С. В. Знаменский, Процессный подход к эволюционированию информационных систем. Ретроспективное индексирование, *Программные системы: теория и приложения*, 2011, том 2, выпуск 4, 127–137

ISSN 2079-3316 ПРОГРАММНЫЕ СИСТЕМЫ: ТЕОРИЯ И ПРИЛОЖЕНИЯ № 4(8), 2011, с. 127–137

УДК 004.651.53:004.827

С. В. Знаменский

Процессный подход к эволюционированию информационных систем. Ретроспективное индексирование

Аннотация. Прорабатывается навеянный идеями ISO9000 процессный подход к архитектуре масштабируемой информационной поддержки коллаборативной творческой активности. Подход базируется на непосредственном ретроспективном доступе к информации, раздельной записи данных и функционально-реактивном исполнении. Предоставляя пользователям возможности перемещений во времени и организации параллельных миров, он обещает высокую реактивность, постоянную доступность, повышенную устойчивость к ошибкам исполнения и администрирования, неожиданным перегрузкам, поломкам и внезапным перестройкам системы.

Кратко рассматривается архитектура будущей системы и более подробно ретроспективное индексирование информации.

Ключевые слова и фразы: коллаборативное ПО, архитектура ИС, процессный подход, ретроспективная СУБД, устойчивость к перегрузкам, человеческий фактор.

Введение

Новые задачи коллаборативного творческого сотрудничества выдвигают [1] трудно совместимые требования к обработке данных:

- интерфейс должен быть живым и мгновенно реагирующим на действия пользователя;
- он должен быть логически целостным, свободным от алогизмов;
- он должен содержать дружественные подсказки, формируемые на основе деятельности в близких контекстах похоже мыслящих в близких контекстах пользователей.

Работа проводилась при поддержке Министерства образования и науки Российской Федерации и при поддержке РФФИ, грант № 09-07-00407.

Большие объёмы данных делают задачу формирования интеллектуально обновляемых подсказок крайне ресурсозатратной.Чтобы решать её без задержек, приходится отказаться не только от стандартной методологии, основанной на реляционной СУБД, но и от более изощрённых попыток сохранения свойств ACID, таких как доски объявлений [2].

Отказ от тотальной ежесекундной согласованности данных даёт альтернативный подход к обеспечению быстрого безупречно адекватного реагирования на действия пользователя. Фокус в том, что каждому пользователю важно, чтобы система быстро реагировала на его действия, но доставляет сомнительное удовольствие когда рабочий стол живёт своей жизнью. Поэтому пользователю удобно работать с застывшим состоянием системы, которое меняет он один, а обновления исследовать при новом входе в систему.

Это означает, что многие ресурсоёмкие задачи интеллектуального соединения информации от остальных пользователей, персонализации пользовательского интерфейса и фильтрации могут обрабатываться в фоновом режиме, параллельно и распределённо и в промежутках уточняться простыми алгоритмами, как это делается в рекомендательных системах [3]. Это означает, что в системе одновременно должны работать несколько независимых процессов, имеющих доступ на чтение к данным друг друга и постоянно улучшаемых без необходимости приостановки сервиса.

Поскольку идея достаточно непривычна, а план и составляющие проекта рассмотрены в [4], то лишь ради замкнутости изложения приведём краткое описания основы.

1. Архитектура для процессного подхода

Процесс поддержки текущей работы пользователя осуществляется клиентским приложением и отделяется от серверных процессов. Он накладывает персональные изменения на запрашиваемые и получаемые с сервера данные. Серверные процессы в свою очередь также разделяются на ряд групп:

(1) процессы вычисления разностей контекстов и по ним окрестностей каждого контекста;
(2) процессы кластеризации множества пользователей в окрестностях контекстов, выделения базовой и альтернативных версий

интерфейса в контекстах, и построение стратификации активных в окрестности пользователей;
(3) процессы формирования основы для подсказок;
(4) процессы анализа истории работы пользователей и уточнения их авторитетности;
(5) процесс распределения системных ресурсов между процессами.

Последний процесс регулирует распределение ресурсов между остальными группами процессов, работающими асинхронно и автономно.

1.1. Контекстная автономность

Всё адресуемое пространство системы разделяется на автономно обновляемые части — контексты данных. Каждая часть обновляется своим процессом понимаемым не в смысле процесса операционной системы, а в смысле процессного подхода ISO9000, имеющего человека-хозяина, цель, описание входных и выходных данных и периодически улучшаемого на основе контроля качества.

Смысл существования каждого такого контекста в специфическом информационном обслуживании, то есть в обеспечении клиентов качественно обработанной информацией. Например, работа отдельного пользователя в отдельной подсистеме - это отдельный процесс, пишущий свои данные в отдельную область памяти.

Любая часть выходной информация контекста может быть объявлена доступной для любого множества контекстов. Другие процессы могут читать эти данные, если им сообщить, где они расположены.

Хозяину всегда доступна информация, сколько пользователей и насколько часто используют те или иные выходные интерфейсы напрямую или через какие контексты. Для этого факты обращений к интерфейсам контекста фиксируются системой и используются для балансировки загрузки и распределения ресурсов между процессами на основе устанавливаемых хозяевами приоритетов.

В системе известно, где находится информация о контекстах, делегировании, ролях, приоритетах и настройках ресурсов, как собирается статистика использования и организуется выполнение обработки информации и обеспечивает эти работы в точном соответствии с регламентом. Мониторинг ресурсов настраивается в рамках каждого автономного контекста с использованием явных и неявных оценок качества кластерами потребителей.

1.2. Ретроспективная основа

Главным препятствием к обеспечению требуемой производительности в больших системах с тесно взаимосвязанной внутренней структурой проблема состоит необходимость поддержания данных в согласованном состоянии. Версионирование на основе временных штампов становится общепризнанным подходом к отделению актуальной информации от устаревшей. Опора на эффективный доступ к прошлой информации системы становится и основным средством защиты системы от ошибок в прикладных обработках.

Отсюда идея сохранять любую информацию с указанием времени актуальности исходных данных и иногда с дополнительными штампами времени достаточно долго, чтобы бесконфликтно выделить актуальную согласованную информацию на почти любой момент прошлого. Почти любой в том смысле, что, например из очень старой информации останется только та, которая была актуальна в полночь, а изменения, не дожившие до полуночи, не столь значимы и пропадут (если специально не продублированы).

Иначе говоря, шаг дискретизации времени для путешествий в прошлое должен расти по мере удаления от настоящего. Вкупе с географически распределённым дублированием информации это способно полностью решить проблему администрирования резервного копирования, снимая с администратора головную боль хранения резервных копий.

Любой запрос на чтение новой информации мгновенно возвращает установившееся в системе прошлое состояние и остается гарантированно воспроизводимым с тем же результатом. Если запрос к системе адресован к медленно обрабатываемым изменчивым данным, то система не заставит ждать окончания обработки, а немедленно выдаст последнюю готовую версию с индикацией момента времени актуальности и ставит запрос в очередь обработки.

Такой пока непривычный интерфейс позволит значимо ускорить работу с системой, задержки в работе которой неустранимы по причине больших расстояний или сложных алгоритмов. Ремонт бизнес-логики будет воспринято пользователем как замедление обработки новых данных. Если сопровождающий код программист автоматически будет извещаться о поломке, то будет шанс стол быстрого устранения, что пользователи не заметят дефекта.

1.3. Организация данных

Особенность предъявляемых требований к B+tree СУБД в том, что от неё не требуется модифицировать записи или удалять недавно созданные. Для рассматриваемого класса СУБД каждая запись состоит из ключа и значения. Добавление к ключу каждого записываемого данного времени нулевого байта, строки, лексикографически упорядоченно идентифицирующей момент актуальности, возможно, ещё одного нулевого байта и строки, идентифицирующей момент окончания обработки, превращает такую B+tree СУБД в ретроспективную, предоставляющую за малое логарифмически растущее время доступ к актуальной версии любого данного на любой запрашиваемый момент.

Синхронизация достаточно просто сводится к защищённому от потерь обмену списками новых записей. Ситуация, в которой требуется записать старый ключ с новым значением может возникнуть только при редкой ошибке и какое именно значение окажется у ключа в этом случае не важно. Поэтому порядок, в котором производятся записи, не существенен и это снижает конкурентность записи, существенно облегчая задачу синхронизации данных.

Логическая согласованность обеспечивается

- пометкой версии пользовательских изменений единым временем,
- пометками результатов обработки единым для группы процессов временем пользовательских изменений,
- непосредственным доступом к нужным версиям.

2. Ретроспективный индекс

Никакая "Машина времени переносящая нас в прошлое состояние информационной системы, не сможет полностью воспроизвести функциональность старых интерфейсов, если доступ к информации будет затруднён отсутствием актуальных индексов. Ниже описывается ретроспективный индекс — механизм, предоставляющий производить поиск информации по состоянию на заданный момент времени. Он описывается на примере поиска персон.

2.1. Пример индекса

Пусть например, индекс с ключом `fio` используется для поиска в следующем массиве

`1z` => Абанин Александр Владимирович
`1adc` => Абрамов Сергей Михайлович
`1dz` => Абрамов Николай Сергеевич
`1sk` => Амелькин Сергей Анатольевич
`1uk` => Амосов Александр Анатольевич
`1ty` => Аникина Анна Михайловна

Это означает наличие в базе следующих записей в указанном порядке (время сохранения и прочие служебные атрибуты для простоты не показаны):

`ifio А` => [[’Аб’, 3], [’Ам’, 2], ’1ty’],
`ifio Аб` => [’1z’, [’Абрамов ’, 2]],
`ifio Абанин Александр Владимирович` -> [’1z’],
`ifio Абрамов` => [’1dz’, ’1adc’],
`ifio Абрамов Николай Сергеевич` => [’1dz’],
`ifio Абрамов Сергей Михайлович` => [’1adc’],
`ifio Ам` => [’1sk’, ’1uk’],
`ifio Амелькин Сергей Анатольевич` => [’1sk’],
`ifio Амосов Александр Анатольевич` => [’1uk’],
`ifio Аникина Анна Михайловна` => [’1ty’],

При наборе ’А’ в окошке поиска браузера

- сервер заменит ’1ty’ на [’Аникина Анна Михайловна’, 1, ’1ty’],
- высветятся подсказки

`Аб... (3)`
`Ам... (2)`
`Аникина Анна Михайловна`,

позволяющие продолжить выбор мышью и показывающие, сколько записей начинается на выбираемую строку.

Интерфейс браузера может быть и более интеллектуальным, сейчас обсуждается чтение информации.

Будут видны только актуальные на указанный момент записи. Более новые, чем требуется, записи из базы не считаются.

2.2. Модификация индекса

Допустим, добавилась персона

`1ta` => Абрамовский Игорь Николаевич

Это означает, что нужно как при вводе мышью

- сделать запрос 'ifio\tA ~ '
- выбрать содержащееся в добавляемой 'Аб', заменить 3 на 4
- сделать запрос 'ifio\tАб ~ ';

Так продолжается, пока не выявится уникальность продолжения 'овский Игорь Николаевич'; после этого:

- выбрать имющее более длинную общую часть из 'Абрамов ' и 'Абрамовский Игорь Николаевич',
- заменить в 'ifio\tАб\t' ключ 'Абрамов ' на новую общую часть 'Абрамов' с увеличенным на 1 количеством,
- создать для новой общей части 'Абрамов' новую запись,
- создать запись "ifio\tАбрамовский Игорь Николаевич\t',

Здесь используется специальная кодировка символов, в которой символ ~ располагается после всех букв, а \t - перед всеми. В итоге должно получиться

`ifio А` -> [['Аб', 4], ['Ам', 2], '1ty'],

`ifio Аб` -> ['1z', ['Абрамов', 3]],

`ifio Абанин Александр Владимирович` -> ['1z'],

`ifio Абрамов` -> [['Абрамов ', 2], '1ta'],

`ifio Абрамов` -> ['1dz', '1adc'],

`ifio Абрамов Николай Сергеевич` -> ['1dz'],

`ifio Абрамов Сергей Михайлович` -> ['1adc'],

`ifio Абрамовский Игорь Николаевич` -> ['1ta'],

`ifio Ам` -> ['1sk', '1uk'],

`ifio Амелькин Сергей Анатольевич` -> ['1sk'],

`ifio Амосов Александр Анатольевич` -> ['1uk'],

`ifio Аникина Анна Михайловна` -> ['1ty'],

2.3. Реальная структура индекса

В предыдущем пункте для упрощения понимания дерево индекса простроено до последних листиков. На самом деле это никому не нужно. Браузер в состоянии сам осуществить окончательную фильтрацию списка.

`ifio А` -> [[’Аб’, 4], [’Ам’, 2], ’1ty’],
`ifio Аб` -> [’1z’, [’Абрамов’, 3]],
`ifio Абрамов` -> [[’Абрамов ’, 2], ’1ta’],
`ifio Абрамов` -> [’1dz’, ’1adc’],
`ifio Ам` -> [’1sk’, ’1uk’],

Например, при наличии в списке 6 персон разумно сразу показать весь список, позволив пользователю выбрать нужную персону одним кликом мыши. Это означает, что реальный индексы должны быть намного короче: до добавления Абрамовского

`ifio` -> [’1dz’, ’1adc’,’1sk’, ’1uk’,’1ty’],

а после добавления

`ifio` -> [’1z’, ’1dz’, ’1adc’,’1sk’, ’1uk’, ’1ty’],

так что весь индекс заменен одним плоским списком. Обозначим P максимальный размер ветки, которая отображается плоским списком. Например, при P=2 последний индекс приведён выше. Такой приём позволяет уменьшить индексы примерно в P*(ln P) раз, но при P>2 алгоритм обновления усложняется.

При P=3 добавление к списку

1ac => Абрамович Степан Викторович

должно перевести

`ifio А` -> [[’Аб’, 4], [’Ам’, 2], ’1ty’],
`ifio Аб` -> [’1z’, [’Абрамов’, 3]],
`ifio Абрамов` -> [’1dz’, ’1adc’, ’1ta’],
`ifio Ам` -> [’1sk’, ’1uk’],

в

`ifio А` -> [[’Аб’, 5], [’Ам’, 2], ’1ty’],
`ifio Аб` -> [’1z’, [’Абрамов’, 4]],
`ifio Абрамов` -> [[’Абрамов ’, 2], ’1ac’, ’1ta’],

`ifio Абрамов` -> [’1dz’, ’1adc’],
`ifio Ам` -> [’1sk’, ’1uk’],
то есть переполненный список из p+1 элементов должен быть проиндексирован подробнее. В случае P=4 список
`ifio A` -> [[’Аб’, 5], [’Ам’, 2], ’1ty’],
`ifio Аб` -> [’1z’, [’Абрамов’, 4]],
`ifio Абрамов` -> [’1dz’, ’1adc’, ’1ac’, ’1ta’],
`ifio Ам` -> [’1sk’, ’1uk’],

отличается от P=2 (см. ниже) одной строкой и не вполне оптимален, более оптимален для пользователя был бы
`ifio A` -> [’1z’, [’Абрамов’, 4], [’Ам’, 2], ’1ty’],
`ifio Абрамов` -> [’1dz’, ’1adc’, ’1ac’, ’1ta’],
`ifio Ам` -> [’1sk’, ’1uk’],

позволяющий добраться до любой фамилии двумя кликами. Оптимизация произведена раскрытием сильно переполненного списка внутри недозаполненного. Алгоритм поддержания оптимизированных индексов существенно сложнее.

Поскольку после буквы ’A’ могут следовать почти 30 различных букв, то имеет смысл поддерживать P в диапазоне 5-20.

2.4. Структура оптимального индекса

Для построения и поддержания быстрого индекса разумно использовать P=2,
`ifio A` -> [[’Аб’, 5], [’Ам’, 2], ’1ty’],
`ifio Аб` -> [’1z’, [’Абрамов’, 4]],
`ifio Абрамов` -> [[’Абрамов ’, 2], ’1ac’, ’1ta’],
`ifio Абрамов` -> [’1dz’, ’1adc’],
`ifio Ам` -> [’1sk’, ’1uk’],

а оптимизацию производить при выдаче, пытаясь раскрывать в недозаполненном списке наиболее массивный элемент.

При P=4 в список "А"хорошо получается подставить список "Аб а в список "Абрамов"=> список "Абрамов ".

В список "Аб"не удаётся поставить 'Абрамов', поскольку P=4. Но в данном случае этих подстановок уже достаточно для полной оптимизации. Такая оптимизация может выполняться браузером и может учитывать предпочтения пользователя.

2.5. Индекс для поиска слов и словосочетаний

Для поиска слов и словосочетаний не в начале можно использовать '+' в начале строки поиска. Индекс при этом расширится и для нашего списка

`1z` => Абанин Александр Владимирович
`1adc` => Абрамов Сергей Михайлович
`1dz` => Абрамов Николай Сергеевич
`1ac` => Абрамович Степан Викторович
`1ta` => Абрамовский Игорь Николаевич
`1sk` => Амелькин Сергей Анатольевич
`1uk` => Амосов Александр Анатольевич
`1ty` => Аникина Анна Михайловна

дополнение примет примерно такой вид:

`ifio +А` -> [['+Александр', 2], '+Ан', 3]],
`ifio +Александр` -> ['1z','1uk'],
`ifio +Ан` -> ['1ty', '+Анатольевич', 2]],
`ifio +Анатольевич` -> ['1sk', '1uk'],
`ifio +В` -> ['1z', '1ac'],

2.6. Заключение

Описанный алгоритм реализован и ведётся его отладка в информационной системе УГП имени А. К. Айламазяна.

Список литературы

[1] Знаменский С. В. *Архитектура коллаборативно-изменяемой иерархической структуры* // Программные системы: теория и приложения, 2011. **2**, № 4(8), с. 127–138 ↑[]

[2] Демидов А. А. *Проектирование распределённых систем обработки объектных структур данных* // Электронные библиотеки: перспективные методы и технологии, электронные коллекции // Труды XII Всероссийской научной конференции RCDL'2010. — Казань : Казанский университет, 2010, с. 441–447 ↑[]

[3] Федоровский А. Н., Логачева В. К. *Архитектура рекомендательной системы, работающей на основе неявных пользовательских оценок* // Электронные библиотеки: перспективные методы и технологии, электронные коллекции // Труды XIII Всероссийской научной конференции RCDL'2011. — Воронеж : Воронежский госуниверситет, 2011, с. 76–82 ↑[]

[4] Знаменский С. В. *Ретроспективная основа совместной реорганизации сложных информационных ресурсов* // Электронные библиотеки: перспективные методы и технологии, электронные коллекции // Труды XIII Всероссийской научной конференции RCDL-2011. — Воронеж : Воронежский госуниверситет, 2011, с. 93–101 ↑[]

S. V. Znamenskij. *The process approach to information systems evolution. Retrospective indexing.*

Abstract. Inspired by the ideas worked out ISO9000 process approach to IS architecture for scaled collaborative creative activity support. The approach is based on direct access to retrospective information, separated data writing and functional-reactive performance. Giving users the ability to time travel and the organization of parallel worlds, it promises high reactivity, the constant availability, increased fault tolerance and tolerance to unexpected overloads, breakdowns and sudden rearrangements of the system.

The future system architechture and detailed retrospective indexing algorithm are under consideration.

Key Words and Phrases: collaborative software, IS architecture, process aproach, retrospective DBMS, fault tolerance, time travel, human factor.

Образец ссылки на статью:

С. В. Знаменский. *Процессный подход к эволюционированию информационных систем. Ретроспективное индексирование* // Программные системы: теория и приложения : электрон. научн. журн. 2011. № 4(8), с. 127–137. URL: http://psta.psiras.ru/read/psta2011_4_127-137.pdf