Метки времени для глобальной идентификации версий

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

Аннотация

Децентрализованное управление резервированием, локальными перезапусками и обновлением исполняемого в распределённой компьютерной системе кода придаёт системе такие качества интернета как высокая доступность и катастрофоустойчивость. Система становится эклектичной, подверженной потере централизованного управления и непротиворечивости.

Однако только эклектичные системы дают возможность опереться на децентрализацию разработок для решения сложных междисциплинарных задач.

Описана проблема обеспечения доступности и согласованности данных в эклектичных компьютерных системах, решаемая идентификацией версий данных временем изменения.

Предлагаемая полиритмичная система автоматического версионирования объектов разделяемой памяти включает в себя способ компактного единообразного представления меток времени с разным разрешением и способ обеспечения согласованности информации на основе таких меток.

Введение

Для повышения надёжности и качества обслуживания мультипроцессорных компьютерных систем широко используется децентрализованное администрирование.

Без объявления технологических перерывов в обслуживании во многих системах проводятся локальные перезапуски подсистем, переключения на резервные сервисы, автоматические обновления исполняемого кода. Нередко используются резервные копии данных и откатываются автоматические обновления.

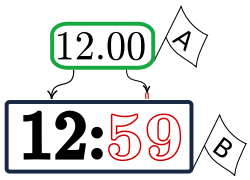

Рис. 1: Пример асинхронного сбоя (*устаревшие данные помечены красным контуром*)

Децентрализация управления потоками данных создаёт проблему системных нарушений порядка событий. Рисунок 1 поясняет её на примере простейшей сети из двух узлов, расположенных в удалении друг от друга, связанных двумя каналами.

Узел A выдаёт по одному каналу текущий час 0,1, ..., 23, 0, ..., а по другому - текущие минуты: 0... 59, 0,...

Узел B это часы, запрашивающие данные у A и выдающие время.

Мы будем считать, что все составляющие системы и каналы связи работают безупречно. Это не отменяет возможных задержек изменчивой продолжительности, неизбежных при передаче данных между узлами. Поэтому около полудня возможны показания 12.59 или 11.00 с ошибкой на час.

Пример иллюстрирует особый класс сбоев в компьютерных системах, связанный с нарушением синхронности процессов. Сбой такого рода способен разрушать связи родственных данных в сложных распределённых системах. В роли часов и минут могут оказаться любые синхронно изменяющиеся данные при возможности распространения по разным путям. Под угрозой разрушения оказываются не только связи данных наблюдений или их обработки, но и связи фрагментов исполняемого кода, метаданных и(или) исходного кода.

Возможность конфликтного сочетания независимых изменений данных может ускользать от внимания разработчиков. Неразумно усложнять систему ради пренебрежимо маловероятной ситуации.

Труды 15-й Всероссийской научной конференции «Электронные библиотеки: перспективные методы и технологии, электронные коллекции» — RCDL-2013, Ярославль, Россия, 14-17 октября 2013 г.

1 Угроза асинхронного сбоя

Определение. *Будем называть* асинхронным сбоем *(asynchronic failure) негативные последствия непредусмотренных задержек или схожих дефектов синхронизации данных.*

Асинхронный сбой выражается в потерях:

согласованности — когда информация различных версий объекта смешивается;

порядка — когда задержки передачи или локального восстановления непроизвольно сменяют новое значение старым;

данных по тайм-ауту при передачах или сложных расчётах.

В результате система может потерять адекватность, продуктивность или отзывчивость вплоть до непригодности.

1.1 Факторы риска

Пример вскрывает факторы асинхронного сбоя:

1. *Неполнота онтологии* системы, допускающей неучтённую связь между данными.
2. *Получение по разным каналам* неявно связанных данных извне системы.
3. *Разброс продолжительности* прохождения информации обозначим δ. Тогда вероятность асинхронного сбоя равна $\frac{\delta}{\tau}$, где τ это средняя продолжительность времени между связанными изменениями данных.

1.2 Характер угрозы

Особое коварство асинхронных сбоев состоит в их способности ускользать от тестирования и в невозможности воспроизведения.

В отличие от случайных сбоев их вероятность пренебрежимо мала. В штатном режиме задержки малы и стабильны, поэтому статистические оценки результатов тестирования асинхронных сбоев не замечают. В нештатных ситуациях временные локальные перезагрузки, потери, очереди, блокировки местами удлиняют задержки обработки или передачи данных в многие тысячи раз, создавая практически неповторимые сочетания локальных задержек, приводящие к асинхронным сбоям.

Диагностика и воспроизведение затруднены отсутствием достаточно полных сведений о временных локальных перегрузках.

В результате, спроектированная и испытанная по всем правилам система сбоит в нештатной ситуации так, что воспроизвести дефект в лабораторных условиях оказывается практически невозможным.

Такая скрытность асинхронных сбоев выводит их за рамки современной теории надёжности, разделяющей дефекты технической системы на случайные, вероятность которых можно оценить с помощью теории и предварительных испытаний, и систематические, выявляемые тщательным тестированием компонент и системы в целом. Получается, что достаточно сложная распределённая система, даже построенная по всем канонам теории надёжности на основе действующих стандартов, требует особых архитектурных решений чтобы не оказаться фатально подверженной асинхронным сбоям [1].

2 Подходы

Теория и практика разработки информационных систем настоятельно рекомендуют базироваться на стандартах [2], основанных на математической теории предикатов и надёжно проверенных широкой практикой создания систем вчерашнего и сегодняшнего дня. Рациональный подход к обеспечению качества информации имеет центральным стержнем

1. Тщательную централизованную проработку логической модели системы.
2. Чёткую организация разработки, точно следующей выверенной логической модели.

В каждый момент времени система имеет формально непротиворечивое в этой модели состояние, которое моментально изменяется на новое столь же логически целостное состояние.

Это означает, что любое событие непосредственно порождает транзакцию, как бы мгновенно вносящую все сопутствующие изменения во все части системы. Параллельно могут выполняться только такие транзакции, которые никак не взаимодействуют по данным. Результат должен быть таким, как если бы все транзакции выполнялись последовательно, что проблематично [3, 4]. Такая изоляция строгих транзакций имеет целью создание для программиста иллюзии (view serializability), что он работает с системой один, чтобы избавить его от сложностей конкурентного взаимодействия. Во многих практических задачах такая иллюзия допустима и рациональный подход оказывается вне конкуренции.

2.1 Ограниченность рационального подхода

Ряд востребованных в ближайшем будущем приложений страдает от принципиальных ограничений рационального подхода:

- Единый лог транзакций [5] **ограничивает масштабируемость**.
- Для системы, которая по сути должна базироваться на неустойчивой сети (например, межпланетные или армейские

системы) **централизованная обработка данных** неприемлема: временно теряющие связь части очевидно должны продолжать функционировать корректно и по возможности эффективно. Распространённое мнение о том, что известная теорема CAP Брювера [6] противоречит возможности создания таких систем, ошибочно [7, 8].

- Гибкость, адаптивность и другие полезные свойства в полной мере реализуемы лишь многими сильными независимыми командами разработчиков при демократическом взаимодействии. **Жёсткий упор на централизацию разработки и сопровождения** резко ограничивает допустимую сложность системы одной головой генерального конструктора [9].
- **Неизменность модели данных** не свойственна долгоживущим системам [10].
- **Монополия централизованной разработки** ограничивает пригодность систем к независимой локальной модификации (информационная система крупной корпорации, межгосударственная система взаимной безопасности, интеллектуальная самовосстанавливающаяся [11] или эволюционирующая [12] системы).

Любое отклонение от рационального подхода обостряет проблему изучения и предупреждения асинхронных сбоев.

2.2 Борьба с задержками

Последний фактор риска — неравномерность задержек — уменьшается такими мерами как

- Использование новейших технологий ускорения обмена и обработки данных.
- Упреждающая балансировка ресурсов, снижающая риски перегрузок [13].
- Поддержание запаса системных ресурсов, обеспечивающего устойчивость к перегрузкам.

Эти меры снижают риск возможно в сотни раз кроме систем, которые (само)восстанавливаются или реконструируются в ходе эксплуатации: восстановление всегда означает существенную неискоренимую задержку.

Если возможно гарантировано ограничить время задержки малой величиной, то риск асинхронных сбоев можно полностью исключён гистерезисной фильтрацией неустойчивых изменений. Для этого результаты каждого запроса должны сверяться с предыдущим показанием. Если оно изменилось, то надо скажем каждую секунду или с другим периодом Δ запрашивать время снова и пока полученное значение не повторится, считать его неустойчивым и не показывать. Если разница в задержках δ не превысит Δ, то такой фильтр исключит ошибку.

Для полного подавления ошибок требуется $\Delta \gg \delta$. При этом часы будут всегда отставать не менее чем на Δ. Уменьшение Δ повысит риск ошибки часов, увеличение усилит их отставание. Если же ускорение достигнуто оптимизацией в штатных ситуациях, то оно вправе не сработать в особом случае, задержка превысит гистерезисную и сбой произойдёт.

Таким образом, гистерезисная фильтрация полезна лишь на входе системы.

Остаётся путь версионирования данных эклектичной системы.

3 Эклектичность

Эклектичность модельного примера лишает эффективности верифицированные протоколы обмена и обработки информации и строгих транзакций.

3.1 Апология эклектичности

Популярная идея единой универсальной онтологии, на которой базируются стандарты и теория разработки информационных систем, проекты Semantic Web, Web 2.0 и Web 3.0, к сожалению принципиально не реализуема. Теорема Гёделя о неполноте подсказывает, что полная непротиворечивая онтология возможна лишь для примитивных систем.

Чтобы показать, что онтологии изменчивы в любой практически значимой области, рассмотрим символ астрономической точности — измерение времени.

В сутках 24 часа. Однако в системе, учитывающей переход на летнее поясное время, всё гораздо сложнее и количество часов в сутках иногда оказываться 23 или 25, что зависит от административного подчинения. С секундами в минуте сложнее. В последнюю минуту полугодия иногда (но почти каждый год) по итогам астрономических наблюдений вводится дополнительная високосная 61-я секунда. Никто не знает, будет ли последняя секунда следующего года високосной. Более того, Международный астрономический союз всерьёз обсуждает возможности отмены високосных секунд в частности с заменой их високосными часами.

Даже в математике общеизвестные факты уточняются с новых позиций. Пифагорейцы хранили в тайне иррациональность корня из двух поскольку это противоречило понятию числа как отношения. Ещё до рождения Христова было известно, что сумма углов в треугольнике равна развёрнутому, а квадрат любого числа положителен. Всё это неверно в геометриях

Лобачевского и Пуанкаре и алгебре комплексных чисел. В учебниках по математическому анализу написано, что разрывные функции производных не имеют. Но теория обобщённых функций изучает их производные.

В предметных областях, отличных от математики и астрономии, уточнение онтологии ещё быстрее выводит на передний край науки и научно-технической политики и становится зыбким и неустойчивым. В частности, это верно и для сетевых технологий [14].

Безупречность онтологии оказывается жёстко ограниченной во времени и привязанной к конкретной сфере приложений.

Правильное понимание недолговечно

Но именно правильное понимание является основой стандартов разработки компьютерных систем. Вывод парадоксален и неутешителен: *императивные компьютерные системы в обозримой перспективе обречены продолжать разочаровывать пользователей* [15].

Выход подсказывает общепризнанные принципы разделения труда и разделения зон внимания (separation of concerns). Будущее за надёжными согласованно развивающимися взаимопроникающими системами, улучшаемыми независимыми группами, компетентными в своих областях [16–18]. Такие системы правильно называть эклектичными. Интернет в целом — это пример эклектичной системы.

Негативное отношение, закрепившееся за термином и системами, исходит от наивной мечты об абсолютном знании и отсутствия теории, технологий и примеров согласованных эклектичных систем.

Не исключено, что технологии эклектичных систем откроют путь к эффективной децентрализованной разработке на фоне постоянно доступного качественного сервиса.

Подобно тому, как объединив творческие силы мира Википедия превзошла по широте, полноте, доступности, актуальности и популярности все лучшие энциклопедии мира, совместное творчество независимых групп разработчиков [21] способно привести к результату, превосходящему лучшие ожидания [22]. Речь идёт о качественно превосходной точности, надёжности и дружелюбности взаимодействующих компьютерных систем, о самовосстанавливающихся, эволюционирующих и мультиагентных системах, о качественном скачке функциональности, надёжности и удобства пользовательских интерфейсов.

3.2 Модель эклектичной системы

Эклектичные системы не похожи на конечные автоматы: их состояние меняется не мгновенно. Это ближе к реальности распределённых систем, в которых допускаются переключение на резервный сервис, перезапуск локального сервиса, восстановление из резервной копии и иные внезапные приостановки активности. Для эклектичных систем (ЭС) более адекватной представляется модель, основанная на идеях контекстной автономности из [19] и [20]:

- ЭС может состоять из подсистем, являющихся ЭС, и входить в другие ЭС.
- ЭС может считывать информацию датчиков, пользовательских интерфейсов и иных источников.
- ЭС предоставляет доступ
  - (1) к актуальной на указанный момент версии каждого выработанного ею информационного объекта,
  - (2) к описаниям белых пятен истории, т.е. промежутков времени переходов к согласованным состояниям в указанном контексте данных.

Проблема распределения ресурсов между направлениями и темпами обновления и поддержкой функционирующих сервисов сложна и многопланова, но делится на очевидные части:

1. Уточнение стратегии развития.
2. Определение приоритетов внутренних процессов.
3. Организация обработки информации в соответствии с установленными приоритетами.

Рассмотрение первых двух частей выходит за рамки настоящей статьи. Её задача описать систему приоретизированной обработки информации, полностью исключающую асинхронные сбои.

3.3 Ретроспективность и обновления

Любые данные поступают в эклектичную систему с указанием времени. Полезны протоколы своевременного получения информации об изменениях (comet).

Если в источнике есть информация о времени актуальности данных, то она должна быть корректно использована. Иначе идентификаторы версий могут быть сгенерированы на основе текущего времени.

Вывод данных из эклектической системы по запросу может в основном осуществляться двумя способами:

Проспективный означает ожидание когда появится версия, актуальная на момент запроса или более поздний.

Ретроспективный [19] означает немедленную выдачу последней на момент запроса версии данных.

Проспективный вариант привычно включает ожидание завершения обработки всех внесённых на момент запроса данных. Ретроспективный непривычен тем, что исключает ожидания обработки. Это напоминает чтение данных их кэша или реакцию поисковых серверов интернет.

Механизм автоматического версионирования заменяет иллюзию изоляции необходимостью корректной идентификации версий объектов. Происходившее в системе становится доступным не через лог, а через машину времени, показывающую входные и выходные данные в динамике обработки. Обработка может программироваться на любых языках программирования, а её корректность диагностироваться и отлаживаться средствами этих языков. Требуется лишь чтобы выходные данные правильно соответствовали входным.

Гладкое обновление естественно будет происходить поэтапно:

- подготовка и отладка новой версии исполняемого кода системы,
- запуск его в параллель с работающим с автоматическим извещениями о расхождениях в выходных данных,
- включение в качестве бета-версии с возможностью мгновенного переключения между версиями,
- бета-тестирование потребителями,
- получение статуса базовой версии,
- получение статуса резервной версии,
- отключение по причине длительной невостребованности.

От программистов потребуется не только тестирование своей системы, но и сравнительное тестирование версий систем-поставщиков входной информации.

Организация исполнения в эклектичных системах может основываться на следующей схеме:

- Автоматическое версионирование закладывается на низшем уровне, а к нему адаптируются структуры данных и интерфейсы.
- Использование штампа времени в качестве метки версии обеспечивает синхронность данных разной природы и происхождения.
- При обмене информацией между узлами передаются фрагменты истории изменений, а при обработке совместно обрабатываются или показываются данные, относящиеся только к одной версии.

На пути реализации этой идеи лежат многочисленные подводные камни.

4 Трудности автоматического версионирования

4.1 Одновременность

Первая проблема состоит в определении того, какие данные считать одновременно актуальными, то есть относящимися к одному состоянию системы.

Классические исследования показали запредельную техническую сложность синхронизации всех изменений в распределённой компьютерной системе. Система рассматривалась при этом как набор конечных автоматов, обменивающихся сообщениями о событиях. Время в ней трактовалось как вспомогательное средство для выстраивания всех событий в единый линейный порядок.

В интересующей нас эклектичной модели время это показания локальных часов, то есть физическая величина, измеренная с некоторой точностью.

Выделить актуальные в общий момент времени данные далеко не просто по многим причинам.

Во-первых, окончание актуальности данного X может быть настолько близко к началу актуальности данного Y, что момент их одновременной актуальности существует лишь с некоторой вероятностью.

Во-вторых, погрешность указания времени Δ может варьироваться от наносекунд для данных быстро протекающих процессов до тысячелетий для археологических данных.

В-третьих, конкретная разница во времени событий может свидетельствовать об одновременности или о неодновременности событий в зависимости от постановки задачи.

В-четвёртых, различаются актуальность в некоторый момент промежутка и актуальность в течение всего промежутка.

В-пятых, проблематично корректно совместить данные, относящиеся к приближенно известной общей границе промежутков шкалы времени с одним из этих промежутков.

В-шестых, случается, что актуальная достоверная информация недоступна и пользователям порой нужен доступ к новейшим неполным или непроверенным данным. В таких ситуациях

разумно предоставлять особый интерфейс к неполной сводной информации.

Нужен прозрачный способ безупречно автоматически выделять одновременно актуальные версии данных. Идентификация моментами времени должна

- быть достаточно универсальной,
- отражать значение и погрешность промежутка актуальности в широких диапазонах,
- позволять быстро и просто собирать последние одновременные данные.

Организация исполнения должна до предела снижать вероятность ситуации, в которой обработка задерживается из-за неспособности системы выделить одновременно актуальные данные.

4.2 Эффективность и алгоритмы

Работа эклектичной системы должна опираться на эффективные алгоритмы решения принципиально новых, ранее не рассматривавшихся задач, таких как

- поддержание и сохранение истории датированных изменений,
- обмен фрагментами истории изменений в семантике глобальной разделяемой памяти,
- отбор согласованных версий входных данных для совместной выдачи или обработки,
- эволюция структур данных во времени.

В последнем пункте соединяются два направления поиска. Во-первых, поддержка истории изменений структур традиционно ведётся без привязки ко времени. Обычная структура в памяти либо не хранит историю изменений объекта, либо хранит фиксированное количество последних изменений, либо (persistent data) хранит полностью, но только порядок событий без привязки ко времени. Последнее практически означает экспоненциальный рост требуемых ресурсов памяти.

Привязка ко времени позволила бы без дублирования информации постоянно поддерживать достаточное число снимков данных (snapshots), синхронно забывая данные промежуточных изменений [23].

Например, можно сохранить все ежечасные снимки прошлого месяца, все ежедневные прошлого года, чтобы существенно сэкономить ресурсы памяти, оставляя доступной для разнообразной обработки (такой как машина времени [24, 26–28]) значимую часть истории.

Во-вторых, это поддержка эволюции структур данных. Для иерархической структуры (скажем, дерево XML-документа) это прежде всего появление промежуточного уровня иерархии. Скажем, была классификация данных по городам России, а понадобилось ввести административные округа и распределить данные городов по ним.

Сейчас такая структурная перестройка требует удаления всех веток изменяющегося уровня и добавления их к добавленным узлам промежуточного уровня. Эта операция нарушает связи между элементами структуры, портит предысторию веток и тем самым делает невозможным гладкое (без перерывов в обслуживании) обновление. Для представления данных, которые должны быть непосредственно доступны с длительной историей структурных изменений, необходима дальнейшая проработка универсальных и эффективных гибких структур данных, поддерживающих эволюцию, см., например, [25].

4.3 Продолжительные изменения и согласованность

Пока обработка использует версии данных, актуальные на общий момент времени, результат однозначно определяется этими данными и идентифицируется тем же временем, угрозы рассогласования данных не возникнет. Поэтому сколько бы ни длилась такая обработка, её результат относится к той же версии данных и должен быть помечен тем же временем, чтобы не нарушить логический порядок.

Угроза рассогласование станет явной только если обработка соединит данные, не актуальные на общий момент времени. Это может случиться и для изменений, помеченных общим временем если одно из данных снова изменилось, а допустимое отставание локальных часов отнесло это событие к следующему моменту.

Если при алгоритмической обработке есть надежда избежать всего этого, то в пользовательских интерфейсах проблема остаётся. Как бы тщательно ни были отфильтрованы для пользователя согласованные данные, привычное использование дополнительных источников актуальной информации может сделать результат неадекватным.

Это придётся учитывать при создании пользовательских интерфейсов. Недостаточно просто принимать все меры к тому, чтобы информация в интерфейсе для пользователя была предельно актуальной и полной. Важно правильно идентифицировать результат. Идентификатор должен остаться прежним лишь в случае, когда пользователь действовал по ранее установленному алгоритму и не использовал сторонней более поздней информации. Иначе при сохранении идентификатор должен корректно отразить возникший разброс времени и вероятен конфликт для разрешения которого потребуются адекватные инструменты.

5 Система и перспективы

5.1 Способ кодирования времени

Шкалой S назовём разбиение числовой прямой на полусегменты равной длины, называемой шагом дискретизации. Шаг дискретизации определяется прикладными задачами и должен быть практически несущественным. Например, для полученных из веб-формы данных не имеет практического значения, нажал ли пользователь кнопку отправки десятой долей секунды раньше или позже.

Правый конец в полусегмент не входит, этим из предыстории исключаются события, на обработку которых не было времени. Базовый пример даёт разбиение на целочисленные промежутки $[n, n+1), n \in \mathbb{Z}$.

Мы будем использовать глобальные отметки времени (Новый год, полночь и т.д.) как разделители. Поскольку потребуется отчёт за год, то каждое событие должно определённо относится либо к прошлому году, либо к будущему. Сложность с событием, произошедшим в момент такого разделителя: допустимая погрешность часов позволяет отнести его в одних подсистемах к предшествующему разделителю полусегменту, а в других к последующему за разделителем.

В таких случаях логически неправомерно производить обработку данных на момент конца года и нужно выбирать момент, на который череда изменений приостановилась. Пользователя однако может интересовать состояние на конец года. По-видимому, лучшее, что может сделать система в этой ситуации, это выдать корректный ответ на близкий момент времени. В ситуации продолжительной высокой интенсивности конфликтующих изменений входных данных, в которой пользователя интересует не только (вероятно отдалённый) момент истины, но заведомо логически небезупречная оперативная оценка текущего состояния дел.

Для получения такой оценки можно игнорировать несущественные расхождения во времени, но при записи результата указать оценку дефекта (времени рассогласованности) τ. Эта оценка должна быть использована при выдаче пользователю предупреждения о размере возможной некорректности данных.

Одновременное использование многих шкал нуждается в их согласованности, означающей что из двух пересекающихся полусегментов один обязательно содержит другой. Обозначим $\mathbb{B} = \{k \cdot 2^l | k, l \in \mathbb{Z}\}$ множество всех двоичных дробей и назовём *бинарным семейством* такое семейство шкал, все сегменты которого имеют вид $[t-\delta, t) = [k \cdot 2^l, (k+1) \cdot 2^l)$, где $k, l \in \mathbb{Z}$.

Предложение 1. *Полусегмент бинарного семейства однозначно определяется своей серединой. Множество серединок таких полусегментов совпадает с* $\mathbb{B}$

Простота идентификации полусегментов бинарного семейства делает их привлекательными для использования в качестве идентификаторов моментов событий с учётом погрешности определения времени. Например, момент события, предшествовавшего моменту 2 с точностью примерно 0.1%, идентифицируется как $1.11111111111_{(2)}$.

Для ускорения поиска нужной версии в бинарном дереве важно чтобы идентификаторы были лексикографически упорядочены по актуальности. Несмотря на простоту и естественность, порядок при таком представлении не согласуется с желаемым лексикографическим: правый конец полусегмента t существенно более значим, чем левый $t - \delta$.

К сожалению, количество дней в году и другие соотношения мер времени не являются степенями двойки (и, к тому же, не все являются заранее фиксированными числами). Моменты времени практически задаются конечными наборами $t = (Y, M, D, h, m, s, \mu_0, \mu_1, ...)$ из указаний года Y, месяца M, дня D, часа h, минут m, секунд s, миллисекунд μ_0, микросекунд μ_1 и т.д.

Предложение 2. *Функция*

$$\begin{aligned} x(t) = {} & 2^{-3}Y + 2^{-7}(M-1) + 2^{-12}(D-1) \\ & + 2^{-18}h + 2^{-24}m + 2^{-30}s \\ & + \sum_{k \geqslant 0} 2^{-30-10k}\mu_k \end{aligned}$$

монотонно и однозначно представляет моменты времени двоичными дробями.

Замечание 1. *Функция теоремы 2 переводит привычные шкалы времени (годы, месяцы, дни, недели, часы и т. д) в шкалы бинарного семейства.*

Например, 13:35 30 июня 2014 года с секундной точностью идентифицируется двоичной дробью

$$\begin{aligned} x = {} & 2014 \cdot 2^{-3} + 29 \cdot 2^{-7} + 13 \cdot 2^{-12} \\ & + 35 \cdot 2^{-18} + 0.5 \cdot 2^{-24} + 2^{-31} \\ = {} & 11111011.110\,0101\,11101\,01101\,100010\,111111\,1_{(2)} \\ = {} & 2^{11} - 100.001\,1010\,00010\,10010\,011101\,000000\,1_{(2)}. \end{aligned}$$

Следующее утверждение описывает ускоряющий сравнение переход от чисел произвольной длины со знаком к битовым массивам (строкам произвольной длины).

Предложение 3. *Кусочно-линейная функция*

$$B(x) = \frac{1}{2} + \frac{\operatorname{sgn}(x)}{2}\left(1 - \frac{3}{2^{k(x)+1}} - \frac{1+|x|}{2^{2k(x)}}\right), \text{ где}$$
$$k(x) = \operatorname{ceil}\left(\log_2(1+|x|)\right),$$

непрерывна, монотонна и взаимно-однозначно отображает множество $\mathbb{B}$ *всех двоичных дробей на* $(0,1)\cap\mathbb{B}$.

Для промежутка с 1984 по 2112 годы это выражение удобно масштабируется $B(x(t)-2^{11})$ и упрощается до $B(x(t)-2^{11}=\frac{1}{2}+\frac{x}{(}t)2$ потому что $|x-1|<1$. Для длительности δ удобнее суточный масштаб времени $2B(-2^1 2x(\delta))-1$, а для дефекта τ минутный $2B(-2^2 4x(\delta))-1$.

Замечание 2. *Обратная к* $y=B(x)$ *функция вычисляется по формуле*

$$x = 1+\operatorname{sgn}(2y-1)\left(3\alpha-2\alpha^2|2y-1|-1\right),\ \textit{где}$$
$$\alpha(y)=2^{k(y)-1},$$
$$\tilde{k}(y)=-\operatorname{floor}\left(\log_2\left(1-|2y-1|\right)\right).$$

Замечание 3. *Количество значащих цифр после запятой в двоичной записи* $B(x)$ *равно сумме* $|k(x)|$ *и количества цифр в записи двоичной дроби* $|x|+1$, *взятого без заключительных нулей.*

В частности, образ $B(x(t)-2^{11})$ 34-значной двоичной дроби из последнего примера длиннее записи самой дроби на два знака поскольку для неё $k(x)=2$.

Для полусегментов $[t-\delta,t)$ и дефектов τ требуется монотонно закодировать лексикографически упорядоченные вектора из записи двоичных дробей $B(x(t)),B(2^{12}x(-\delta))B(2^{24}x(-\tau)),)$.

Замечание 4. *Можно монотонно закодировать лексикографически упорядоченные вектора из записи двоичных дробей* $(x_1,x_2,x_3)\in[0,1]$ *строками из* $\sum_{i=1}^{3}\operatorname{ceil}\left(\frac{\operatorname{length}(x_i)}{6}\right)+2$ *байт, принимающих 65 различных значений.*

Для этого битовый массив разбивается на группы по 6 бит, последняя дополняется нулями, каждая группа бит кодируется байтом (Base64) и x_i представляется строкой (в нашем примере длины 6 байт). Пусть «!» - байт с меньшим кодом, чем использованные в Base64. Тогда конкатенация $x_1.\text{«!»}.x_2.\text{«!»}.x_3$ даёт эффективную кодировку для замечания 4.

История изменений информационного объекта предстаёт упорядоченным по актуальности списком версий. Представление этого списка в виде структуры B^+Tree [31, 32] обеспечивает быстрый доступ к актуальной на любой заданный момент версии. Версии с недостаточно высокими значениями при этом быстро пропускаются.

5.2 Способ организации исполнения

Функционирование эклектичной системы распадается на процессы обработки данных. Это как прикладные сервисные процессы, так и

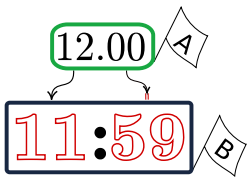

Рис. 2: Устойчивость системы к асинхронным сбоям (*устаревшие данные помечены контуром*)

системные, обеспечивающие сопровождение согласования стратегии развития, планирования и реализации обновлений, мониторинга состояния системы и уточнения приоритетов исполнения. Системная архитектура должна незаметно для прикладных программистов обеспечивать, что старый согласованный результат останется в действии и будет обновлён лишь когда полностью соберётся требуемая для согласованного обновления информация, см. рисунок 2.

Процессу доступны упорядоченные списки версий каждого объекта входных данных для обработки. Системный планировщик определяет идентификатор версии объектов результата обработки и обеспечивает обработчику быстрый доступ к версиям объектов, которые должны быть включены в обработку. Прикладная программа-обработчик (бинарный код или скрипт на любом языке программирования) обрабатывает пакет изменившихся входных данных. Пакетная обработка известна как средство резкого ускорения обработки данных [30].

Запуск очередной обработки данных осуществляется системным планировщиком в установленном для процесса ритме (аналогично [29]). Если предыдущая обработка этого процесса не завершилась, то запуск отменяется.

Результат может быть сохранён с версией (t,δ,τ) если версия любого использованного при обработке объекта o удовлетворяет условию, зависящему от положительности τ.

$\tau>0$, $t-\delta\le t_o-\delta<t_o+\tau_o\le t+\tau$;

$\tau=0$, $t-\delta\le t_o-\delta<t_o=t_o+\tau_o\le t$ и некоторое время после t_o; объект не менялся

При запуске обработки правый конец отрезка-идентификатора результата выбирается на шкале с ритмом запуска обработки так, чтобы оказаться на шкале с наибольшим шагом. Идентификатор всегда должен быть больше предыдущего сохранения, но меньше текущего момента.

Длина полусегмента δ изначально берётся вдвое меньше, чем было в предыдущей сохранённой версии результата этого процесса и $\tau=0$. Если для такого результата не хватает актуальных входных данных, то δ увеличивается вдвое, если не помогает, то ещё вдвое, а если и это не помогает и есть другие варианты

выбора для t, то проверяются они и если версия не выбирается и есть давно необработанные входные данные то появляется $\tau > 0$,

Последнее обеспечивает мягкую деградацию качества сервисов при перегрузках и аномальных задержках (передачи и исполнения) и незамедлительное полное восстановление качества сервисов при исчезновении перегрузок и задержек.

6 Выводы

1. Эклектичные компьютерные системы обещают качественное превосходство по важнейшим показателям.
2. Конфликты версий неизбежны внутри сложных эклектичных систем.
3. Автоматическое версионирование на основе меток времени открывает путь к исключению таких конфликтов.
4. В начале этого пути теоретическая проработка и создание новых алгоритмов и структур данных.
5. Описаны способ маркировки моментов событий и способ корректной организации исполнения в эклектичных системах.

Список литературы

[1] M. Ghafari, P. Jamshidi, S. Shahbazi, H. Haghighi. An architectural approach to ensure globally consistent dynamic reconfiguration of component-based systems // Proceedings of the 15th International ACM SIGSOFT Symposium on Component-based Software Engineering (CBSE'2012). – Bertinoro, Italy, June 2012.

[2] J. Kasser, D.K. Hitchins. Unifying systems engineering: Seven principles for systems engineered solution systems // The 20th International Symposium of the INCOSE. – Denver, 2011. – P. 1–11.

[3] H. Berenson, P. Bernstein, J. Gray, J. Melton, E. O'Neil, P. O'Neil. A Critique of ANSI SQL Isolation Levels // Proc. of the ACM SIGMOD International Conference on Management of Data. – 1995. – P. 1–10.

[4] R. Normann, L.T. Østby. A theoretical study of 'snapshot isolation'// ICDT. – 2010.– P. 44–49.

[5] С. Д. Кузнецов. Транзакционные параллельные СУБД: новая волна // Труды Института системного программирования РАН. – 2011. — Т. 20 – http://cyberleninka.ru/article/n/tranzaktsionnye-parallelnye-subd-novaya-volna.

[6] S. Gilbert, N.A. Lynch. Perspectives on the CAP Theorem // IEEE Computer Society. – 2012.– P. 30–36.

[7] K.P. Birman et al. Overcoming CAP with consistent soft-state replication // Computer. — 2012. – Т. 45. – № 2. — С. 50–58.

[8] С.В. Знаменский. Ретроспективная основа распределённой памяти для изменчивой вычислительной среды // Материалы VI Международной конференции «Параллельные вычисления и задачи управления» (PACO'2012). – М.: ИПУ РАН, 2012. – Т. 2. – С. 259–272.

[9] I. Sommerville, D. Cliff, R. Calinescu, J. Keen, T. Kelly, M. Kwiatkowska, J. McDermid, R. Paige. Large-scale Complex IT Systems // Communications of the ACM. – 2012. – V. 55, Issue 7. – P. 71–77.

[10] V. Andrikopoulos, S. Benbernou, M.P. Papazoglou. On the evolution of services // IEEE Transactions on Software Engineering. – 2012. – V. 38, Issue 3. – P. 609–628.

[11] E. Hollnagel. From protection to resilience: Changing views on how to achieve safety // 8th International Symposium of the Australian Aviation Psychology Association. – Sydney, Australia, 2008.

[12] I. Fehérvári, W. Elmenreich. Evolutionary methods in self-organizing system design // Proceedings of the 2009 International Conference on Genetic and Evolutionary Methods. — 2009. – P. 10–15.

[13] T. Chalermarrewong, S. See, T. Achalakul. Parameter Prediction in Fault Management Framework // Proceedings of The International Symposium on Grids and Clouds (ICGC 2012). – 26 February–2 March 2012, Taipei, Taiwan. – http://pos.sissa.it/archive/conferences/153/005/ISGC%202012_005.pdf – 2012. — Т. 1. – P. 5.

[14] C.M. Kelty. Conceiving Open Systems // Wash. UJL & Pol'y. – 2009. – Т. 30. – P. 139.

[15] Y. Merali, T. Papadopoulos, T. Nadkarni. Information systems strategy: Past, present, future? // J. Strateg. Inform. Syst. – 2012. – http://dx.doi.org/10.1016/j.jsis.2012.04.002

[16] P. Feiler, R.P. Gabriel, J. Goodenough, R. Linger, T. Longstaff, R. Kazman, M. Klein, L. Northrop, D. Schmidt, K. Sullivan, K. Wallnau. Ultra-Large-Scale Systems: The Software Challenge of the Future // Technical Report. – Carnegie Mellon University Software Engineering Institute. – 2006.

[17] L. M. Northrop. Ultra-Large-Scale Systems: Scale Changes Everything // SMART Ultra-Large-Scale Systems Forum. – March 6, 2008.

[18] C. Ncube. On the Engineering of Systems of Systems: key challenges for the requirements engineering community // Requirements Engineering for Systems, Services and Systems-of-Systems (RESS). – Aug. 2011. – P. 70–73.

[19] С. В. Знаменский. Ретроспективная основа совместной реорганизации сложных информационных ресурсов // Электронные библиотеки: перспективные методы и технологии, электронные коллекции». – RCDL-2011. – Воронеж, Воронежский госуниверситет, 2011. – С. 93–101. – http://ceur-ws.org/Vol-803/paper10.pdf

[20] P. Helland, D. Haderle. Engagements: Building Eventually ACiD Business Transactions // 6th Biennial Conference on Innovative Data Systems Research (CIDR '13). – January 6–9, 2013. – 12 pp.

[21] S. R. Jeffery, L. Sun, M. DeLand, N. Pendar, R. Barber, A. Galdi. Arnold: Declarative Crowd-Machine Data Integration // 6th Biennial Conference on Innovative Data Systems Research (CIDR '13). – January 6-9, 2013. – 8 pp.

[22] W. Van Osch, M. Avital. Collective Generativity: The Emergence of IT-Induced Mass Innovation // Proceedings of JAIS Theory Development Workshop. – http://sprouts.aisnet.org/9-54.

[23] J. Stender, M. Hogqvist, B. Kolbeck. Loosely time-synchronized snapshots in object-based file systems // Performance Computing and Communications Conference (IPCCC). – IEEE 29th International. – IEEE, 2010. — P. 188–197.

[24] L. Shrira, C. van Ingen, R. Shaull. Time travel in the virtualized past: Cheap fares and first class seats // Haifa Systems and Storage Conference. – SYSTOR 2007.

[25] Yu. Rogozov, A. Sviridov, S. Kucherov. Meta-Database for the Information Systems Development Platform // 6th Spring/Summer Young Researchers' Colloquium on Software Engineering (SYRCoSE 2012), – P. 164–171.

[26] P. Ta-Shma, G. Laden, M. Ben-Yehuda. Factor: Virtual machine time travel using continuous data protection and checkpointing // Operating Systems Review. – 2008. – V. 42, Issue 1. – P. 127–134.

[27] Ragib Hasan. Trustworthy History and Provenance for Files and Databases // PhD thesis, University of Illinois at Urbana-Champaign. – Urbana, Illinois, October 2009.

[28] G. Fourny, D. Florescu, D. Kossmann. A time machine for XML // Technical report № 734. – ETH Zurich, Switzerland, 2011.

[29] J. Wang, K. Y. Lam, S. Han, S. H. Son, A. K. Mok. On Co-Scheduling of Periodic Update and Application Transactions with Fixed Priority Assignment for Real-Time Monitoring // Advanced Information Networking and Applications (AINA). – 2012 IEEE 26th International Conference. – 2012, March. – P. 253–260.

[30] A. Thomson, T. Diamond, S. C. Weng, K. Ren, P. Shao, D. J. Abadi. Calvin: fast distributed transactions for partitioned database systems // Proceedings of the 2012 international conference on Management of Data. – 2012, May. – P. 1–12.

[31] Г. М. Адельсон-Вельский, Е. М. Ландис. Один алгоритм организации информации // Доклады АН СССР. — 1962. — Т. 146, № 2. -- С. 263–266.

[32] L. Jiang, B. Salzberg, D. Lomet, M. Barrena. The BT-Tree: A Branched and Temporal Access Method // VLDB'00 Proceedings. – 2000. – P. 451–460.

Timestamps for a global version identification

Sergej Znamenskii

Decentralized management of data restore, local restarts and code updates for a distributed computer system gives the system some of such the Internet's qualities as high availability and disaster recovery. The system becomes eclectic, prone to loss of centralized control and consistency.

However, only the eclectic systems make it possible to rely on the decentralization of development to solve complex interdisciplinary problems

The article describes the problem of ensuring the data consistency and availability in eclectic computer systems to be solved with timestamp-based versioning.

Proposed polyrythmic system of automatic versioning of shared memory objects includes short uniform presentation of timestamps with different resolution and a timestamps-based way to ensure consistency.