача помогает гарантировать единый порядок и отсутствие пропусков в общей части информации серверов и корректно снизу оценить момент истины системы $\mu(t)$, вся история изменений вплоть до которого получена на обоих серверах.

Пользователь получает ответ на свой запрос с учётом реакции только ближайшего сервера. Реакция удалённого (например, уведомление о получении перевода в другой город) может быть учтена в отдельном последующем через секунду запросе (который может делаться автоматически либо клиентским приложением либо на основе технологии comet, позволяющей локальному серверу самостоятельно передать такое уведомление в момент получения).

1.4.1 Скорость и пользовательский интерфейс

Условия задачи не влекут никаких технических препятствий к получению быстрого (в течение десяти миллисекунд) ответа на запрос на основе имеющейся на ближайшем сервере информации. Повышение реактивности системы в сто раз здесь достигнуто по сути включением в реакцию системы на запрос пользователя только немедленно доступной информации.

Это позволяет обеспечить пользователя удобным интерфейсом, дающим возможность выбора, наблюдать за процессом обработки, любо спокойно выключить клиентское приложение либо заняться операциями с другими счетами.

Взаимодействие с внешней системой может породить сложную совместную транзакцию (например, перевод с одного счёта на другой в другой город, оттуда на третий и снова назад) и потери времени станут неизбежны. Для клиента, ожидающего поступлений на счёт, может быть создано клиентское приложение, ожидающее изменения состояния счёта для повторения запроса, отвергнутого из-за недостатка денег на счету.

Разумеется, при переводе денег упомянутая секунда задержки не составляет проблемы для клиента. Сюжет задачи выбран для ясности рассмотрения, а ничем не компенсированные задержки в других ситуациях могут быть критическими.

1.4.2 Точность и логичность

В конкретной задаче общая сумма денег на счетах в реальности не константа. Ясно и привычно, что отправленные с одного счёта деньги поступают на другой не мгновенно. Стандартная реализация на основе транзакций прячет время транзакции в неповоротливость пользовательского интерфейса, искусственно создавая иллюзию мгновенности сделки.

Ретроспективное решение делает находящиеся в пути деньги доступными для точного последующего анализа.

История изменений счёта, полученная пользователем, полна до момента $\mu(t)$, а после $\mu(t)$ в ней могут временно отсутствовать деньги, «находящиеся в пути».

Запросы сводной информации на момент, предшествующий $\mu(t)$, не зависят от сервера. Для запроса сводных данных на последний момент у пользователя есть две возможности:

1. посмотреть сводную по всем доступным (неполным) данным с предупреждением о неполноте,
2. посмотреть *полную* сводную по всем данным на последний момент $\mu(t)$, для которого они доступны.

Если запросы следуют не чаще, чем может обработать традиционная система, то выдаётся в точности тот ответ, каким он был бы в традиционной системе на запрос, посланный из оптимально выбранного момента прошлого (неважно, был ли такой запрос в действительности).

1.4.3 Устойчивость к катастрофам и потерям связи

Поломка связи между городами при стандартном подходе полностью блокирует операции в одном из них. При ретроспективном блокируются лишь операции, непосредственно нуждающиеся в этой связи, остальное продолжает работать и сервис немедленно полностью восстанавливается при восстановлении связи.

Кроме упомянутых, ретроспективная организация имеет и другие полезные особенности:

Поломка или неожиданная потеря одного сервера базы данных (попадание метеорита) при стандартном подходе должно быть заранее предупреждено специальными мерами, например, чёткой организацией резервного копирования или репликации базы. Иначе потери могут оказаться фатальны.

При ретроспективном подходе без каких-либо дополнительных мер исчезновение сервера приводит к потере лишь части обновлений за последние доли секунды, что оставляет шансы на спасение.

1.5 Результат

Теорема. *Существует возможность реализации прикладной информационной системы с ретроспективным доступом, вполне адекватной функциональностью при разделении, быстрым автоматическим восстановлением после восстановления связи и с временем отклика в сто раз ниже, чем у любой реализации на традиционной основе.*

Эта теорема разумеется противоречит не самой теореме CAP, а её популярным интерпретациям.

Рассмотренный пример вырос из неоднократно обсуждавшихся на конференции [9–11] ретроспективных систем. Он ставит ряд вопросов:

1. Ведёт ли он к общей технологии разработки ретроспективных информационных систем?
2. Не чрезмерны ли требования ретроспективной парадигмы к объёмам памяти и вычислительных ресурсов?
3. Сможет ли ретроспективная технология кардинально упростить исправление концептуальных ошибок и создание эволюционирующих систем?
4. Сможет ли ретроспективная технология резко уменьшить потери и неудобства пользователей, связанные с обновлением систем?
5. Сможет ли положительный эффект ретроспективного подхода проявиться при организации многопроцессорных распределённых систем и высокопроизводительных вычислений с большими гранулами параллелизма либо с задержками переконфигурирования FPGA [12]?

Возникшие вопросы ждут комплексной фундаментальной проработки. Здесь мы затронем лишь ключевые аспекты первых двух из них.

2 Ретроспективная парадигма программной инженерии

Проектирование системной архитектуры предполагает разделение системы на наиболее крупные составные части и принятие конструктивных решений, которые после их принятия с трудом поддаются изменению.

В рассматриваемой общей постановке неограниченно изменяются концептуальная модель, используемое оборудование и приложения (сервисы). Поэтому они не могут быть общей основой системной архитектуры для ретроспективной системы. Такой основой может быть только стабильная подсистема идентификации структур процессов и данных, обеспечивающая безусловную доступность неподдельной истории при произвольных изменениях.

Прежде, чем её описывать, следует разобраться с понятиями модульности и информационных объектов, которые в системе с историей и изменчивостью имеют свои особенности.

2.1 Модульность

Модульность освобождает разработчиков прикладных программ от сложностей организации исполнения. Она традиционно понимается как разделение на изолированные модули со своими кодом и данными, обменивающиеся сообщениями. Рассмотрим, как такое понимание может при структурных изменениях приводить к неконтролируемому росту сложности организации обмена сообщений.

Сложность эта порождается двумя принципиальными дефектами неуправляемого обмена сообщениями:

1. Новому исполнителю недоступна информация, разосланная участникам совместно выполняющейся работы.
2. Сообщения теряются по разным причинам.
3. Сообщение читается с неизвестным запозданием, за это время оно может потерять адекватность, а отменить непрочитанное сообщение автор не может.

Отсюда опасность использования изоляции модулей с организацией обмена сообщениями в коллаборативном приложении: после успешного тестирования малейшее изменение структуры взаимодействия или возникновение задержек может неожиданно выявить необходимость корректирующих сообщений. В итоге безупречная работа обеспечивается лишь до замены модулей, и откатов неправильных действий, а запуск новых исполнителей на замененном оборудовании и (или) с новыми алгоритмами исполнения требует серьёзного рефакторинга системы.

Альтернативой обмена сообщениями является публикация данных с подпиской на изменения, и, в частности, стандартное решение DDS (Data Distribution Service) [13].

Механизм обработки запросов на подписку и на публикацию авторы не регламентируют. Во избежание коллизий запросы вероятно должны обрабатываются строго, то есть проходить через общий вычислительный узел, что принципиально ограничивает масштабируемость (см., например, [14]). В любом случае это не снижает ценности DDS как перспективного новейшего технического решения для общей памяти распределённой системы, которое повидимому может эффективно масштабироваться при устойчивых структурах данных с достаточно редко меняющимися настройками.

Подписка производится с настройками, устанавливающими количество хранимых версий, время жизни данных, количество реплицируемых копий, планово допустимые задержки, учитывающими период изменения данных и приоритетность передачи. DDS ничем не ограничивает свободы программиста по выбору этих настроек.

Получая полный доступ к гибкому распределению вычислительных ресурсов, прикладной программист к сожалению практически лишён разумных ориентиров. Это создаёт сложнейшую проблему избыточной гибкости настроек, которые для долгоживущей системы должны быть ещё и динамически управляемыми.

Решение этой проблемы требует системной организации для гибкого регулирования минимальным количеством прозрачных настроек.

Построив дерево всех объектов, настройки будем считать вектор-функцией от узла. Если предположить, что настройки чаще должны совпадать для смежных узлов, то задание настроек сводится к разрезанию дерева и заданию настроек для каждой связной части. Поскольку каждая связная часть имеет ближайший к корню элемент, то именно в нём можно задавать или менять настройки. Тогда настройки, сделанные в любом узле, наследуются во всей ветке, за исключением подветок со своими настройками.

Это означает, что проблема настроек разделяемой памяти сводится к разумному представлению всех данных системы в виде дерева объектов.

Примерами таких веток в рассмотренном примере являются данные о изменениях в счёте отдельного пользователя и данные по городу. В общем случае такое разделение означает иерархию управления изменениями в системе.

2.2 Контекстная автономная архитектура

Архитектура контекстно-автономной системы [9] формируется на основе иерархии управления изменениями. Используется идея процессного подхода, рекомендованного стандартами ИСО серии 9000. Однако вместо термина *процесс* мы будем использовать более общий термин *активность*, не предполагающий обязательности входных данных и не ассоциирующийся с процессами операционной системы. Так, множество всех ИСО9000-процессов является подмножеством всех активностей системы.

Другие примеры активностей:

- нажатие пользователем кнопки, переход по ссылке или ввод текста;
- обнаружение события, нуждающегося в обработке.
- обработка данных, полученных от разных пользователей, при совместном редактировании.

Мониторинг сложных активностей осуществляют активности управления качеством, в частности корректирующие взаимных приоритетов дочерних активностей.

Системная активность управляет качеством системы в целом через такие дочерние активности, заведомо неполный список которых включает:

- мониторинг наличия и состояния физических устройств и каналов связи;
- мониторинг информационного обмена между активностями и между физическими узлами и перемещение активностей.
- мониторинг взаимных приоритетов активностей с общим родителем;
- мониторинг качества обновления информации;
- документирование кода и данных системы;
- уточнение приоритетов информационного обслуживания активностей;

Большинство задач системной активности в начальной фазе существования системы может осуществляться человеком, а затем переводиться на автоматическое управление. Автоматизация управления требует постановок задач поиска эффективных алгоритмов. Эти алгоритмы должны будут разумным образом перестраивать размещения активностей по физическим устройствам и оптимизировать коммуникации, используя в качестве входных данных изменения физической конфигурации системы, изменения приоритетов и статистику мониторинга активностей.

2.2.1 Логическая идентификация данных

Элемент данных (например, атрибут объекта) может сохраняться и реплицироваться в разных узлах системы. Изменение данного создаёт его новую версию. Прежде чем идентифицировать версии в различных местах системы, мы должны идентифицировать то, версии чего в ней сохраняются. Идентификатор самого элемента (например, поле конкретной формы, заполненное конкретным пользователем, содержащее код исполняемого в системе скрипта), не учитывающий версии и места хранения, будем называть *логическим идентификатором элемента данных*. Прозрачность идентификации данных крайне важна для доработок системы.

Для каждой активности A определим её *контекст* $\mathcal{D}(A) = [d_0(A), d_1(A)] \subset U$ как сегмент в линейно упорядоченном универсуме U всех возможных логических идентификаторов данных [15].

Универсум U будем считать *неисчерпаемым* в следующем смысле:

$$\forall x, y \in U \exists z \in U : x < z < y$$

Неисчерпаемость обеспечивает возможность размещения в любом контексте множества данных любой конечной мощности. Например, универсум строк символов неограниченной длины с отношением лексикографической упорядоченности неисчерпаем.

Для вложенных контекстов активность с более широким контекстом будем называть родительской, а с более узким — дочерней.

Принцип разделения ответственности и принцип Байрона-Галуа приводят к следующим требованиям:

1. *Контексты любых двух активностей либо вложены, либо не пересекаются.*
2. *Активность с более широким контекстом не модифицирует контексты дочерних активностей.*
3. *Перемещение активности в новую область резервирует старый контекст для возможного возврата и изоморфно переносит все дочерние активности.*

Добавляя совокупную активность, поддержанную системой, получаем в каждый момент времени дерево активностей. Оно изменяется — активности появляются, замирают, глубина вложенности активности может изменяться. Это обеспечивает произвольные структурные перестройки и гарантирует доступ к неискажённому прошлому.

2.3 Организация исполнения

Иерархия контекстов активностей упрощает проблему регулирования разделения ресурсов. Приоритеты допустимости задержек, уровня защищённости и ценности истории устанавливаются (и могут в любой момент быть пересмотрены) для общих активностей верхнего уровня.

1. Состояние изменяющегося объекта однозначно определено лишь в достаточно давнем прошлом.
2. Безупречный ответ может быть дан лишь на запросы об устоявшемся (линейно упорядоченном) прошлом, предшествующим моменту истины множества данных, нужных доля обработки запроса.
3. *Момент истины* (ранее которого прошлое устоялось) быстро определяется по времени и идентификатору.
4. Обновление информации в автономном контексте данных происходит с ограниченной частотой при появлении изменений, необработанных на момент, когда все входные данные устоялись.
5. Изменения помечаются комбинированным временем обработки, интерпретируемым и как частично упорядоченное логическое (линейно упорядоченное до момента истины), и как приближение к физическому времени.

Ограниченность частоты изменений обеспечивает локализацию последствий ошибок и открывает возможность оперативного автоматического выявления проявившихся в работающей системе противоречий. Локализация последствий ошибок вместе с автоматической диагностикой особенно важна для ретроспективной системы, поскольку перестройку «на ходу» невозможно осуществить без временных рассогласований и исправляемых ошибок.

Поясним сказанное.

Пример: один процесс обеспечивает $a = -b$, а другой — $b = a + 1$.

Подобное нарушение логики, проскользнувшее в обычную работающую систему, либо останется незамеченным, либо заставит систему бесконечно пересчитывать a и b.

Фиксация частоты изменений ограничивает любые негативные эффекты, в том числе и этот.

При появлении нескольких кандидатов для значения в близком времени возникает конфликт, требующий разрешения. Иногда для разрешения

могут использоваться либо выбор значения, поступившего последним, либо выбор наибольшего (рекордного) из поступивших значений.

Остальные значения игнорируются или удаляются при записи. Если конфликты сложнее, то могут вводиться дополнительные активности для разных источников значений с целью разрешения конфликтов.

Если, например, от пользователя поступает более десяти форм в секунду, то это возможная попытка взлома. Классическое эффективное средство борьбы — это игнорирование промежуточных запросов от одного пользователя в малом промежутке времени. При этом запросы от всех разных пользователей должны быть сохранены и корректно обработаны.

Уменьшение частоты изменений контекстов ресурсозатратных активностей, таких как профиля пользователя или объёмная статистика, экономит ресурсы.

2.4 Экономия памяти

История изменений необходима

- для эффективного согласования распределённых данных,
- для быстрого восстановления при поломках,
- для ускорения отладки нового кода,
- для упрощения расследования вторжений и других происшествий,
- для повышения ответственности персонала.

Последние две цели требуют гарантий неподдельности истории. Эти гарантии нуждаются нуждаются в комплексной поддержки истории изменений *на более низком уровне*, чем механизмы разрешение конфликтов доступа и механизмы обеспечения прикладной функциональности.

Доступ к физическим ресурсам (процессоры, память) обеспечивается ретроспективной СУБД [11], задачей которой является сохранение истории изменений объектов (в частности исполняемого кода, системной конфигурации и состояния процессов репликации и обработки данных) и обеспечение быстрого доступа к доступным прошлым состояниям.

Экономное сохранение истории достигается разделением данных на *небольшие независимо изменяемые части*. Например, для веб-сайта организации, управляемого CMS, должны сохраняться история изменений шаблонов фрагментов страниц, версии фрагментов заполнений, версии исполняемых процедур, обеспечивающих формирование страницы на этой основе (об истории активности пользователей разговор особый).

Пример: *За два года существования сайта каждый из этих элементов изменяется в среднем пять раз (редкие 100, многие 1), а средний размер элемента превышает длину штампа времени и текущий размер информации сайта 100 Mb.*

Общий объём всех версий исходных файлов, помеченных моментом сохранения, составляет

$$(100 + 100) \times 5 = 1000 \text{ Mb}.$$

Объём выданных пользователям веб-страничек, которые получаются разными сочетаниями исходных файлов и шаблонов, может на порядки превосходить эту цифру.

Для быстрого доступа к любой страничке из прошлого нужно лишь добыть актуальное на этот момент сочетание файлов и шаблонов, что несложно организовать с логарифмически растущими с объёмом издержками [10]. Много резервных копий системы не потребуется, поскольку вся история остаётся нетронутой и сможет пропасть только вместе с используемыми данными.

Пример показывает возможность хранения высоко доступной полной истории изменений при вполне умеренном росте ресурсозатрат.

Даже для сложных систем может оказаться достаточно обеспечить прозрачную логику сборки и сохранять только версии исходной информации и результаты недетерминированных процедур.

Если ненужные версии данных занимают слишком много места, часть давно неактуальной информации может автоматически удаляться [15], создавая *белые пятна* истории на оси времени — промежутки, состоящие из моментов времени, информация на которые недоступна. Этот процесс мог бы проводиться автоматически субоптимально [16], оставляя относительный размер каждого белого пятна близким к минимально возможному при заданном ограничении общего объёма хранилища.

Выводы

Описан подход к информационному моделированию, основанный на доступности неподдельного прошлого и реактивном программировании. Теоретически подход одновременно обеспечивает:

- Возможность неограниченного обновления концептуальной модели, оборудования и сервисов;
- Устойчивость к предусмотренным задержкам исполнения и временным потерям связи;
- Быстроту реакции.

Очередным шагом к созданию такой системы станет создание ретроспективной СУБД [11].

Благодарности

Работа проводилась при финансовой поддержке Министерства образования и науки Российской Федерации. Результаты получены благодаря многолетней поддержке со стороны С. М. Абрамова и конференций RCDL. Особая признательность М. Р. Когаловскому за ценные замечания и рекомендации по изложению.

Список литературы

[1] Ji Hong Yan, Chun Hua Feng. Sustainability-Oriented Product Modular Design Using Design Structure Matrix (DSM) Method. Applied Mechanics and Materials, 2011, pp.1468-1471

[2] Betty H.C. Cheng, Rogério de Lemos, Holger Giese, Paola Inverardi, Jeff Magee Eds. Software Engineering for Self-Adaptive Systems: A Research Roadmap B.H.C. Cheng et al. (Eds.): Self-Adaptive Systems, LNCS 5525, 2009, pp. 1–26.

[3] A. Metzger, K. Pohl, M. Papazoglou, E. Di Nitto, A. Marconi, D. Karastoyanova, “Research challenges on adaptive software and services in the future internet: Towards an scube research roadmap,” ICSE 2012.

[4] Eric Brewer, Towards Robust Distributed Systems, Proceedings of the Nineteenth Annual ACM Symposium on Principles of Distributed Computing, 2000, p. 7.

[5] Кузнецов С.Д. Транзакционные параллельные СУБД: новая волна `http://citforum.ru/database/articles/kuz_oltp_2010/` 2010.

[6] Seth Gilbert, Nancy Lynch. Brewer’s conjecture and the feasibility of consistent, available, partition-tolerant web services. ACM SIGACT News, **33**(2), 2002, pp. 51-59.

[7] D. Sun and C. Sun, Operation Context and Context-based Operational Transformation. CSCW 2006. pp. 279–288.

[8] Goldman, M., Little, G., and Miller, R. C. Real-time collaborative coding in a web IDE. UIST 2011, pp. 155–164.

[9] С.М. Абрамов, С.В. Знаменский, Н.С. Живчикова, А.В. Котомин, Е.В. Титова Информационная система для разработки технологий организации сложной совместной деятельности. RCDL-2009, с. 186-192.

[10] С. В. Знаменский. Гибкая основа информационной системы для обучения. RCDL-2010, с. 451-460.

[11] С. В. Знаменский. Ретроспективная основа совместной реорганизации сложных информационных ресурсов. RCDL-2011, с. 93-101

[12] L. Bauer, C. Braun, M.E. Imhof, M.A. Kochte, H. Zhang, H.-J. Wunderlich, J. Henkel OTERA: Online Test Strategies for Reliable Reconfigurable Architectures. AHS12, 2012, pp. 38-45.

[13] Angelo Corsaro, Douglas C. Schmidt. The Data Distribution Service – The Communication Middleware Fabric for Scalable and Extensible Systems-of-Systems. System of Systems, 2012.

[14] Крюков В.А. Операционные системы распределенных вычислительных систем. Курс лекций (4^{й} курс факультета ВМиК МГУ). `http://parallel.ru/krukov/`.

[15] С. В. Знаменский. Глобальная идентификация данных в долговременной перспективе. Программные системы: теория и приложения 2012. Т. 3, № 2(11), с. 77–88.

[16] С. В. Знаменский. Показатели эффективности расписания резервного копирования. Программные системы: теория и приложения 2012. Т. 3, № 2(11), с. 51–60.

On the way to new information systems theory

Sergej Znamenskii

The long-lived evolving system that can adapt to changing requirements and withstand unforeseen threats, must be resilient to local losses and temporal delays of information. Well-known Brewer CAP theorem denies the possibility of such systems.

New foundations of information systems allow us to construct a counterexample. This construction theoretically proves the possibility to implement the information system with a hundred and more times better quality than any possible implementation of the standard methods.

The rough sketch of the system architecture is given for general purpose application with unlimitly changeable both the functionality and the conceptual model and also the physical implementation.