Ретроспективная основа совместной реорганизации сложных информационных ресурсов♣

© С.В. Знаменский

Институт программных систем РАН имени А.К. Айламазяна
svz@latex.pereslavl.ru

Аннотация

Информационные ресурсы, адресованные комплексным проблемам, непрерывно дорабатываются с привлечением широкого круга специалистов и организаций.

Доработка часто нуждается в неожиданных коррекциях структуры ресурса, параллельной разработке и сопоставительном тестировании версий частей, восстановлении ранее доступного.

Обсуждается идея построить такую систему на неизменяемых записях, адресуемых с учётом давности изменений.

Новый подход предположительно должен удешевить и ускорить неограниченную совместную доработку сложных ресурсов.

1. Введение

Часть информационной системы, обладающую в каждый момент времени внутренне согласованным состоянием, принято называть компонентой. Кроме данных, относящихся к состоянию, компонента может содержать незавершённые транзакции. Каждая транзакция должна скачком изменить систему, перевести её в новое согласованное состояние. Компонента может поддерживаться реляционной СУБД, либо иными средствами.

Несмотря на огромное количество теоретических и прикладных разработок, направленных на повышение эффективности механизмов обработки транзакций, компонента может гарантировать ожидаемо быструю реакцию на каждый запрос к системе только в случае прозрачной фиксированной логики без трудоёмких алгоритмов обработки данных. Иначе очередной скачок к новому состоянию порой требует неожиданно много времени.

Для пояснения на примере рассмотрим пользовательский интерфейс интернет-магазина. Изменение заказа доступно как на страничке описания товара, так и в корзине заказанных товаров. Удобно открыть несколько окон с описаниями товаров. При изменении количества в одном окне оно должно измениться во всех остальных, где присутствует возможность изменения. Это изменение должно происходить быстро, в идеале за маленькие доли секунды, и оставаться при открытии в другом браузере или компьютере (при отказе браузера).

По завершению оформления заказа он передаётся в службу формирования. Поскольку у обрабатывающего заказ сотрудника случаются очереди других дел или заказов, то на пересылку не жалко потратить доли минуты.

Заказанный список используется для уточнения профиля пользователя и близости товаров и уточнения рекомендательного сервиса и персонализации пользовательского интерфейса. Эта обработка информации сложна, но не нуждается в более быстрой реакции, чем за доли часа.

Во всех трёх ситуациях замедление в десятки раз не бросается в глаза пользователю.

Если реализацию такого сервиса основывать на одной компоненте, то в минуты обработки медленной транзакции база не сможет обслуживать быструю, что повлечёт непозволительные задержки других пользователей в первой ситуации. Выход мог бы видеться в параллельной обработке транзакций, но это (см., например, [1]) получается лишь для некоторых приложений.

По стандартной технологии (в частности, СОА), проблема решается разбиением на компоненты, обменивающиеся системными сообщениями. В итоге архитектура системы оказывается тесно завязанной на сложную логику конкретного приложения, что резко повышает как стоимость разработки, так и риски архитектурных (и поэтому принципиально неустранимых) ошибок, всплывающих в ходе верификации и сопровождения системы.

1.1 Цель исследований

Целью работы является упрощение разработки систем, направленных на высококачественную информационную поддержку такой совместной творческой деятельности, которая включает в себя совершенствование организации самой этой деятельности. Подобные саморазрабатываемые системы [2] сейчас базируются на СОА и настолько ресурсозатратны в разработке и сопровождении, что доступны лишь немногим десяткам богатейших организаций мира: разнонаправленность потоков информации и изменчивость структур управления данными делают задачу эффективного разде-

Труды 13[й] Всероссийской научной конференции «Электронные библиотеки: перспективные методы и технологии, электронные коллекции» - RCDL'2011, Воронеж, Россия, 2011.

ления сложной требуемой логики на компоненты разрешимой лишь под постоянным контролем крупной команды высококвалифицированных специалистов-разработчиков.

Потребность в устойчиво развивающихся информационных системах (так зачастую называют саморазрабатываемые системы) высока не только в богатейших организациях.

Обсуждаемая в работе гипотеза состоит в том, что эту потребность возможно при существенно более скромной стоимости разработки и владения удовлетворить на основе предлагаемого нестандартного подхода к созданию саморазрабатываемых систем.

Главной целью разработки ставятся задачи организации более сложной совместной деятельности по информационному обслуживанию, в которой пользователи и группы с не очень высоким уровнем квалификации смогут с минимальным риском самостоятельно произвольно перестраивать логику взаимодействия в доверенных им частях сложной информационной системы.

Нестандартность подхода означает, к сожалению, и необычно высокие затраты на создание первой конкурентоспособной реализации (требуются годы на дальнейшие исследования, создание нового инструментария, испытания прототипов...). Это должно окупиться резким упрощением дальнейших сложных разработок. Острую нужду в таких долгосрочных вложениях доказывают общеизвестные крайне неутешительные прогнозы обострения дефицита программистов высокой квалификации.

1.2 Идея подхода

Ключевой идеей является замена механизма системных сообщений между компонентами обращениями к общей ретроспективной СУБД.

Ретроспективной будем называть СУБД, которая принимает все запросы на чтение и изменение данных, параллельно и асинхронно в фоновом режиме обрабатывает изменения, хранит все исходные данные и результаты обработки и в ответ на любой запрос возвращает пользователю ранее обработанные для такого запроса результаты.

Любой запрос на чтение новой информации мгновенно возвращает установившееся в системе прошлое состояние и остается гарантированно воспроизводимым с тем же результатом. Это упрощает отладку кода и расследование инцидентов.

Поломка фонового обработчика воспринимается пользователем как замедление обработки новых данных. Если сопровождающий код программист быстро устраняет неисправность, то пользователи не замечают дефекта.

При отсутствии достаточно свежих результатов по своему запросу может быть организовано оповещение пользователя об их появлении. Отличие от привычных ожиданий результатов обработки в неблокирующем взаимодействии с системой: пользователь может одновременно открывать и параллельно использовать несколько интерфейсов для работы в системе, а иногда может успеть отменить свой ввод до начала его обработки в очереди.

Такой, пока непривычный, интерфейс позволит значимо ускорить работу с системой, задержки в работе которой неустранимы по причине больших расстояний или сложных алгоритмов. При любых замедлениях обработки возрастёт среднее время ожидания обработки изменений, но обслуживание не прекратится, а ввод и выдача готовой информации практически не замедлятся.

Совместная обработка всех поступивших в общем контексте изменений вместо разделения их на отдельные инициированные запросами транзакции упрощает логику реализации и снимает с системы тяжелейшее требование во что бы то ни стало обеспечивать избыточную согласованность [3]. Открывается возможность сочетания живых интерактивных пользовательских интерфейсов с прозрачно реализуемой сложной фоновой обработкой.

На ретроспективной основе могут на более низком уровне реализации, чем логика приложения (а, значит, без усложнения прикладных разработок) быть реализованы и дополнительные возможности. Отладка изменяющейся логики приложения может проводиться на «живых» данных непосредственно в системе незаметно для остальных пользователей. Катастрофоустойчивость системы может быть резко повышена дублированием информации на географически удалённых параллельно работающих серверах; при разрыве связи контексты данных, менявшиеся через другие серверы, замрут до восстановления. При этом целостность данных будет гарантирована с высокой надёжностью, что особенно важно для уникальных результатов совместного творчества специалистов.

Вынос поддержки этих соблазнительных качеств на низкий уровень архитектуры означает резкое расширение возможностей разработчиков сложных прикладных систем, удобных для использования и сопровождения.

Возникают и специфичные проблемы. Хранение всех версий исходных данных и результатов обработки, включая промежуточные, в десятки раз увеличивает требуемые ресурсы физического хранилища. Увеличение объёмов хранения всегда снижает производительность. Чтобы десятки раз не превратились вдруг в тысячи раз, необходима направленная на ресурсосбережение тщательная теоретическая проработка на всех архитектурных уровнях.

2. Область применимости

Чтобы пояснить необходимость рассмотрения нестандартной технологии, рассмотрим простейшую из решаемых ею задач — задачу классификации большого множества ресурсов. Независимо от того, какие ресурсы классифицируются —

публикации, советы по устранению проблем пользователей, описания биологических объектов, товаров или образовательные ресурсы, — можно считать, что каждый объект представлен своим url и рассмотреть задачу построения каталога веб-ресурсов.

2.1 Коллаборативный каталог-классификатор

Обычно большие каталоги-классификаторы строятся и поддерживаются небольшими командами специалистов и содержат до нескольких сотен тысяч ресурсов. Готовые классификаторы, как правило, нуждается в уточнении, расширении (часто на порядки) и в согласовании с другими.

Поднять качество каталога на существенно более высокий уровень можно только путём привлечения к классификации всех заинтересованных специалистов (либо, вообще, всех желающих). Для этого нужно предоставить каждому из них возможность быстро, удобно, эффективно и конструктивно классифицировать ресурсы, исправлять ошибки, видя общее мнение по каждому конкретному вопросу и имея возможность поддержать или оспорить его.

Новичку такая система предстанет каталогом, оптимизированным с учётом всех мнений всех пользователей-разработчиков. Пользователю-разработчику внесённые изменения покажутся несомненно учтёнными на фоне общих изменений.

При наличии хорошего начального наполнения и привлечения достаточного круга пользователей, новых пользователей будут привлекать простота добавления в каталог и изменения его структуры, богатство и оптимальность каталога.

Подобный каталог смог бы в перспективе органично дополнить поисковые сервисы интернет. Пройти по каталогу проработанной структуры часто окажется быстрее и легче, чем правильно сформулировать запрос. Привлекательна и возможность получить в результате вместо списка тысяч предположительно релевантных ресурсов рейтинговый список ресурсов, рекомендованных другими пользователями.

Критичными для такого сервиса являются: дружелюбный пользовательский интерфейс, масштабируемость и надежность; в частности, защищённость от спама и других угроз. Чем шире круг участников, тем полнее, актуальнее и качественнее будет каталог при правильной организации системы.

2.2 Описание функциональности каталога

Сама возможность построения такой системы не кажется очевидной. Казалось бы, система должна анализировать и объединять структуры независимо созданных разными пользователями каталогов. Однако принцип действия сервиса не столь сложен. Есть множество U пользователей, пополняемое пользователями множество R ресурсов и также пополняемое пользователями множество T имён признаков (они же имена разделов и подкаталогов). Кликая по имени не выбранного признака $t \in T$, пользователь тем самым добавляет его в текущий контекст $C \subset T$, попадая на страничку нового контекста $C \cup \{t\}$, на которой между выбранными и доступными для дальнейшего выбора признаками расположены ссылки на ресурсы в рейтинговом порядке.

Ниже выбранных признаков следуют другие характерные для всех ресурсов этого контекста признаки.

Пользователь одним движением мышкой может

- изменить и зафиксировать рейтинг и контекст ресурса;
- изменить порядок предпочтения признаков;
- переместить ресурс в подкаталог либо вынести в родительский каталог;
- добавить или удалить ресурс, или каталог в данном контексте;

переименовать или скопировать подкаталог;

- вынести подкаталог наружу, изменив имя.

Все эти действия фиксируют индивидуальные рейтинги ресурсов и каталогов (для подкаталогов контекстно), доопределяют или изменяют пользовательскую функцию инцидентности

$$I_u : C \times R \to \{0,1\}$$

и функцию желательности тега в контексте

$$D : C \times R \to [0,1].$$

Авторитетность пользователя A_u в контексте рассчитывается по относительному количеству других пользователей, поддержавших его выбор в этом и ближайших контекстах.

Значения функции инцидентности усредняются по U с весом авторитетности, образуя идеальную функцию инцидентности I,. С такими же весами усредняются рейтинги. Полученные данные используются для формулировки задачи оптимизации, решением которой является структура каталога, максимально отвечающая пожеланиям пользователей.

Структура каталога формализуется как ориентированное дерево в ориентированном графе всевозможных контекстов, для которого сумма всех штрафов минимальна. Штрафы должны отражать среднюю длину нерасширяемого пути в дереве и подходящим образом определённые суммы значений функций D и I. Речь ни в коем случае не идёт о поиске точного решения (вряд ли оно будет доступно даже суперкомпьютерам будущего), а об итерационном улучшении имеющегося каталога перебором возможных простейших операций редактирования структуры каталога.

При такой оптимизации в корневом разделе со временем могут оказаться совсем иные признаки; структура каталогов сможет беспрепятственно перестраиваться, следуя изменяющимся потребностям пользователей.

Очевидно, что каждый контекст при оптимизации может улучшаться практически независимо от других контекстов. Контексты, к которым утрачен интерес, можно не оптимизировать, а популярные контексты необходимо оптимизировать чаще и, может быть, более тщательно.

Алгоритмы и настройки оптимизации каталога и расчёта авторитетности пользователя, дизайн контекстной страницы, должны совершенствоваться с ростом каталога.

Потребуется развивающаяся система подсказок, помогающих пользователям упрощать каталог, выявляя равнозначные признаки или наборы признаков.

Полезно будет увидеть произошедшие за указанный им период изменения в контексте и пересмотреть свои действия (или действия, сделанные от его имени).

Потребуется добавить платные сервисы для самоокупаемости. Если мнение пользователя отличается от других, ему должно быть это ненавязчиво показано с возможностью простого обсуждения наличия конкретного признака у конкретного ресурса или рейтинга конкретного ресурса или каталога.

Сложность сочетания многих перестраиваемых обработок с быстротой реакции на действия пользователя очевидно препятствует использованию стандартной основы в этой задаче.

2.3 Другие возможные применения

Приведённый пример не является исключением. В интернете наблюдается бурный рост

- ресурсов открытых технических стандартов, включая библиографические классификаторы;

ресурсов открытых образовательных стандартов и систем дистанционного образования;

- общедоступных ресурсов по биологии и другим наукам, склонным к пересмотру концепций и классификаций;

ресурсов открытого исходного кода больших программных систем;
государственных и корпоративных ресурсов законодательно-нормативной информации.

Потребность в частой коррекции структуры любого подобного ресурса обусловлена не столько неизбежной для изменчивых систем неполнотой предпроектного обследования, сколько

- наличием внутренних несогласованностей и требующих исправления противоречий,
- рассогласованностью политик и приоритетов хозяев ресурса,

непредсказуемостью изменений в предметной области.

Рост масштабов, сложности и глобальности информационных систем актуализирует проблему совместной доработки их наполнения.

Такие наполнения сложнее, чем коллаборативный каталог-классификатор. Для их качественного улучшения нужны разнообразные новые инструменты, часто узкоспециальной направленности. Такие инструменты зачастую могут быть созданы или адаптированы только узкими специалистами-пользователями и это процесс полезно организовать в самой системе.

Полезные применения подобная система может найти и в образовании [4, 5].

Непрерывная интеграция [6] обещает удешевление и ускорение совместной разработки программного обеспечения и ему подобного сложного контента при одновременном повышении качества за счёт сочетания известного списка шаблонов.

Эффект непрерывной интеграции определяется качеством организации тестирования и связи разработчиков с пользователями.

Обещая локализовать в пространстве данных и времени любую ошибку (будь то ошибка в коде или в действиях администратора), основанные на ретроспективной СУБД системы могли бы стать намного более эффективны и дружелюбны к разработчикам и пользователям, чем строящиеся на стандартной основе. Цикл «ошибка замечена пользователем - исправлена - проверена» мог бы быть сокращен до нескольких минут.

Не менее соблазнительна возможность реализации прозрачного автоматического учёта авторских вкладов участников в улучшение каждого контекста данных с последующей оценкой соответствия пожеланиям пользователей.

Подход может рассматриваться и в качестве основы для системы неограниченно возрастающей сложности, долговременно устойчиво развивающейся (“perpetual evolution”) на фоне технических, политических и юридических преображений. Новые исходные требования к такой системе лаконично сформулированы институтом SEI (Software Engineering Institute) в [7,8] в виде ключевых особенностей широкомасштабируемых социотехнических систем будущего (ULS):

- Децентрализация;
- Противоречивые по своей сути, непознаваемые, и разнообразные требования к системе;
- Непрерывное развитие и развёртывание;
- Разнородные, рассогласованные и изменяющиеся элементы;
- Стирание границы между людьми и системой;
- Устойчивость к ошибкам в исходных данных и при их обработке;
- Соревнование за системные ресурсы;
- Наличие организаций и участников, ответственных за установление политик;
- Наличие организаций и участников, ответственных за производство самой системы;
- Локальные и глобальные показатели здоровья, которые будут подключать необходимые изменения в политиках и в поведении элемента и системы;
- Дизайн и эволюция производственных отношений;
- Оркестровка и управление;

- Наблюдение и оценка.

Сформулировав перечисленные требования, рабочая группа SEI дала им недвусмысленную оценку:

“These characteristics undermine the assumptions we make in most current technical, management, and acquisition approaches.”

“Today’s approaches are based on perspectives that fundamentally do not cope with the new characteristics arising from ultra-large scale. The mentality of looking backward doesn’t scale.”

Такая оценка несомненно ориентирует на пересмотр стандартных подходов.

Напрашивается гипотеза о том, что создание и развитие систем [5], основанных на ретроспективном доступе к информации, указывает значительно более простой, просматриваемый и экономный путь к ULS, чем предложенное SEI развитие сервер-ориентированной архитектуры.

Подтвердить или опровергнуть эту гипотезу могут только будущие результаты прикладных исследований.

3. Общий план разработки

Логику сложной творческой деятельности практически невозможно чётко выяснить и формализовать в предпроектном обследовании. В идеале она вырастает в ходе совместной деятельности разработчиков и субъектов самого бизнес-процесса и гибко меняется при необходимости. Поэтому логика приложения не закладывается в основу архитектуры, а выращивается в процессе использования.

Система, удовлетворяющая требованиям децентрализации, непрерывного развития и развёртывания, содержащая разнородные, рассогласованные и изменяющиеся элементы и устойчивая к ошибкам в исходных данных и ошибках исполнения, по-видимому, должна, подобно персику, состоять из ядрышка, косточки и мякоти [9]. Косточка – это закрытая для изменений часть системы, обеспечивающая её целостность. Ядрышко — это защищённая история версий исходных кодов и иной информации косточки и мякоти. Мякоть — это изменяемая часть логики системы, обеспечивающая возможные изменения платформы системы и логики приложений.

Разработка очередной системы описываемого класса должна быстро и просто базироваться на подходящем прототипе, в котором постепенно будет корректироваться и наращиваться требуемая (не обязательно классическая, см. [10]) логика приложений.

Создание «косточки» для первого такого прототипа, способного эффективно координировать действия разработчиков и пользователей, является сложной научно-технической проблемой. Используемые для создания стандартных систем инструменты, заготовки и готовые архитектурные решения практически непригодны для достижения поставленной цели. Требуется в комплексе решить ряд непростых тесно взаимосвязанных специфических задач. Этим задачам не видно эффективных применений в рамках традиционных технологий. Вместе они образуют труднопреодолимый барьер: никакая самостоятельная часть работы не может быть выполнена в короткий срок.

3.1 Разработка многопоточной B+tree СУБД

Вставка в ключ каждой записи B+tree СУБД строки, идентифицирующей автора изменений, и в конец ключа строки, лексикографически упорядоченно идентифицирующей моменты актуальности и окончания обработки, обеспечивает целостность истории и быстрый доступ к актуальной записи.

Особенность предъявляемых при этом к СУБД требований в том, что от неё *не требуется модифицировать или удалять актуальные записи.*

Для рассматриваемого класса СУБД каждая запись состоит из ключа и значения. Ситуация, в которой требуется записать старый ключ с новым значением, может возникнуть только при редкой ошибке и тогда уже не важно, какое именно значение окажется у ключа. Поэтому порядок, в котором производятся записи, не существенен. Коммутативность и идемпотентность добавления записей резко упрощает [3] задачу объединения изменений при параллельной обработке. Основная проблема здесь состоит в необходимости предоставления беспрерывного доступа на чтение записанных данных во время пополнения базы свежими данными. Такая возможность была документирована в Berkeley DB 4.2, но в более поздних версиях разработчики отказались от поддержки этой возможности. Сейчас она документирована только в TokyoCabinet, но заслуживающих доверия тестов качества поддержки этой функциональности пока не известно. Возможно, что изменяемая база TokyoCabinet периодически нуждается в дефрагментации, во время которой она не должна быть доступна на чтение.

В работе [5] предложена идея каскадного использования B+tree СУБД, позволяющая получить требуемую функциональность за счёт одновременного использования и подготовки нескольких баз вместо одной. От СУБД требуется поддержка *поиска ближайшего ключа* к произвольной строке запроса (имеется в виду близость в лексикографически упорядоченном списке ключей, пополненном строкой запроса). Идея состоит в формировании маленьких баз, содержащих дополнения за долю секунды и слияния их в фоновом режиме в более крупные с дефрагментацией и оптимизацией. Приступая к обработке запроса, интерфейс СУБД проверяет, не устарел ли список баз в памяти системного процесса (либо в памяти нити исполнения) и при необходимости обновляет этот список. Устаревшие базы удаляются в фоновом режиме.

Ожидаемая производительность СУБД по

чтению в 4–10 раз ниже, чем у неизменной оптимизированной СУБД того же размера при задержке видимости записи, выполненной на том же сервере, порядка 0.1 секунды. Задержка видимости записей, произведённых на других серверах, определяется качеством соединения и в штатном режиме может составлять несколько секунд. По-видимому, таких показателей уже достаточно для рассматриваемых приложений.

Завершение отладки и тестирования позволит говорить о выполнении этой части плана.

3.2 Создание ретроспективного словаря

Ретроспективным словарём назовём подсистему ИС, данные которой являются ассоциативным массивом с ретроспективной индексацией. Название происходит от используемых в ИС системных словарей, представляющих собой списки названий сущностей (организаций, должностей, степеней и званий и т.п.).

Нужны эффективные алгоритмы поддержания ретроспективной согласованности: автоматически обновляемый словарь должен корректно учитывать указанное в запросе время. Кроме целых слов и фраз, полезно индексировать их общие начала для комфортного автозаполнения формы поиска, показывать пользователю точное количество найденных продолжений фраз. Эта часть работы также находится в состоянии отладки и экспериментальной проверки алгоритмов.

3.3 Обеспечение низкоуровневой навигации по ретроспективной СУБД

Низкоуровневая навигация по дереву данных означает возможность видеть, перемещаясь по дереву данных, в каком месте дерева мы находимся, и что в нём лежит. Для многоуровневых структур данных описание узла должно быть лаконичным и полным, а документирование (составление описаний) не должно требовать повторных действий.

Так, для ветки, содержащей несколько пользовательских словарей, полное имя узла может получаться из имени ветки, имени пользовательского словаря и слова, значение которого содержится в узле. При этом имя ветки и имя пользовательского словаря извлекаются из системного словаря по полному пути к корню ветки и по относительному пути к словарю в ветке, а искомое слово — это относительный путь узла в родительской ветке.

В общем случае полное имя каждого узла или ветки для низкоуровневой навигации должно быть получено из сохраненной в системном словаре информации и идентификаторов строк, записанных в ключах. Требуется проработать и чётко описать эффективные алгоритмы, обеспечивающие поддержание качественных полных и составных имен для неограниченно изменяющихся структур данных.

3.4 Хранение изменяющихся структур данных

Хранение изменяющихся структур данных в сериализованном виде требует дублирования всей структуры при любых изменениях или перестановках элементов, что на порядки увеличивает потребности в ресурсах. Ретроспективный доступ к информации и её эффективное хранение могут быть обеспечены только раздельным хранением элементов структур данных. В ретроспективном хранилище без особых усилий могут быть поэлементно поддержаны такие структуры как:

- строки произвольной длины;
- массивы однородных данных (строк, массивов или ассоциативных массивов);
- ассоциативные массивы строк или ссылок на данные базовых типов;

Особенность хешей в ретроспективной СУБД состоит в легкости реализации дополнительных операций: операции поиска лексикографически ближайшего снизу к заданной строке ключа и операций перехода к следующему и предыдущему ключу. Числовые данные удобно представлять строками в новой стандартной форме, обеспечивающей возможность лексикографического сравнения.

Для всех структур требуются операции вставки, удаления и перестановки элементов или подструктур, которые сами могут иметь сложную структуру. Для эффективной поддержки возможностей быстрой отмены перемещения подструктур и применения сложных перестроек структур данных полезны временные ссылки, работающие следующим образом. Появление в корне ветки *A* временной ссылки на ветку *B* логически эквивалентно полному удалению ветки *B* с немедленным копированием ветки *A* в корень ветки *B*. Интерпретация ссылок должна осуществляться с учётом очерёдности их появления. При одновременных ссылках больший приоритет имеет более близкая к узлу.

Для надёжности допускаются произвольные комбинации временных ссылок. Процесс перемещений узла при интерпретации ссылок останавливается, если очередная ссылка уже использована или ведёт к исходному узлу. Например, две одновременные противоположных ссылки между непересекающимися ветками просто меняют их местами, а перемещение ветки внутрь себя срабатывает один раз.

Появление временной ссылки приводит к резкому замедлению чтения информации, вызванному необходимостью нетривиальной корректной интерпретации временных ссылок в ретроспективных запросах. Требуется операция компиляции, приводящая к замене совокупности временных ссылок на ветки совокупностью статических ссылок на конкретные записи. Такая ссылка не требует сложной интерпретации при чтении.

Реализация находится в стадии предварительной проработки алгоритмов и вспомогательных структур данных.

3.5 Ретроспективная рекурсия шаблонов

Обработка запроса браузера должна с минимальными затратами возвращать страничку контекста по состоянию входящих данных на заданный момент прошедшего времени. Чтобы небольшие изменения в страничках не дублировали неизменную часть информации, независимые части страничек должны храниться отдельно и собираться при обработке запроса.

Общими параметрами запроса на чтение являются идентификатор автономного контекста данных и строка времён запроса. Строка времён может отсутствовать, и тогда – если в контексте указан момент начала незавершившихся изменений, – то используется предыдущий к нему момент, а, если не указан, – то предыдущий к моменту получения запроса.

Если авторизованный пользователь имеет административные полномочия в контексте, то в запросе может присутствовать категория доступа, идентифицирующая класс эквивалентных в данном контексте наборов ролей, которыми может обладать пользователь, либо идентификатор другого пользователя, интерфейс которого хочет увидеть администратор.

По категории, языку и иным предпочтениям пользователя формируется строка, идентифицирующая шаблон, получаемый из подходящей записи в хранилище в ответ на запрос пользователя. Если точно подходящего шаблона нет, то используется лексикографически предшествующий идентификатор. При необходимости используются статические либо динамические ссылки на строки. Последние отличаются тем, что указывают не на конкретную запись, а на текущее значение логического ключа.

В шаблон рекурсивно подставляются наиболее подходящие шаблоны и строки с учётом категории, языка и данных пользователя. Такая упрощенная рекурсивная подстановка должна быть реализована без сложных библиотек, обычно используемых развитыми средствами обработки шаблонов, чтобы эффективно и экономно обеспечить ретроспективный доступ к интенсивно дорабатываемым страничкам интерфейсов, минимизировать дублирование кода и облегчить его повторное использование.

Пробный интерфейс рекурсивной подстановки реализован на Perl и нуждается в испытаниях на реальных данных.

3.6 Поддержка динамических ссылок

Динамические ссылки создают эффект наложения слоёв информации. При неудачном чтении по заданному пути проверяется наличие ссылки в этом пути и производится попытка чтения из ветки, на корень которой указывает ссылка. Этот процесс может неограниченно повторяться без повторного использования ссылок.

Динамические ссылки могут быть двух типов: жёсткая (по умолчанию) адресует только данные, источника записанные до ссылки, и мягкая (для особых случаев), адресующая и более поздние данные. Хотя выявление и использование динамических ссылок в разы замедляет чтение информации, но разумные алгоритмы оптимизации могли бы значительно сократить накладные расходы при чтении структур данных.

Наличие жёстких ссылок особо усложняет удаление из базы давно неактуальных данных. Для этой цели придётся ввести и поддерживать системный индекс, позволяющий для любой записи быстро узнать, адресуется ли к ней существующая ссылка или нет.

Детальная проработка вопроса отсутствует.

3.7 Контекстно-автономные разграничение доступа и управление обновлениями

Идея контекстной автономности родственна идее процессного подхода по ISO 9001:2008. По принципу единоначалия, один человек отвечает за систему в целом, но поскольку большая система состоит из подсистем, ответственных за отдельные процессы, то руководитель делегирует управление и ответственность за некоторые или все подсистемы своим подчинённым, которые могут продолжить делегирование для составляющих своих процессов. Таким образом, всё информационное пространство системы предстаёт иерархией автономно управляемых подпространств — контекстов, обслуживающих процессы в понимании ISO 9001.

Контексты данных отличаются от компонент или объектов тем, что их данные доступны без задержек в последней согласованной (утверждённой) редакции (версии). Вводимые пользователем данные либо пишутся в индивидуальные контексты, либо помечаются идентификатором пользователя.

Данные в контексте могут иметь разные версии для разных групп пользователей. Группа, к которой относится пользователь, устанавливается контекстно или наследуется из более широкого контекста.

Автономные контексты независимо обновляют свою информацию и содержат динамические ссылки на ветки других контекстов

Мониторинг ресурсов настраивается в рамках более широкого автономного контекста с использованием явных и неявных оценок качества кластерами потребителей.

Момент актуальности контекста – последний из моментов времени, на который информация актуальна – определяется записями с особым логическим ключом. При обновлении контекста определяется последний момент, на который актуальны входные данные, запускается их обработка, в контекст записываются изменившиеся значения с новыми датами, а значение ключа момента актуальности устанавливается на этот момент в пустую строку. При подтвержденном пользователем вводе данных в контекст значением ключа становится список ключей изменённых

данных.

Каждый запрос к системе адресуется к конкретному контексту. Во фрагмент странички ответа на запрос могут включаться фрагменты страничек других контекстов на указанный момент актуальности.

Ответ на запрос сопровождается указанием момента актуальности. Более поздняя информация временно недоступна. Если включаемые фрагменты неустойчивы на указанный момент, то выбирается самый ранний из моментов актуальности и запрос повторно обрабатывается на более ранний момент, до тех пор, пока вся информация странички не окажется актуальной на указанный момент.

При чтении информации в отдельном для каждого контекста ключе (непосредственном или через динамическую ссылку) делается (при отсутствии действующей) пометка об использовании контекста конкретным пользователем или контекстным обновлением. Эти пометки вносятся так, чтобы облегчить автоматическое поддержание приблизительного рейтинга контекстов по популярности.

Этот рейтинг, априорная оценка контекстов и пометки об изменениях используются для выделения контекстов, срочно нуждающихся в обработке. Обработка производится параллельно и независимо. При высоком рейтинге и отсутствии изменений в данных обработка сводится к пометкам об использовании контекстов исходных данных.

Детальная проработка подсистемы отсутствует.

3.8 Ресурсосбережение

Хранение истории исходных данных, промежуточных и окончательных результатов обработки требует затрат, существенно зависящих от способа хранения и поэтому трудно поддающихся оценке.

Чтобы удаление данных не нарушило требуемую целостность системы, предполагалось выделять в далёком прошлом короткие промежутки времени, так называемые «белые пятна истории», связанные с интенсивным изменением данных, и удалять версии данных, не актуальные вне этих промежутков. Поскольку выделение таких пятен при больших объёмах данных является непростой задачей, то в [11] предлагалось решать её в рамках отдельных автономных контекстов, что могло нарушить согласование данных между разными контекстами.

Найдено более простое решение, позволяющее чистить ненужную часть истории без перерасхода ресурсов и угрозы рассогласования. Оно состоит в увеличении шага дискретизации для очень старых данных. Например, если шаг дискретизации для данных более чем десятилетней давности переустанавливается в одни сутки, то из всех изменений любого данного в течение таких давно прошедших суток оставляется последнее, актуальное на момент полуночи. Все изменения, актуальные на границе каких-либо суток, остаются в доступе, а все очень старые изменения, не дожившие до ближайшей полуночи, теряются безвозвратно.

Требование устойчивости к ошибкам будет сохранять актуальность и по чисто технической причине [12,13]: уменьшение размеров транзисторов и их энергопотребления при повышении быстродействия приводит к усилению эффектов квантовой механики, неотвратимо повышающих вероятность ошибки в конкретной ячейке памяти. Эти ошибки в серверном процессоре компенсируются использованием дублирования и особого кода, исправляющего ошибки, но требующего дополнительных ресурсов. Предлагаемая архитектура предъявляет высокие требования только к хранению информации и исполнению регламентных процедур косточки. Обработки данных приложений в ряде случаев могут проводиться дешевле и с меньшей надёжностью, если выходные данные в контексте подвергать независимому контролю и при необходимости перевычислению.

3.9 Разработка пользовательских интерфейсов косточки

Пользовательские интерфейсы косточки должны предоставлять полноценный доступ к низкоуровневым администрированию и разработке системы, и истории работ и одновременно обеспечивать асинхронное информирование о системных сбоях и проблемах пользователей в поддерживаемых контекстах данных. В этом направлении отлажено несколько частичных прототипов, но окончательных результатов пока нет.

4. Выводы

Предложено описание ретроспективного подхода к реализации прикладных информационных систем, качественно превосходящих те, которые могли бы быть созданы на стандартной основе. Описаны область применимости подхода, общий план и текущее состояние реализации. Сформулирован ряд задач и направлений дальнейшей разработки.

Литература

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

[2] Mart Roost, Karin Rava, and Tarmo Veskioja. 2007. Supporting self-development in service oriented information systems. In Proceedings of the 7th Conference on 7th WSEAS International Conference on Applied Informatics and Communications - Volume 7 (AIC'07), Minh Hung Le, Metin Demiralp, Valeri Mladenov, and Zoran Bojkovic (Eds.), Vol. 7. World Scientific and Engineering Academy and Society (WSEAS), Stevens Point, Wisconsin, USA, 52-57.

[3] Helland, Dave Campbell. Building on Quicksand. Proceedings of the Fourth Biennial Conference on Innovative Data Systems Research (CIDR 2009),

January 4-7, 2009, Asilomar, Pacific Grove, CA USA. Перевод на русский язык: Пэт Хелланд, Дейв Кэмпбел. Дом на песке, 2010. - http://citforum.ru/database/articles/quicksand/.

[4] В. А. Болотов, С. В. Знаменский. Требования к информационной системе управления качеством образования. "Программные системы: Теория и приложения", №2(2), 2010, с. 3-13.

[5] С.В. Знаменский. Гибкая основа информационной системы для обучения. //Труды XII-й Всероссийской научной конференции «Электронные библиотеки: перспективные методы и технологии, электронные коллекции» RCDL-2010. Казань: Казанский университет. 2010, с. 451-460.

[6] P. M. Duvall. Continuous Integration: Improving Software Quality and Reducing Risk, Addison-Wesley, 2007. - http://www.amazon.com/gp/product0321336380/?tag=integratecom-20.

[7] L. Northrop.The Impact of Scale: Carnegie Mellon University. December 17, 2009. - http://www.sei.cmu.edu/library/assets/20091217webinar.pdf.

[8] L. Northrop. Ultra-Large-Scale Systems. The Software Challenge of the Future. June 2006. Pittsburgh, PA 15213-3890. — http://www.sei.cmu.edu/library/assets/ULS_Book20062.pdf.

[9] С.В. Знаменский. Хорошо масштабируемое автономное администрирование доступа. Труды Международной конференции "Программные системы: теория и приложения", Переславль-Залесский, октябрь 2006, Наука-Физматлит, М., Т.1, с. 155-169. - http://skif.pereslavl.ru/psi-info/psi/psi-publications/e-book-2006/index.html.

[10] Н.Н. Непейвода. Прикладная логика. Новосибирск: НГУ, 2000.

[11] С.М. Абрамов, Н.С. Живчикова, С.В. Знаменский, Е.С. Иванов, А.В. Котомин, Д.Н. Степанов, Е.В.Титова, В.Н. Юмагужина. "Архитектура системы для разработки технологий организации сложной совместной деятельности". Прикладная информатика, №2(26), 2010, с. 31-41

[12] J.C. Laprie, Dependability: Its Attributes, Impairments and Means. In Predictably Dependable Computing Systems, B. Randell, J.C. Laprie, H. Kopetz, and B. Littlewood (eds.), pp. 1-28, Springer-Verlag, 1995.

[13] B. Bloom. Space/time Trade-Offs in Hash Coding with Allowable Errors. In Communications of ACM, volume 13(7), 1970, p. 422-426.

Factographic Base for the Complex Collaborative Information Resources Reorganization

© S.V. Znamenskij

Some Information Resources, especially those addressed the complex issues are known to be continuously improved with a broad range of individuals and organizations.

Such refinement is often needed in unexpected corrections of the structure of the resource, parallel development and comparative testing of users of multiple versions of its components, removal of excess and restore previously available data.

We discuss the idea of building a representation of data in such a system on immutable records addressable given date, time and authorship of their creation and providing accelerated multiplayer improvements due to

(1) immediate access to the same state information and the essential characteristics of change;

(2) localization of execution errors in time and information space;

(3) the reorganization of a pluralistic view and update data and multi-level testing.

♣ Работа поддержана грантом РФФИ № 09-07-00407.