ISSN 2079-3316 ПРОГРАММНЫЕ СИСТЕМЫ: ТЕОРИЯ И ПРИЛОЖЕНИЯ № 2(11), 2012, с. 77–86

УДК 004.051+004.623

С. В. Знаменский

Глобальная идентификация данных в долговременной перспективе

Аннотация. Ретроспективный подход к хранению данных характеризуется высокими неподдельностью данных, доступностью истории и живучестью. Эти качества особенно привлекательны для будущих масштабируемых систем, долгоживущих в сложно изменяющихся условиях.

Описываются система элементарных запросов к ретроспективному хранилищу, источники ветвления информационных структур. Приводятся рекомендации по выбору системы глобальной идентификации объектов ретроспективного хранилища.

Ключевые слова и фразы: система версионирования, хранилище данных, структура B+tree, метка времени, история изменений, долгоживущие информационные системы.

Введение

Целостная неподдельная история изменений незаменима и широко используется

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

в организации более эффективного доступа к истории изменений перестраивающихся структур данных [1, 2]. Рост сложности и объёмов накапливаемых данных затруднят замену используемой информационной систем. Всё чаще исследователи обращаются к идее информационной системы, способной к гибкой эволюции [3].

Работа проводилась при финансовой поддержке Министерства образования и науки Российской Федерации.

При этом накопление структурных изменений в истории данных способно экспоненциально увеличивать объём хранимой информации, существенно нелинейно увеличивая затраты на обработку запросов.

Мы сосредоточим внимание на истории изменений иерархических структур данных, обеспечивающих универсальность в известном смысле: Любые существующие структуры данных общепринято описывать в формате XML. Такое описание имеет иерархическую структуру, однозначно и полностью фиксирующую исходную структуру данных. Если исходная структура перестраивается, то изменяется и иерархия XML-описания. Таким образом, любые перестройки произвольных структур данных могут качественно сохраняться в виде перестроек иерархической структуры. Иерархическая структура данных здесь понимается как дерево,с узлами которого ассоциируются данные, сериализованные строками произвольной длины, а порядок следования детских узлов любого родителя фиксирован.

Следуя [4], рассмотрим задачу эффективного построения ретроспективной СУБД для хранения изменений иерархической структуры данных. Объектом будем называть ветку иерархии. Система в целом рассматривается как объект, содержащий все остальные объекты. Перечислим особенности ретроспективного доступа к данным, которые позволяют рассматривать его как перспективную основу для долгоживущей и масштабируемой в сложных условиях системы:

Неподдельность: Любые данные сохраняются с фиксацией логического времени [6], настоящего или будущего и неизменны до их окончательного удаления из системы;

Живучесть: Окончательное удаление излишне детальной информации и задержки исполнения порождают «белые пятна истории объекта» — относительно малые легко обходимые промежутки времени, верное состояние информации внутри которых не обеспечено; острый недостаток ресурсов для хранения или обработки информации приведёт к многократному снижению качества, выраженном в увеличении белых пятен, но не к потерям функциональности или доступности сервиса;

Доступность истории: Для каждого объекта доступен список всех моментов его изменения, список белых пятен и его состояние на любой доступный момент времени;

На основе качественно реализованного ретроспективного доступа могут реализовываться системы, эмулирующие изменение данных и эксперименты с прошлым и будущим, и другие будущие механизмы доступа к темпоральной информации.

Подход разрабатывается для приложений, приоритетно нуждающихся в сочетании повышенных доступности, жизнеспособности и гибкой модифицируемости информационного обеспечения.

Если например он будет положен в основу операционной системы, то любые системные обновления будут включаться или отменяться мгновенно, не отвлекая без необходимости пользователя и не создавая угроз необратимой потери доступа к данным. При этом система на компьютере будет частью распределённой системы разработки и поддержки этой ОС.

Для реализации такой системы ключевой является проработанность основ организации, обеспечивающих разумную экономию ресурсов. Предлагаемая статья описывает возможный подход к глобальной идентификации данных, независимой от их размещения в распределённой системе.

1. Основа ретроспективного доступа

Структура идентификаторов данных обязана обеспечить эффективность обработки данных в долговременной перспективе. Рассмотрим структуру запросов на ввод и вывод этих данных.

1.1. Элементарные ретроспективные запросы и архитектура локальной части хранилища

Каждый элементарный запрос может сопровождаться пометкой времени, относительно которого определяется актуальность данных. При обращении время может уточняться, об этом ниже.

(1) чтение актуальных данных объекта, не входящих в подобъекты, и времени их последнего изменения;
(2) чтение актуальных данных последнего подобъекта (включая ранее удалённые);
(3) чтение актуальных данных предыдущего объекта (возможно удалённый ранее);
(4) чтение всех согласованных данных объекта в целом, вместе с подобъектами;
(5) чтение нуждающихся в согласовании данных объекта в целом, вместе с подобъектами;
(6) запись объекта с пометкой времени;

Трёх первых видов запросов достаточно для получения полной информации об истории изменений данных например, на языке SPARQL [7][1]. Четвёртый вид предназначен для оптимизации удалённого дублирования данных, а пятый- для их согласования.

[1]Для ускорения обработки полезно ввести служебные атрибуты, индексирующие информацию; например, время последнего изменения ветки, растущей из родительского узла.

Это означает, что перемещение объектов осуществляется копированием и удалением[2].

Поскольку согласованные данные в согласовании уже не нуждаются, то представляется разумным разделить локальное хранилище на две части: оперативно обновляемое хранилище текущей информации, участвующей в согласовании, и периодически заменяемое основное хранилище, оптимизированное на обработку запросов (1)–(3). При этом оперативно обновляемое хранилище должно поддерживать и операцию удаления данных, перенесённых в основное хранилище. Из-за более развитой функциональности продуктивность оперативно обновляемое хранилище сложнее. Но поскольку более 99% информации находится в основной неизменной структуре, доступ к которому легко сделать очень быстрым, то выигрыш от такого разделения должен с избытком компенсировать затраты на его поддержку.

1.2. Автономные объекты и белые пятна

Некоторые объекты объявляются автономными. Именно с ними связываются белые пятна. Согласованность записанных данных гарантируется только для автономных объектов вне белых пятен. Для системы в целом устанавливается шаг дискретизации времени, который может корректироваться для отдельных объектов. При обработке индексов или синхронизации данных, расположенных в разных частях распределённой системы, каждый раз образуется белое пятно.

Белые пятна, укладывающиеся в шаг дискретизации, не нуждаются в фиксации. При отсутствии перегрузок и непредусмотренных задержек синхронизации белых пятен не фиксируется

Мы предполагаем, что родительский объект определяется по идентификатору объекта. Это, в частности, позволяет запрашивать родителей по цепочке, чтобы получить информацию о белом пятне.

---

[2]Экономное дублирования может быть в дальнейшем реализовано на описываемой основе через создание объектов особого типа - символических ссылок на объект-время и пометок об удалении. Ссылки и пометки об удалении будут располагаться в особом системном объекте, используемом при обработке неадминистративных запросов. К сожалению, это несколько замедлит выполнение всех запросов.

1.3. Уточнение времени запроса

При запросе на чтение прошлое время, попадающее в белое пятно, по умолчанию сдвигается на начало этого пятна. Это гарантирует мгновенное получение согласованных данных. Если, например, много пользователей обращаются с одинаковыми запросами, требующими много времени на обработку, то каждый из них будет мгновенно видеть последний из имеющихся согласованных результатов и время, на которое произведена обработка. Точность реляционных СУБД при этом сочетается с оперативностью NoSQL-сервисов. Хуже если запрос не очень частый, а нужна последняя актуальная информация, тогда придётся подождать и повторить запрос. Запрос может иметь признак, указывающий на необходимость чтения ближайших актуальных, пусть и не согласованных данных либо на чтение с актуальностью в конце пятна.

При запросе на запись время также уточняется: прошлое в пределах белого пятна фиксируется как время записи с уточнением в виде настоящего времени, а прошлое или будущее время за пределами текущего белого пятна делают запись невозможной и запрос теряется.

Основное хранилище организует бинарное дерево для доступа к упорядоченному списку. Список изменений упорядочивается по времени и быстрый доступ к актуальной версии данного осуществляется делением всего упорядоченного списка изменений пополам до тех пор, пока не будет обнаружено изменение, актуальное на запрошенный момент времени. Это единственный известный способ перейти к актуальной на заданный момент времени информации для запроса (1),(4) с логарифмической сложностью.

Обработку (2)-(3) облегчает семантически насыщенная система идентификации узлов. При лексикографическом упорядочении таких идентификаторов родитель предшествует ребёнку и порядок детей одного родителя сохраняется, а идентификаторы узлов любой ветки заполняют интервал упорядоченного списка. Операция чтения дополнительной структуры на диске заменяется вычислением функции, задающей границы этого интервала, и чтением основной структуры.

2. Ветвления и их идентификация

2.1. Семантическая идентификация узлов

Система идентификации узлов, обеспечивающая связность множества идентификаторов узлов любой ветки, не является чем-то доселе неизвестным. Знакомой всем системой таких идентификаторов

является идентификация файлов полным путём в файловой системе ISO 9660. Схема проста: имена содержащих файл директорий и имя самого файла конкатенируются с разделителем '/', обладающим важным свойством: этот разделитель не должен при лексикографическом упорядочении оказаться между возможными продолжениями имени любого имени файла или директории. Именно это свойство обеспечивает связность множества идентификаторов, а его нарушение угрожает нарушением этой связности.

Возможны и альтернативные варианты без разделителей. Можно, например, условиться, что имя файла или директории всегда начинается с заглавной латинской буквы, а продолжается только строчными латинскими буквами.

Эти варианты нуждаются в поправке в нашей ситуации, когда начало ветки также является узлом, с которым могут ассоциироваться данные. Без поправки начало отрывается от ветки. Поправить ситуацию можно всегда добавляя в конец пути лексикографически большую строку, чем возможные продолжения имени. Наиболее экономный вариант поправки в первом варианте — использование в качестве разделителя большего байта, чем возможные символы продолжения имени, а во втором варианте — такое изменение кодировки, при котором заглавные латинские буквы займут позиции в конце кодовой таблицы.

Возможны и иные варианты. Постепенно выявляются ограничения, требующие учёта при выборе варианта. Одно из них — необходимость использовать буквы национальных алфавитов в идентификаторах с соблюдением алфавитного порядка [10].

Вопрос об обоснованном выделении схемы семантической идентификации узлов дерева требует таким образом отдельного исследования.

2.2. Уровни версионирования

Иерархия объектов не является единственной основой ветвления данных. Возможны дополнительные ветвления, связанные с различными версиями данных в системе

2.2.1. *Административные версии системы*

При анализе инцидентов и подготовке обновлений требуется проведение экспериментов на копии системы или её распределённой части. Многие такие эксперименты могут проводиться на имеющейся физической основе на виртуальной копии системы. При этом принципиальное отсутствие возможности изменить старые версии открывает безопасный путь создания виртуальной копии без фактического копирования данных. Речь идёт об одновременном поддержании нескольких версий состояний системы, отличающихся поздними изменениями.

2.2.2. *Управляемое пользователями версионирование*

Объект может содержать информацию, подлежащую контролю версий, например, систему управления версиями исходного кода исполняемых модулей. В этом случае разработчики выделяют основную и иные версии, которые могут иметь значительные общие части, но изменяться изолированно. Так же, как система в целом, объект сначала появляется и изменяется, а часто уже после возникает необходимость выделения веток и версий.

Универсальным решением было бы лексикографически разделить множество идентификаторов составляющих объекта и множества версий этого объекта. Если используется строка-разделитель '/', то её достаточно продублировать. Например, `name//01.01.34/a` это версия 01.01.34 части `name/a/`, а `//test/` это тестовая версия системы в целом.

2.2.3. *Автоматическое версионирование*

При обработке транзакций и других сложных вычислениях возникают версии данных, контролируемые не людьми, а вычислительными процессами. Они могут ветвиться без сознательного участия пользователей[5]. Во избежание ошибок их надо изолировать от идентификаторов, формируемых прикладными программистами. Это можно сделать символом-разделителем, не используемом в идентификаторах объектов.

2.3. Схемы версионирования

В зависимости от потребностей организаторов маркировка версий может отличаться[9]. При этом задача семантической идентификации узлов дерева версий по сути ничем не отличается от аналогичной задачи для дерева иерархии данных.

Единственный подводный камень, который здесь просматривается, — это общепринятая нелексикографическая маркировка версий одного уровня. К примеру, версия `10` лексикографически меньше, чем версия `2`. Общепринятый путь решения этой проблемы состоит в приписывании достаточного числа нулей, например, `010` уже больше, чем `002`.

Этот путь рано или поздно заведёт в тупик исчерпания доступных номеров версий. В этой ситуации обычно в пожарном порядке меняют схему именования версий вместо того, чтобы использовать простой выход из этого тупика — добавление разряда вместе с буквой. Например, `a1001` лексикографически больше, чем `999`.

Комбинирование предлагаемых конструкций для идентификаторов даёт общий подход к реализации различных комбинированных вариантов версионирования на основе B+ деревьев, обобщающий [8].

3. Направления дальнейших исследований

К сожалению, проблему идентификация ретроспективных данных удалось охарактеризовать лишь в общих чертах. Для успешной реализации прототипа системы необходимо проработать

- систему идентификации служебной информации, ссылок и пометок об удалении,
- систему идентификации элементов одномерных и разреженных многомерных и ассоциативных массивов данных,
- кодировку, удобную для реализации системы,
- формальные описания алгоритмов сохранения и чтения информации.

По итогам предварительной проработки должен быть создан и опубликован проект стандарта.

4. Выводы

(1) Выделены особенности ретроспективного подхода, перспективные для использования в будущих масштабируемых долгоживущих в сложно изменяющихся условиях системах.
(2) Описана система элементарных запросов к ретроспективному хранилищу.
(3) Описаны источники ветвления информационных структур.
(4) Сформулированы рекомендации по выбору системы идентификации объектов ретроспективного хранилища.

Список литературы

[1] Miles S., Groth P., Munroe S., Moreau L. *PrIMe: A Methodology for Developing Provenance-Aware Applications* // ACM Transactions on Software Engineering and Methodology, 2009, no. 3, p. 1-42 ↑[]

[2] Kieseberg P., Schrittwieser S., Mulazzani M., Huber M., Weippl E. *Trees Cannot Lie: Using Data Structures for Forensics Purposes* // Intelligence and Security Informatics Conference (EISIC), 2011, p. 282–285 ↑[]

[3] Nierstrasz O., Denker M., G îrba T., Lienhard A., Röthlisberger D. *Change-Enabled Software Systems* // Challenges for Software-Intensive Systems and New Computing Paradigms. LNCS 5380 : Springer Verlag, 2008, p. 64–79 ↑[]

[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 ↑[]

[5] Landes T. *Tree Clocks: An Efficient and Entirely Dynamic Logical Time System* // Proceedings of the IASTED International Conference on Parallel and Distributed Computing and Networks / ed. Burkhart H. — Innsbruck, Austria : ACTA Press, 2007, p. 375–380 ↑2.2.3

[6] Lamport L. *Time, Clocks, and the Ordering of Events in a Distributed System*, 1978. Vol. **21**, no. 4, p. 558–565 ↑[]

[7] Préz J., Arenas M., Gutierrez C. *nSPARQL: A navigational language for RDF.e* // Web Semantics: Science, Services and Agents on the World Wide Web, 2009, p. 55–270 ↑1.1

[8] Jiang L., Salzberg B., Lomet D., Barrena M. *The BT-Tree: A Branched and Temporal Access Method* // VLDB '00 Proceedings, 2000, p. 451–460 ↑2.3

[9] Zhu N. *Data Versioning Systems*, Stony Brook University, 2003 ↑2.3

[10] Знаменский С. В. *Процессный подход к эволюционированию информационной системы. Ретроспективное индексирование* // Программные системы: теория и приложения, 2011. **Т. 2**, № 4(8), с. 115–125 ↑2.1

Рекомендовал к публикации *д.ф.-м.н. Н. Н. Непейвода*

Об авторе:

Сергей Витальевич Знаменский

Автор критерия разрешимости уравнений свёртки в пространстве функций, голоморфных на множестве, понятий выпуклости в направлении и $\mathbb{C}$-выпуклости, русификации TEX для журналов Отделения математики РАН, графического пакета `mfpic3d` и нового подхода к построению информационных систем.

e-mail: svz@latex.pereslavl.ru

Образец ссылки на эту публикацию:

С. В. Знаменский. *Глобальная идентификация данных в долговременной перспективе* // Программные системы: теория и приложения : электрон. научн. журн. 2012. Т. 3, № 2(11), с. 77–86.

URL: http://psta.psiras.ru/read/psta2012_2_77-86.pdf

S. V. Znamenskij. *Global data identification in a long term perspective.*

Abstract. The retrospective approach to data storage is characterized by high data authenticity, availability and survivability of history. This particular features are promising for use in future scalable long-lived systems for difficult changing conditions.

The basic requests to the retrospective storage and possible sources of branching for the information structures are described. Recommendations are given for the choice of a global object identification system for a retrospective storage. (in Russian)

Key Words and Phrases: versioning, data storage, B+tree structure, timestamp, changes history, long term information system.