Электронная библиотека многоверсионных текстовых документов

Максим Губин

Информационная компания «Кодекс»
max@gubin.spb.ru

Аннотация

В данной статье описываются модели представления и основные сценарии доступа к многоверсионным текстовым документам.
Рассматривается несколько подходов к организации хранилища текстов. Для оптимизации изменения индекса при создании новой версии документа описывается метод разбиения текста на фрагменты с использованием хэш-функции.
В статье приводятся результаты экспериментальной оценки работы описываемых алгоритмов для двух коллекций многоверсионных документов.

1 Введение

Часто при разработке программного обеспечения электронных библиотек встает задача поддержки работы с документами, текст которых может меняться с течением времени.

Например, библиотека нормативных документов должна хранить редакции документов - многие российские кодексы имеют десятки действующих в разные периоды редакций. Системы хранения технической документация должны обеспечивать доступ к различным версиям описаний и руководств, возникающих в процессе жизненного цикла изделий и продуктов. Системы документооборота организаций отслеживают документы в процессе разработки и согласования. Таким образом, необходимость работать с многоверсионными документами возникает во многих случаях. В данной статье рассматриваются некоторые особенности организации хранилища документов, поддерживающего возможность работы с документами с множеством версий текстов.

2 Модель представления и основные сценарии доступа к документу.

В современном документообороте и электронных библиотеках, как правило, используется достаточно простая схема образования редакций текста. Документ обычно имеет одну действующую на данный момент редакцию текста и несколько устаревших редакций, которые действовали в определенный момент времени. Более сложные варианты возникновения версий, такие как деревья версий в системах управления конфигурацией исходных кодов программных продуктов[1], в документарных системах практически не используются.

Если рассмотреть возможные варианты запросов на доступ, то в порядке убывания частоты их возникновения можно выделить следующие:

1. Получить последнюю редакцию документа.
2. Получить редакцию документа, действующую в заданный период.
3. Сравнить тексты двух редакций и сформировать отчет об изменениях.

Современные коллекции текстовых документов обязательно содержат средства полнотекстового поиска. Наиболее частыми запросами являются:

1. Выполнить полнотекстовой запрос, в результат которого попадают последние редакции документов.
2. Выполнить полнотекстовый поиск, в результат которого попадают все документы, хотя бы одна версия текста которых удовлетворяет запросу.
3. Выполнить полнотекстовый поиск для текстов редакций, которые действовали в заданное время.

Поставленные задачи требуют специальных методов представления документов и организации индекса по тексту. Более подробно некоторые возможные подходы и экспериментальная оценка их эффективности описаны далее в этой статье.

3 Хранение документов

Современные электронные библиотеки и хранилища систем документооборота основываются на специализированном программном обеспечении или СУБД общего назначения. К особенностям хранения полнотекстовых документов можно отнести следующее:

1. Относительно большие объемы хранимой информации. Обычно хранятся десятки тысяч документов с объемом текста от единиц килобайт до единиц мегабайт.
2. Множество поддерживаемых форматов. В большинстве случаев в системе хранятся документы в нескольких форматах. Например, система контроля конструкторской документации может хранить текстовые описания продукции в формате текстового процессора MS Word, внешние стандарты в формате PDF и чертежи в формате AutoCAD

Можно предложить несколько вариантов реализации хранения документа с несколькими редакциями текстов, некоторые из которых описаны далее.

3.2 Редакции как независимые объекты хранилища.

В этом случае, вся логика обработки версий переносится на уровень бизнес-логики приложения. Данный подход наиболее распространен, он достаточно прост в реализации, обеспечивает высокую скорость выполнения запросов. Недостатком является необходимость реализации логики обработки версий на высоком уровне и относительно неэффективное использование дискового пространства.

3.3 Хранилища, поддерживающие многоверсионные объекты.

Подобные системы пока не получили широкого распространения, хотя экспериментальные системы, имеющие такую функциональность, известны давно[2]. Этот подход обещает достаточно эффективное хранение и доступ к версиям документа, но не может обеспечить одинаковую эффективность для различных форматов представления текстов при использовании внешних редакторов с их собственными форматами. Например, в текстовом процессоре MS Word при изменении одного символа документа файл сохраненного документа может содержать килобайты несовпадающих последовательностей байтов. Поэтому такие хранилища могут быть эффективным только в случае, если форматы представления текстов документов заранее известны и обрабатываются специальным образом. Определенные надежды в этом направление дает использование таких универсальных форматов представления информации как XML[3], тем более что все большее число приложений позволяют сохранять документы в форматах, основанных на этом стандарте[4].

Для хранения многоверсионного XML файла существует несколько известных подходов:

1. Подход, основанный на редактировании. При этом, в системе сохраняется первая версия текста документа и сценарий ее изменения (редактирования). Сценарий содержит последовательность операций редактирования, которые должны быть выполнены для получения следующей версии. Как правило, это операции вставки, замены и удаления. Сценарий либо формируется непосредственно при изменении текста программой-редактором, либо отдельной функцией, которая сравнивает две версии. Этот подход наиболее приемлем в случае небольшого количества изменений и версий. Он обеспечивает минимальные затраты дискового пространства для хранения версий при приемлемой скорости выборки версии. При большом количестве изменений сценарии становятся слишком большими и получение версии происходит медленно.

Так как версии восстанавливаются последовательно, то если документ имеет множество версий, необходимо выполнение большого количества сценариев для получения самой последней версии. Так как чаще всего требуется выборка как раз последней версии, то иногда применяется слегка модифицированный вариант данной схемы хранения, когда хранится последняя версия текста, а предыдущие восстанавливаются с использованием сценариев.

2. Подход, основанный на хранении страниц. При этом подходе документ разбивается на множество страниц и создается специальная индексная структура, хранящая на указатели на страницы. При модификации документа создаются новые страницы, которые заменяют модифицированные и создается новая индексная структура, описывающая набор страниц для новой версии. Более подробное описание подобного подхода можно найти в статьях [2,7].

При этом подходе время восстановления любой из версий одинаково, но требования к памяти для хранения при небольшом количестве модификаций документа выше, чем при предыдущем подходе.

3. В статье [3] предлагается комбинированный подход, когда документ делится на страницы, но изменения в страницах записываются с помощью сценариев редактирования. Если для данной страницы сценарий изменения становится слишком большим, то формируется новая страница и изменяется индексная структура. Тем самым используются преимущества обоих методов.

3.4 Использование средств поддержки версий приложений-редакторов.

Многие современные приложения для создания документов имеют встроенные средства работы с редакциями. Например, популярный текстовый процессор MS Word имеет возможность сохранять протокол изменений документов, аналогичные возможности имеют многие другие продукты: WordPerfect, OpenOffice. Однако данные возможности, как правило, не позволяют полностью реализовать все функции, которые нужны для работы с редакциями из-за отсутствия необходимых интерфейсов. Например, перечисленные продукты не имеют доступа к тексту предыдущих редакций, а только к последней и протоколам изменения. Поэтому, если возникает необходимость использовать встроенные средства приложений-редакторов, то разработчиками системы приходится использовать их в сочетании с одним с другими подходами. При этом средства работы с версиями редактора, как правило, прозрачны для системы хранения и поиска. То есть хранилище рассматривает текст вместе с информацией об изменении в формате приложения как один объект (файл).

Для создания отчета об изменении документа встроенные средства приложений могут быть полезными и позволяют реализовать необходимую функциональность удобнее, чем универсальные средства.

4 Индексирование документа

Одним из самых важных требований к хранилищу является обеспечение возможности полнотекстового поиска. В настоящее время наиболее распространенной структурой для поддержки поиска по тексту является инвертированный файл [5]. Эта индексная

структура достаточно хорошо изучена, однако ее использование для многоверсионных текстов пока исследована недостаточно. Из известных работ по этой тематике можно указать, например, статью [6].

Задачами, которые требуется решить, при оптимизации структуры инвертированного файла для работы с многоверсионными документами, являются следующие:

1. Обеспечение заданной производительности при выполнении запросов.
2. Минимизация операций изменения индекса при создании новой редакции текста, т.е. обеспечение заданной производительности обновления индекса при создании новой редакции.

Решение данных задач особенно важно при создании инвертированного файла, который сохраняет позиции терминов в текстах. Предположим, что при создании новой редакции текста документа была произведена вставка слова в начале документа. При этом произойдет изменение позиций всех входящих в документ терминов и, соответственно, их листов вхождения. Если документ большой и содержит много различных терминов, то это может привести к большому количеству обращений к диску, вплоть до изменения всех страниц, хранящих индексную информацию.

Решение этой проблемы основывается на том допущении, что изменения обычно затрагивают только отдельные участки документа. Документ при индексации разбивается на фрагменты с помощью специальных отметок (landmarks), и положение слов указывается относительно этих отметок. При изменении документа изменяются листы вхождения только для терминов, которые попали во фрагменты, которые затронуты редактированием.

Авторы статьи [6] предлагают следующие методы разбиения документа:

1. На фрагменты одинаковой длины;
2. Основываясь на структуре документа.

Однако, основываясь на наших экспериментах, которые описаны далее в данной статье, сделан вывод, что они не дают разбиения достаточно «устойчивого» к изменениям документа, которые встречаются в практических задачах, или требуют достаточно сложных и ресурсоемких алгоритмов учета смещения точек разбиения при создании новой версии. Предлагается новый метод выделения фрагментов, основанный на хэш-функции от скользящего по тексту окна.

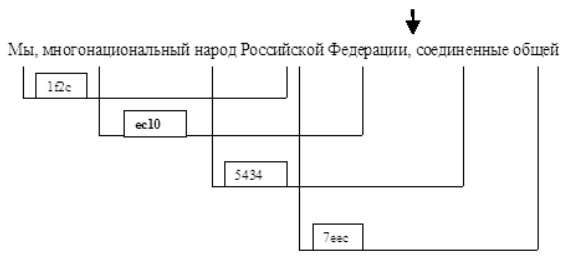

Рис. 1. Формирование точек разбиения.

Рассмотрим фрагмент текста (см. рис.1), где для каждой последовательности из n слов формируется некоторая хеш-функция. На рисунке n принято равным 4. Последовательности берутся с перекрытием, то есть формируется скользящее по тексту окно. Когда функция принимает определенное значение, то за словом, которое заканчивает последовательность, отмечается граница разбиения (landmark). В начале текста всегда устанавливается первая точка разбиения. Например, в качестве такой хеш-функции можно использовать остаток деления контрольной суммы слов последовательности на некоторую константу. На рисунке, при использовании константы 16 последнее слово второй последовательности является границей разбиения, что отмечено стрелкой.

Используемая хеш-функции должна удовлетворять следующим требованиями:

1. Вычисляться только от параметров, которые учитываются в текстовом индексе. Это обеспечивает неизменность разбиения при таких модификациях документов, которые не изменяют индексную информацию. Например, если в индексе не хранится информация о регистре букв слов, то перед вычислением хеш-функции они должны приводится к одному регистру.
2. Формировать фрагменты текста приблизительно одинакового, близкого к определенному заданному, размеру. Это требуется для того, чтобы минимизировать число измененных фрагментов при точечном изменении текста. Например, можно слегка модифицировать описанную выше хэш-функцию и добавить условие, что от предыдущей точки разбиения прошло не менее 7 слов, и всегда вставлять точку разбиения, если она не встретилась течение последних 256 слов.

Используя для слов листы вхождения, содержащие позиции относительно landmarks можно эффективно реализовать требуемые поиски по версиям текстов документов. При этом каждая позиция слова в инвертированном файле описывается парой: идентификатор точки разбиения, позиция в разбиении. При этом необходимо хранить информацию о последовательности идентификаторов точек разбиения для каждой версии текста. Для этого авторы статьи [6] предлагают специальную структуру «каталог landmarks», которая описывает порядок вхождения точек разбиения. Там же описаны алгоритмы выполнения поисковых запросов при такой организации индекса и приведена оценка производительность операции обновления инвертированного файла при создании новой версии.

Существенным недостатком приведенных в указанной статье алгоритмов является то, что наиболее часто выполняемые запросы к последним редакциям документов требуют обращения к дополнительной структуре (каталогу landmarks), что приводит к снижению производительности. Учитывая то, что в большинстве

запросовуказывается небольшое расстояние между словами (обычно менее 7), можно изменить алгоритм нумерации позиций и, введя некоторую избыточность, обеспечить обработку таких запросов без обращений к дополнительным структурам. При этом позиции слова указываются не только относительно последнего landmark, но, для слов, которые отстоят от него не более чем на k слов, относительно предыдущего. Это позволяет выполнение запроса «найти документы содержащие слова, отстоящие не далее k" свести к выполнению запроса «найти фрагменты документов, содержащие слова не далее к». Искомая фраза будет целиком содержаться в одном из фрагментов, даже если она разбита landmark, за счет указания позиций после разбиения. При этом несколько увеличивается размер индекса, т.к. для многих слов позиции указываются несколько раз, но правильный выбор алгоритма сжатия листа вхождения позволяет минимизировать это увеличение.

Кроме описанного изменения нумерации позиций слов, для того, чтобы поиск по последним редакциям не требовал дополнительных обращений к вспомогательным структурам, в листе вхождения для каждого идентификатора landmark можно добавить флаг принадлежности к последней версии. При этом увеличится объем обновления индекса, т.к. потребуются дополнительные обращения для снятия этого флага.

5 Экспериментальная оценка

Для экспериментальной проверки описанных в данной статье подходов использовались коллекция российского законодательства и технических документов.

Коллекция российского законодательства содержала порядка 70000 документов со средним размером 4 Кб, из них около 5000 документов содержали в среднем 3 редакции.

Коллекция технических документов содержала 40 документов технорабочих проектов разработки информационных систем средним размером 800 Кб, каждый из которых имел в среднем 5 редакций.

Первая серия экспериментов должна была выяснить накладные расходы при использовании различных схем хранения редакций,. При этом документы были преобразованы в XML формат.

Экспериментальное программное обеспечение реализовывало хранилище, в котором сохранялись коллекции документов. Было реализована два варианта хранения версий:

1. Версии текстов сохранялись отдельно в виде файлов, сжатых с помощью библиотеки zlib (http://www.gzip.org/zlib/);
2. Все версии текстов сохранялись в одном объекте хранилища с формированием лога изменения, позволяющего восстановить отдельные версии. В качестве хранилища использовалась система хранения версий файлов CVS (http://www.cvshome.org). Так как данная система не поддерживает сжатия, то она была модифицированна таким образом, что могла сохранять файлы в сжатом формате.

Для каждого варианта измерялось дисковое пространство, которое требовалось для сохранения коллекции, время загрузки хранилища и время выборки всех версий всех документов хранилища. Результаты представлены в таблице 1.

Таблица 1

Вариант системы	Объем на диске (МБ)	Среднее время выборки версии (сек)
Первая коллекция		
Сжатые файлы	445	0.15
CVS без сжатия	756	0.21
CVS со сжатием	385	0.43
Вторая коллекция		
Сжатые файлы	12	0.8
CVS без сжатия	14	1.7
CVS со сжатием	7	2.4

Из результатов видно, что преимуществом сложных схем хранения является экономия дискового пространства. Но оно заметно проявляется (составляет около 60%) только для коллекций больших документов. Недостатком сложных схем хранения при этом значительно (в 2-3 раза) возрастает время доступа к версиям.

Для исследования алгоритмов индексирования изменяемых документов были проведены эксперименты, которые должны были выяснить, насколько устойчивы различные схемы разбиения документа к типичным изменениям.

В ходе эксперимента оценивалось количество изменившихся позиций слов при переходе от редакции к редакции документа. При этом использовались следующие виды разбиения:

1. На блоки фиксированной длины размером 64 слова;
2. По элементам форматирования – абзацам текста;
3. С помощью предложенного метода генерации точек разбиения со средней длиной фрагмента 128

В отличие от статьи [6] мы не использовали diff алгоритм для учета возможных смещений блоков фиксированной длины, т.к. это значительно усложнило реализацию каталога landmark, замедляло индексирование коллекции и усложняло обработку запросов. При нашей реализации каталог landmarks содержит только последовательность идентификаторов фрагментов для каждой из версий. При сохранении новой версии текст разбивается на фрагменты. Не изменившиеся части, получали один и тот же идентификатор landmark, изменившиеся фрагменты получали новые идентификаторы. В Таблице 2 приведено отношение количества измененных позиций к общему количеству позиций в документе при создании новой версии:

Таблица 2

Метод разбиения	% измененных позиций	
	1 коллекция	2 коллекция
Блоки по 64 слова	60	75
По абзацам	30	15
С помощью хэш-функции	10	12

Высокий процент изменений при разбиении по блокам фиксированной длины обусловлен тем, что практически всегда при изменении документов происходит смещение текста.

Большое количество изменений в первой коллекции при разбиении по абзацам вызвано тем, что правовые документы имеют достаточно большие абзацы, в которых часто в конце помещаются комментарии, связанные с изменением версии.

Разбиение с помощью хеш-функции, как оказалось наиболее устойчивым к типичным изменениям, производимым на практике, и тем самым минимизирует изменения инвертированного файла, построенного на его основе.

6 Выводы и перспективы

Предложенные алгоритмы и их экспериментальное исследование позволяют сделать следующие выводы:

1. Как показывают экспериментальные данные, современные алгоритмы сжатия и низкая стоимость памяти приводят к тому, что выигрыш от сложных схем хранения версий незначителен. Поэтому, на практике, целесообразно в аналогичных хранилищах текстов просто представлять версии как отдельные объекты. Область применение более сложных методов хранения версий - обеспечение минимального объем передаваемой информации, а время формирования версии текста не критично, например, при передаче по каналам связи.
2. Реализация инвертированного файла, когда листы вхождения слов содержат позиции в текстах относительно специальных отметок, устойчивых к изменениям версий, позволяют значительно уменьшить объем изменяемой индексной информации при создании новой версии текста документа. Эксперименты показали, что в этом случае лучшим является формирование отметок с использованием хэш-функции от скользящего по тексту окна. Предлагаемая реализация инвертированного файла позволяет эффективно выполнять полнотекстовые запросы к редакциям текстов документов.

Описанные в статье решения предполагается использовать в коммерческих версиях электронных библиотек и хранилищ систем документооборота.

Литература

[1] SteveBerczuk (berczuk@acm.org), “Configuration Management Patterns” http://www.bell-labs.com/cgi-user/OrgPatterns/OrgPatterns?ConfigurationManagementPatterns

[2] M. Carey, D. DeWitt, J. Richardson, and E. Shekita,"Storage Management for Objects in EXODUS", in "Object-Oriented Concepts, Databases, and Applications",W. Kim and F. Lochovsky, eds., Addison-Wesley Publishing Co., 1989.

[3] Chien, S.Y., Tsotras, V.J., Zaniolo, C.: “XML Document Versioning.” SIGMOD Record 30 (2001) 46--53 http://citeseer.ist.psu.edu/chien01xml.html

[4] Chuck Urwiler “Working with Microsoft Office Word 2003's XML” http://www.devx.com/codemag/Article/18227

[5] Baezo-Yates R. and Ribeiro-Neto B., “Modern Information Retrieval” ACM Press Addison Wesley, 1999

[6] Lipyeow Lim, Min Wang, Jeffrey Scott Vitter, Sriram Padmanabhan, Jeffrey Scott Vitter, Ramesh Agarwal, “Dynamic Maintenance of Web Indexes Using Landmarks” WWW2003, May 20-24, 2003, Budapest, Hungary. ACM 1-58113-680-3/03/0005.

[7] Б.А. Новиков Системы хранения баз данных и знаний, Программирование 1992

Digital library of multiversion documents.

Maxim Gubin
Kodeks Company
max@kodeks.ru

Processing versions of text documents is a critical task for many applications, such as document workflow or legal information systems. In this article we analyze two problems of document versioning: version storage and fulltext indexing.

Many different storing schemas for multiversion data are proposed. However, they are rarely used in text storage systems. Our experiments show that these complex techniques make the storage and retrieval of big documents too slow. We have achieved better results using brute force method of storing versions of a document as separate objects compressed by conventional LZSS algorithm.

The main problem of indexing of multiversion documents is that small changes in a text can cause dramatics changes in fulltext index structures. Imagine that a word is inserted in the beginning of a long document that contains many words. In that case, all lists of positions of words from the document must be changed. Thus, insertion of one word, cause changes in many pages of the index structure. We use a special method of indexing of word’s positions in a text relatively to special points in text or “landmarks”. In our example, the word insertion affects only the positions that are coded relatively to the first landmark. The selection of the landmarks must ensure minimal changes of a word’s positions index caused by typical text changes. We propose a new algorithm of the landmark generation that uses a special hash function from words of sliding text window. Our experiments show that the proposed method reduces the index changes significantly.

The disadvantage of the proposed method is an additional structure – landmark directory that maps landmarks to positions of words in a version. Access to the directory reduces productivity of index searches. To improve the situation we propose two methods that make possible to process typical queries without usage of a landmark directory. First is a special word position numbering scheme, second is additional flags of the last version in post list. These methods only slightly increase the index size and update time, but noticeably decrease a query processing time.

We plan to use the proposed algorithms in commercial multiversion document storage system.