Московский Физико-Технический Институт (Государственный Университет)
Факультет Прикладной Математики и Экономики
Кафедра Информатики

Построение индекса файловых ресурсов поисковой системы мультимедийной информации для локальных сетей

Выпускная квалификационная работа
на степень магистра
студента 273 группы
Басанова Тимофея Вячеславовича

Научный руководитель: Устюжанин Андрей Евгеньевич

МОСКВА 2008 г.

Оглавление

Введение

С развитием цифровых технологий за последние десять лет у людей появилась возможность хранить на своём компьютере целые собрания музыкальных записей, десятки фильмов, тысячи изображений. Локальные компьютерные сети, получившие большое распространение с развитием интернета, открыли пользователям доступ не только к медиа файлам, хранящимся на жестких дисках их персональных компьютеров, но и к огромному количеству файлов из библиотек других пользователей.

В этих условиях крайне актуальной стала задача создания поисковых систем, значительно упрощающих доступ к медиа информации, размещённой на других компьютерах в локальной сети. Существующие на сегодняшний день поисковые системы для локальных сетей не могут решать типичные задачи пользователя, такие как поиск конкретного музыкального альбома некоторого исполнителя или поиск свежей серии любимого мультипликационного сериала. Весьма необходимой стала система, которая сможет решить ту или иную задачу пользователя за приемлемое время, дав ему возможность естественным образом сформулировать свой запрос.

Например, поисковая система локальной сети ФМТИ «AllTer» по запросу пользователя «Madonna Music», выдает многостраничный неупорядоченный список из 70 результатов, остоящий из перечисления всех найденных в локальной сети файлов, содержащих «madonna» и «music» в качестве подстроки в имени файла. В результате пользователь вынужден просматривать страницу за страницей и в конце концов может так и не найти нужного файла, поскольку искомый mp3 с композицией «Music» певицы Madonna может называться «song_01.mp3», а информацию об имени

исполнителя и названии песни содержать в специальной области внутри файла, называемой ID3 тегом.

Нами впервые ставится задача создания поисковой системы по локальной сети, выдающей на подобный запрос небольшой упорядоченный по релевантности список результатов, каждый из которых обозначает конкретный медиа файл (фильм, песню, видеоклип и т.д.) и аггрегирует в себе файлы, различающиеся по названию, качеству содержащейся записи. Новизна задачи определяется нашим новаторским подходом к анализу потребностей пользователя, которые связаны с поиском медиа файлов в локальных сетях.

Серьезные ограничения, связанные с далеко не безграничной вычислительной мощностью технической платформы, доступной для подобной системы из соображений экономической целесообразности, делают проектирование одной из самых важных подистем поисковика – поискового роботоа для индексирования локальной сети – сложной обособленной задачей, заставляют использовать для достижения цели принципиально новую архитектуру робота и разрабатывать новые алгоритмы, при этом, однако, опираясь на существующие достижения в области индексирования информации.

1.1 Постановка задачи

Цель данной работы – на основании потребностей пользователя и технических ограничений сформулировать и формализовать требования к поисковой системе медиа файлов в локальной сети, проанализировать существующие решения, функционирующие в различных поисковых

системах, разработать поискового робота для индексации содержимого локальной сети и поисковые алгоритмы, запрограммировать и внедрить данный модуль поисковой системы.

2. Описание предметной области

2.1 Медиа

За последнее десятилетие с развитием цифровых технологий коренным образом поменялся способ, которым люди ищут, потребляют, организуют и хранят аудио- и видеопродукцию: музыкальные композиции и клипы, полнометражные фильмы, развлекательные программы и ролики.

До конца прошлого столетия самой распространенной формой хранения аудио- и видеозаписей были такие аналоговые носители, как пластинки, кассеты. Как правило, такие носители дорого стоили, не позволяли создавать копии записи или допускали, но с потерей качества, имели материальную форму и заставляли передавать носитель «из рук в руки» или по почте, чтобы поделиться с кем-нибудь. Часто по этим причинам потребитель был вынужден предпочесть просмотру или прослушиванию записи «on-demand» (по требованию) посещение кинотеатра, музыкального концерта или был вынужден слушать или смотреть передачи, заданные программой по радио или телевидению.

Все перечисленные ограничения исчезли с появлением дешевых цифровых носителей, компьютеров и компьютерных сетей.

Дадим определение понятию *медиа файл*, соответствующее наиболее распространенной на текущий момент форме хранения аудио- и видеопродукции. Под медиа файлом в данной работе будем понимать цифровую аудио- или видеозапись, представленную в виде последовательности байт и сохраненную в виде файла на цифровом носителе или жестком диске компьютера. Кроме того оговоримся, что даже файлы,

расположенные на разных компьютерах и содержащие запись одной и той же композиции, но в разном качестве, будут являться экземплярами одного и того же медиа файла.

В настоящее время большинство обладателей персональных компьютеров имеют возможность собрать на жестком диске своего компьютера собственную коллекцию музыкальных композиций и видеозаписей в виде файлов распостраненных форматов.

На данный момент наиболее распространенный музыкальный формат mp3 позволяет хранить аудиокомпозиции в высоком качестве. При этом средний размер файла при длительности композиции 3-7 минут составляет 5-10 мегабайт.

Средний размер файла с видеозаписью художественного фильма, сжатого одним из распостраненных кодеков, варьируется от 700 до 1500 мегабайт. Размер музыкального клипа варьируется от 20 до 100 мегабайт.

Как правило, медиа библиотека пользователя персонального компьютера включает в себя несколько сотен музыкальных композиций и 5-10 фильмов.

2.2 Локальные сети

В настоящее время большая часть пользователей домашних компьютеров имеют доступ в интернет, часто организованый провайдером посредством локальных сетей, обединяющих подключенные компьютеры в одном районе.

Такие локальные сети состоят, как правило, из нескольких сотен компьютеров. В локально сети МФТИ, например, количество компьютров

составляет в среднем полтысячи.

В таких локальных сетях пользователям доступны такие средства обмена файлами, как Samba, FTP, DC++. Пользователь может открыть часть своей файловой системы в общий доступ для чтения и/или записи, а также осуществлять удаленную навигацию по файловой системе других компьютеров с возможностью загрузить себе тот или иной файл.

В развитой локальной сети городского провайдера или университетской локальной сети одновременно в открытом доступе может находиться до нескольких десятков миллионов файлов и папок, при этом до двадцати процентов файлов, как правило, приходится на медиа ресурсы, в основном на музыку в формате mp3.

В локальной сети МФТИ к середине мая 2008 года насчитывалось 500 компьютеров, на которых было открыто приблизительно 20 миллионов файлов и папок, полтора миллиона из которых были файлами с аудиозаписями в формате mp3.

Отметим также некоторые неотъемлемые недостатки локальных сетей, создающие проблемы рядовому пользователю:

- Обычные пользователи настольных ПК часто выключают компьютеры на ночь или перезагружают при необходимости. Естественно в этот момент зайти на компьютер другому пользователю становится невозможно. Кроме того, самый популярный клиент для навигации по Samba сетям проводник Windows вызывает подвисание длительностью несколько секунд, в случае если удаленный компьютер не откликается.
- В зависимости от качества сетевого оборудования, и вычилительной мощности компьютеров в сети скорость соединения даже в рамках

одной сети может быть сильно неоднородной и варьироваться от 1 до 100 мегабит/сек.

2.3 Задачи пользователя в сети

При наличии такой широкой по возможности инфраструктуры, как развитая локальная сеть, у пользователя появляется желание потреблять медиа не только со своего локального копьютера, но и без задержки по времени пользоваться коллекциями музыки и фильмов своих соседей, знакомиться с новыми подборками видеоклипов своих друзей, по мере появления смотреть свежие серии телевизионных сериалов, выкладываемые другими пользователями на серверах общего доступа. Файловые системы и протоколы, обслуживающие локальные сети, не позволяют выполнять никакую навигацию по открытым в общий доступ файлам, кроме как навигацию по древовидной структуре файлов и папок, и не предоставляют возможности поиска, в должной мере удовлетворяющего потребности пользователя и быстро и безболезненно решающие конкретные повседневные задачи.

Наш проект, составной частью которого и является данная научная работа, ставит своей целью решить данную проблему наилучшим образом – создать индексирующий механизм и поисковую систему, дополняющие обычную локальную сеть столь недостающим интерфейсом медиа поиска и удобной навигации.

На словах, как правило, задача поиска медиа файлов в локальной сети формулируется очень просто. Приведём пару примеров:

1) Пользователь хочет скачать на свой mp3-плеер песню певицы Madonna

«Music», а, возможно, и весь альбом с одноименным названием после краткого знакомства.

2) Пользователь хочет проверить появилась ли в сети последняя серия из его любимого сериала Lost и, если да, то тут же посмотреть ее.
3) Пользователь хочет подобрать себе какой-нибудь фильм на вечер, руководствуясь своими предпочтениями (жанр, страна-производитель, актеры из фильма, популярность/оценка фильма теми, кто его уже посмотрел).

В условиях типичной локальной сети задача построения удовлетворяющей этим требованиям поисковой системы оказывается весьма непростой. Перечислим некоторые из возникающих проблем:

1) Даже не очень большие локальные сети могут содержать огромное, с точки зрения индексирующего агента или поискового агента, количество индексируемых или поисковых сущностей – файлов и папок. Например, видеохостинг YouTube в настоящее время обслуживает приблизительно 90 миллионов видеофайлов, при этом работоспособность сервиса, в том числе возможность поиска роликов по ключевым словам, обеспечивают кластера из тысяч компьютеров. Базы данных крупных финансовых корпораций, содержащие в своих недрах объемы данных, сравнимые по количеству сущностей с количеством фалов в локальной сети, обычно размещаются на дорогих кластерных многопроцессорных системах.
2) Как правило, количество денежных средств, которые пользователи готовы потратить на оборудование локальной сети поисковой системы, весьма не велико, что влечет за собой серьезное ограничение на

аппаратную часть поискового сервера, а следовательно, и на жесткие рамки с точки зрения вычислительной мощности системы, объема оперативной памяти и скорости жестких дисков.

3) Локальная сеть не является по своей природе централизованной или стабильной системой. Отдельные компьютеры или целые подсети на некоторое время могут оказаться недоступными из-за сбоев. Система должна упрощать работу пользователя в таких ситуациях, следить за состоянием сети и не показывать в результатах поиска ресурсов, расположенных на недоступных компьютерах.
4) Сеть не является однородной по скорости и время доступа к различным её узлам различается в десятки раз. Индексирующий механизм должен учитывать такую разнородность и правильно распределять вычислительные ресурсы между медленными и быстрыми (с точки зрения скорости соединения) машинами, чтобы максимизировать релевантность хранимого индекса ресурсов локальной сети.
5) Отобразить пользователю оптимальный корректно упорядоченный набор результатов в ответ на поисковый запрос, состоящий из нескольких поисковых слов – отдельная непростая задача.

2.4 Поисковые системы.

Безусловно, задача поиска отнюдь не нова как с математической, алгоритмической так и с тактехнологической точки зрения. Существует большой набор разделов вычислительной математики, теории графов, дискретного анализа посвященных решению различных составляющих частей этой задачи. Если обратиться к исследованию существующих

постановок задачи поиска и перечислить их прикладные решения, то можно смело выделить три наиболее близких к области поставленной нами задачи. Рассмотрим их, выясним область применения, проанализируем математическую и алгоритмическую составляющие и попробуем применить удачные решения из этих поисковых систем к нашей задаче.

2.4.1 Поисковые системы интернета

Прежде всего, сформулируем задачу, которую пытается для себя решить типичный пользователь поисковой системы в интернете. Как правило, пользователь хочет по нескольким ключевым словам найти веб-страницу, которая наиболее сильно ассоциируется с его словами в запросе либо наиболее точно отвечает на прямой односложный вопрос, заданный в поисковом запросе. Из-за огромных масштабов информации, которые на сегодняшний день содержит интернет, практически на любой заданный несколькими словами запрос можно найти тысячи страниц, содержащих эти слова, а чаще и целые выражения из запроса. Таким образом, ключевыми задачами поиска в интернете становятся задачи оптимального ранжирования – упорядочивания результатов поиска таким образом, чтобы в числе первых 5-10 удовлетворяющих запросу страниц отображенных в результатах поиска нашлись те, которые решат потребность пользователя, ответят на его вопрос или приведут на конкретную искомую пользователем страницу.

Приведём пример. Одним из самых популярных запросов к поисковой системе «Google» – это запрос «Britney Spears» (это имя и фамилия популярной американской певицы). Как правило, пользователь, задавший такой запрос, хочет немного узнать о самой певице, о последних новостях, которые с ней связаны, послушать её последние песни, приобрести альбом

либо просто попасть на официальную страницу. Этому запросу удовлетворяют порядка 100 миллионов страниц в интернете, и цель поисковика показать в реультатах только самые релевантные из них, угадать, что пытался найти пользвователь, ведь он точно не жаждил прочесть все 100 миллионов страниц.

Поиск в интернете несколько лет назад превратился в очень привлекательную многоообещающую высокотехнологичную отрасль, давшую жизнь новым компаниям-гигантам, которые теперь сражаются за огромные многомиллиардные прибыли и задают новые стандарты в интернете. Технологическая составляющая основного продукта этих компаний – поисковая инфраструктура, а также высококвалифицированный персонал способный вести научные разработки и обеспечить развитие системы.

Самый популярная на сегодняшний день поисковая система – это интернет поисковик «Google.com». Восхождение компании на вершину поискового олимпа началось с внедрения принципиально новой системы ранжирования результатов поиска, названной PageRank именем его изобретателя, основателя компании Google Ларри Пейджа. Подход заключается в вычислении рейтинга, и называемого PageRank, для каждой страницы в интернете на основании указывающих на эту страницу ссылок, с учетом их весов, вычисляемых по PageRank для страниц, с которых указывают ссылки. Упрощенная формула для рассчета PR (PageRank) для страницы выглядит так

$$PR(A) = (1-d) + d\ (PR(T1)/C(T1) + ... + PR(Tn)/C(Tn)),$$

где PR(A) — это вес PageRank страницы A (тот вес, который мы хотим

вычислить),

d — это коэффициент затухания, который обычно устанавливают равным 0.85,

PR(T1) — вес PageRank страницы, указывающей на страницу A,

C(T1) — число ссылок с этой страницы,

PR(Tn)/C(Tn) означает, что мы делаем это для каждой страницы, указывающей на страницу A.

Данная формула не является явной, а определяет систему уравнений, размер которой – количество всех страниц в интернете. Приближенное решение данной системы и было предложено основателями Ларри и Сергеем, и было положено в основу их поисковой системы.

Подход аналогичный PageRank можно использовать и для других систем, где между объектами, которым необходимо присвоить рейтинг, есть направленные смысловые связи, повышающие важность объекта на которую направлена такая связь.

Наибольшее отличие таких сред для построения поисковых систем, как интернет и локальные сети, заключается, прежде всего, в их масштабах. Причем гораздо ощутимее различие не в масшабах данных, обрабатываемых поисковыми системами, а в размахе технической базы, на которые может рассчитывать поисковик. В отличие от интернет-поисковиков, которым предоставленные серверные фермы из тысяч компьютеров, поисковик по локальной сети может рассчитывать лишь на один весьма ограниченный по вычислительной мощности сервер. Это застявляет использовать новые подходы при проектировании сиситемы.

2.4.2 Энтерпрайз поиск

В сетях больших корпораций с определённого момента возникает задача поиска по внутренним ресурсам, таким как корпоративный сайт, внутренняя документация, данные ERP и прочих управленческих систем.

Закрывают данную потребность продукты созданные либо компаниями, специализирующимися на поиске в интернете, такими как Google с их «Google Mini» или Яндекс с их «Яндекс Сервером», либо компании, которые изначально обслуживали внутренние порталы и/или документооборот компании, такие как IBM с их продуктом Lotus или Microsoft и продукт SharePoint.

Этот тип поиска коренным образом отличается от поиска в интернете тем, что объём данных, на которых надо искать, достаточно мал. К примеру, «Google Mini» рассчитан на 20-300 тысяч документов. Поэтому алгоритмы ранжирования и PageRank отходят на второй план. Самыми востребованными становятся системы, позволяющие устанавливать политики разграничения доступа.

В условиях локальных сетей указанные готовые программные продукты не применимы, поскольку на значительно меньшее количество поисковых сущностей и ориентированны на документы, а не медиа файлы. Также они плохо учитывают особенности больших локальных сетей, такие как разнородность, дублирование информации, ненадёжность соединений.

2.4.3 Поисковые системы в локальных сетях

На текущий момент во всех сколько-нибудь крупных локальных сетях есть свои поисковые системы. «Allter» (ФМТИ), «Lorien» (МГУ),

«search.corbina.ru» (интернет-провайдер в городе Москва), поисковая система, интегрированная в DC++.

Все эти системы обладают сходным внешним интерфейсом и часто похожим функционалом. Пользователь может найти все названия файлов в сети, которые содержат заданную пользователем подстроку. На странице результатов обычно также указывается находится ли найденный файл на включенном или выключенном компьютере. Время отклика этих систем на запрос обычно удовлетворительно – в пределах 10 секунд. Сортировка результатов по релевантности, как правило, не производится. Большим минусом, свойственным этим поисковым системам, является то, что время, за которое поисковик замечает изменения на каком либо компьютере и обновляет поисковый индекс, составляет несколько часов или даже дней, поэтому поисковые результаты далеко не всегда соответствуют действительности. Кроме того, такие поисковые системы как правило не очень дружелюбны по отношению к пользователю. Поиск по подстроке в имени файла, это скорей инструмент в руках системного администратора, а не удобный поисковик в руках рядового пользователя.

Эти и другие недостатки локальносетевых поисковых систем обычно являются последствиями компромиса между функциональностью низким потребностям к вычислительным ресурсам. Системы, как правило, обслуживают большие сети с десятками миллионов файлов и тысячами компьютеров и способны работать на весьма скромной технической базе – практически обычном персональном ПК.

2.4.3.1 Поисковая система МФТИ «AllTer»

Остановимся подробнее на поисковой системе по локальной сети

МФТИ «AllTer».

Кратко опишем ее устройство. Поисковые роботы в несколько потоков обходят сеть и собирают информацию о найденных на компьютерах сети файлах и папках. После того, как построен полный снимок древовидной структуры сети по нему создаётся индекс. Эта операция может длиться несколько часов. После этого старый индекс замещается на новый.

Индекс, в свою очередь устроен следующим образом: для каждого имени файла запоминаются все его четырёхбуквенные сочетания и создаётся обратный индекс четырёхбуквенных сочетаний *[Сегалович]*. После чего для каждого четырёхбуквенного сочетания запоминается кол-во ссылок на имена файлов. Алгоритмы, обслуживающие индекс запрограммированы на языке С. Почти весь индекс хранится в оперативной памяти; вся остальная информация хранится в базе данных MySQL.

Когда приходит запрос пользователя анализируется частоты всех содержащихся в нем четырёхбуквенных сочетаний. После чего извлекаются имена всех фалов, соответствующих самому редкому сочетанию и анализируются на соответствие запросу. Первые 50-100 удовлетворяющих запросу результатов, отображаются пользователю.

Главный минус поисковой системы заключается в отсутствии упорядочивания результатов по релевантности.

2.4.3.2 DC++

Система имеет клиент-серверную архитектуру. Каждый заинтересованный пользователь сети устанавливает на свой компьютер программу-клиент, которая индексирует открытые пользователем в общий доступ папки. Также клиентские программы поддерживают связь с

серверной частью приложения.

При выполнении поиска в DC++ поисковый запрос пользователя отправляется с его клиента серверу, после чего транслируются всем клиентам в сети. На каждой клиентской машине локальные ресурсы проиндексированы. Все удаленные клиенты отправляют результаты поиска по локальным открытым в общий доступ ресурсам обратно на сервер, после чего тот группирует ответ со всех клиентов и отправляет его запросившему пользователю.

Перечислим основные минусы данной системы:

- требует установки клиентской программы; ресурсы, размещенные на компьютерах пользователей, не установивших клиентское ПО, не видны.
- Производительность поисковой системы завязана на производительность клиентских ПК.

2.5 Ограничения, накладываемые оборудованием

2.5.1 Жесткие диски

Объем современных жестких дисков достигает нескольких терабайт. При проектировании системы будем считать, что в нашем распоряжении в пределах одной машины есть 1-2 недорогих жеских диска объёмами пол терабайта, скоростью вращения шпинделя 7200 оборотов в минуту, и скоростью линейного чтения 50 Мб/секунду.

Время произвольного доступа для современных дисков варьируется от 3 до 15 мс. Для оценок времени работы алгоритма, возьмем среднюю для

недорогих моделей величину 8мс.

2.5.2 Оперативная память

Объем оперативной памяти в компьютере, обслуживающем поисковую систему будем считать ограниченным двумя гигабайтами.

Время произвольного доступа к ячейке памяти измеряется в десятках наносекунд, что почти на 3 порядка меньше скорости доступа к произвольному сектору жесткого диска. Этой оценки достаточно в рассчетах.

2.5.3 Сеть

Локальная сеть вносит два основных ограничения на доступ к удалённым компьютерам: задержки, когда от запроса удалённых данных до их получения проходит от 1 до 10 миллисекунд в зависимости от качества сети, и ограничения на пропускную способность, когда скорость взаимодействия сервера с локальной сетью не превышает скорость его подключения к локальной сети, то есть от 100 мегабит до 1го гигабита, а скорость взаимодействия с каждым конерктным компьютером, которая зависит от удалённости компьютера от сервера и качества сети, составляет от 1 до 100 мегабит.

2.5.4 Процессор

Как показало исследование созданного прототипа, алгоритмы сканирования сети не вносят существенного вклада в потребление процессорного времени, так как обеспечение работы сетевых протоколов задача куда более ресурсоёмкая, чем задача обработки полученных данных. Поэтому процессор не становится узким местом работы алгоритмов, к

примеру, процессор Pentium-III 800 смог обеспечить загруженность внешнего сетевого канала порядка 20 мегабит.

2.6 Технологическая среда

Изначальная нацеленность данной работы на прикладное использование задает необходимость разрабатывать и имплементировать алгоритмы в рамках инструментов и ограничений, предоставляемых технологической средой, а именно такими ее составляющими частями как язык программирования, модель работы с оперативной и долгосрочной памятью.

2.6.1 Язык программирования

В настоящее время языки программирования для современных компьютернов практически не создают каких-либо ограничений, касательно эффективности и сложности алгоритмов, которые можно запрограммировать с их помощью.

При выборе языка для программирования нашей поисковой системы, мы остановили свой выбор на языке программирования Java версии 1.5, поскольку это позволило нам убрать ограничения на выбор операционной системы для работы нашего приложения.

2.6.2 Доступ к долгосрочной памяти

Поисковой системе, работающей с такими большими объемами информации, как в нашем случае, безусловно, необходимо использовать долгосрочную память, чтобы хранить массивы данных поисковых

результатов и поисковые индексы.

Языки программирования предоставляют различный уровень доступа к жестким дискам:

- Прямой доступ. Позволяет отдать команду считать или записать определенный сектор с любой дорожки диска.
- Доступ через файловую систему посредствам вызовов, предоставляемых операционной системой. Позволяет считать или записать массив байт, начиная с некоторого места внутри файла.
- Доступ через СУБД *[Бишоп]* – системы управления базами данных. СУБД предоставляют возможность записывать и извлекать информацию, заранее структурированную в виде организованного набора данных. Иными словами позволяет сохранять информацию об определенной сущности не на языке байт, а используя набор зарактеризущих ее полей различных типов данных. Кроме того СУБД предоставляют доступ к структурам данных, таким как Б-дерево, полнотекстовый индекс позволяющим организовать быстрый поиск по большим массивам информации.

При проектировании алгоритма выбор уровня абстракции доступа к жесткому диску не имеет принципиального значения. Более высокоуровневые уровни доступа, такие как файловая система и тем более базы данных создают дополнительные накладные расходы, но в нашем случае, как и в большинстве других, они пренебрежимо малы в сравнении с затратами на выполнение основных операций алгоритма.

В нашей системе мы использовали доступ через СУБД, в частности MySQL *[Шварц]*. Доступ через СУБД автоматически предоставляет следующие преимущества, повышающие надежность системы и упрощающие разработку:

- транзауционная модель и защита от сбоев
- кеширование
- одновременный доступ и блокирование
- встроенные механизмы индексации

СУБД MySQL – одна из лидирующих системм управления базами данных, распространяется с открытым программным кодом по GPL лицензии.

2.6.3 Доступ к удалённым компьютерам

Протокол Samba используется по-умолчанию для Windows систем, что делает его самым распространённым и широкоиспользуемым протоколом. Это наиболее удобный протокол для обмена файлами в локальных сетях на текущий момент благодаря тому, что он глубоко интегрирован в Windows и не требует особой настройки. Далее в тексте этот протокол будет называться самба.

Протокол самба позволяет использовать удалённые диски как свои локальные в пределах локальной сети. В том числе поддерживает удобный random read файлов, что позволяет смотреть видео-файлы и слушать музыку не скачивая файл предварительно на компьютер.

Важные для нас технические детали. Для того, чтобы установить соединение с удалённым компьютером вначале идёт аутентификация и

авторизация пользователя, этот процесс довольно сложен и длителен, он может занимать до нескольких секунд. После этого можно получить список шар (share) или корневых папок документов общего пользования. Для каждой из них уже можно получить список файлов и папок. А для каждой папки список её файлов и подпапок. Для каждого файла можно прочитать любую его часть, но обычно передаваемая информация буфферизуется блоками по 8-64кб.

2.7 Постановка задачи в контексте предметной области

Целью данной работы мы ставим проектирование одной из самых важных составных частей поисковой системы для локальных сетей – поискового робота для индексирования сети.

Рассмотрим исходные условия, в рамках которых должны работать индексирующий и поисковый механизмы. Одним из основных желаний пользователя является желание получать как можно более свежую информацию о состоянии файловой сети *[Чернецов]*. Пользователь хотел бы получать информацию о последних изменених сети через минуты.

Как правило крупные локальные сети состоят сегментов, размера порядка 1000 копьютеров, где внутри каждого сегмента все компьютеры видят друг-друга и могут обмениваться файлами напрямую. Практика показывает, что порядок величин количества файлов – 20млн, количества папок – 2млн, количества медиа файлов – 2млн. Из анализа предметной области мы также знаем, что получение списка файлов папки от удалённого компьютера с уже установленным подключением занимает в среднем 15мс, а получение деталей мета-информации из медиа файла требует порядка 150мс.

Для того, чтобы сканировать сеть сервера используют поисковых роботов (crawlers) *[Сегалович]*, которые представляют собой однотипные потоки выполнения, каждый из которых рекурсивно обходит файловые системы удалённых компьютеров и создаёт образ удалённой файловой системы в базе данных поискового сервера. Из анализа предметной области известно, что количество одновременно работающих поисковых роботов, которые не будут перегружать сервер и его канал в локальную сеть, порядка 10. Это число мы и будем использовать далее в наших оценках времени обновления образа локальной сети, а также занимаемых ресурсов сервера. Для конкретной конфигурации оборудования сервера и сканируемой локальной сети это число может отличаться.

Если имплементировать самый простой алгоритм обхода сети, то получение списка файлов всех папок займёт порядка 8ми часов, а анализ мета-информации всех медиа файлов займёт порядка 80ти часов. Если использовать параллельное сканирование десятью поискового роботами, то индекс поисковой системы будет полностью обновляться каждые 9ть часов и пользователь будет замечать все изменения в локальной сети в среднем через 4,5ой часа. Это куда больше ожидаемых пользователем минут. Более того, никакие изменения в оборудовании сервера и/или кластеризации работы не принесут существенных плодов: для того, чтобы сократить время работы придётся создать целую ферму серверов, которые надо будет разместить в различных частях сети, а также скоординировать их работу, чтобы снизить время обновления индекса до приемлемых минут. Но даже это сможет помочь не во всех случаях, к примеру файловые сервера локальной сети могут вмещать более одного процента сети, поэтому учитывая то, что доступ к одному файловому серверу нельзя осуществлять одновременно от разных

поисковых серверов, чтобы не создавать избыточной нагрузки на файловый сервер, то обновление некоторых частей сети будет всё так же занимать часы.

Однако можно заметить, что большая часть работы производится поискового роботами вхолостую, так как сеть в целом изменяется медленно, её изменения за сутки обычно не превышает 1%, и это даёт возможность создания алгоритма, которые мог бы проверять только те части сети, которые действительно изменились, что дало бы возможность сделать индекс поисковой системы намного релевантнее. Если проводить аналогию с увеличением количества поисковых роботов, то с точки зрения пользователя ситуация стала бы похожей на ту, что достигалась бы 1000 поисковых роботов, вместо 10. Или даже точнее 10, но которые бы работали в 100 раз быстрее.

Создание подобного алгоритма и является предметом данной работы.

3. Математическая модель и алгоритмы индексации

Для того, чтобы получить возможность повторно проверять только те участки сети, которые уже изменились, разобьём сеть на отдельные участки, которые мы назвали *B-поддеревьями*.

3.1 Свойства B-поддерева

Определим набор свойств, которые мы ожидаем увидеть у такого разбиения:

1) *Свойство временной вычислимости.* Количество B-поддеревьев N разбиения сети такое, что время O(N*logN) мало.
   Следует из того, что для того, чтобы мы могли осуществлять комплексный анализ участков сети необходимо, чтобы они были подвержены хотя бы сортировке.
2) *Свойство атомарности сканирования.* Время проверки B-поддерева поисковым роботом много меньше времени среднего нахождения удалённого компьютера онлайн.
   Следует из того, что поисковым робот должен считывать изменения B-поддерева атомарно, без включения и выключения удалённого компьютера в процессе.
3) *Свойство релевантности сканирования.* Время проверки B-поддерева поисковым роботом много меньше ожидаемого времени релевантности поиского индекса и файловой системы удалённого компьютера.
   Следует из того, что найденые поисковым роботом изменения в B-поддереве не должны устаревать ещё до занесения их в базу данных

поиского сервера.

4) *Свойство временной целесообразности.* Время сканирования B-поддерева много больше времени установления соединения до удалённого компьютера.
   Для того, чтобы большую часть времени поисковым робот сканировал B-поддерево, а не ждал установления соединения с удалённым компьютером.

5) *Свойство однородности доступа.* Одно B-поддерево принадлежит одному компьютеру.
   Следует из того, что файлы с похожим временем доступа, частой изменений и типом содержимого находятся чаще всего находятся на одном компьютере.

6) *Свойство импликации папок и файлов.* Для каждой папки и для каждого файла поставлен в соответствие ровно одно B-поддерево.
   Следует из того, что B-поддерево – это разбиение файлов и папок.

Из свойства временной вычислимости следует, что количество B-поддеревьев в разбиении локальной сети не будет сильно превосходить 10 000 *[Кнут]*. Из свойства однородности доступа и того, что в нашей локальной сети 1000 компьютеров следует, что количество B-поддеревьев в разбиении локальной сети будет не менее 1000. При этом по свойству импликации, учитывая то, что количество папок в нашей сети 2 миллиона, файлов 20 миллионов, а медиа файлов 2 миллиона, папок и файлов на одно B-поддерево будет приходиться от 2 до 20 тысяч файлов, от 200 до 2000 папок и от 200 до 2000 медиа файлов. Учитывая то, что получение списков файлов требует порядка 10мс, а чтение деталей медиа файла требует порядка

150мс получим, что сканирование папок займёт от 3 до 30с, а проверка мета-информации медиа файлов займёт от 30с до 5 минут, что, в свою очередь, удовлетворяет свойствам атомарности сканирования, релевантности сканирования и временной целесообразности.

Оговорка корневого B-поддерева. Следует оговориться, что на каждом компьютере будет существовать не более одного B-поддерева, которая не будет удовлетворять свойству временной целесообразности. Пример: компьютер на котором доступно ноль папок и файлов. Он будет разбит по крайней мере на одной B-поддерево по свойству однородности доступа, что нарушит свойство временной целесообразности. Далее будем считать, что этим свойством обладает корневая B-поддерево, чьё определение мы дадим чуть позже.

Поскольку по свойству однородности доступа границы B-поддерево никогда не пересекают границ удалённого компьютера, то будем далее рассматривать структуру папок одного удалённого компьютера и B-поддерева на которые она будет разбиваться.

3.1.1 Представление в виде B-поддеревьев

Все папки удалённого компьютера представимы в виде дерева, где листьями соотносятся с папками без вложенных подпапок, а все остальные вершины соотносятся папкам, которые содержат в качестве подпапок те папки, которые соответствуют вершинам, которыми оканчиваются их инцидентные рёбра. Далее это дерево будет называться деревом папок и различий между вершинами соответствующими папкам и самими папками удалённого компьютера производиться не будет.

Вполне естественно определить B-поддерево, как некоторое поддерево

дерева папок, которое определяется корневой вершиной поддерева. Теперь рассмотрим свойства корневых вершин поддеревьев B-поддеревьев.

Назовём *корневым B-поддеревом* такое B-поддерево у которого корневая вершина его поддерева совпадает с корнем дерева.

Теорема о корневом B-поддереве. Корневое B-поддерево существует и единственно.

Доказательство. По свойству импликации файлов и папок B-поддерева существует единственное B-поддерево для которой корень дерева является её вершинрй. Поддерево содержащее корневую вершину имеет своей корневой вершиной корень дерева *[Ахо]*. *Что и требовалось доказать*.

Теорема о родителях B-поддерева. Либо B-поддерево является корневым, либо родитель корневого элемента его поддерева принадлежит другому B-поддереву.

Доказательство. Либо корневая вершина поддерева B-поддерева является корневой вершиной дерева папок и тогда B-поддерево является корневым по определению, либо она имеет родителя, который по свойству импликации принадлежит какому-либо B-поддереву. Однако она не может принадлежать тому же самому B-поддереву, поскольку в этом случае выбранная нами вершина поддерева не являлась бы корневой. Поэтому она принадлежит другому B-поддереву. *Что и требовалось доказать*.

Поскольку каждое B-поддерево кроме корневой имеет ровно одно B-поддерево, которое содержит в себе вершину, которая является родительской по отношению к корневой вершине её поддерева, то назовём такое B-поддерево для неё *родителем*, а саму её *потомком*. Отметим, что для корневого B-поддерева нет родительского B-поддерева. Из этих свойств

следует, что B-поддеревья образуют дерево.

3.1.2 Алгоритм разбиения дерева на B-поддеревья

Построим алгоритм, который разбивает дерево на B-поддеревья, на который наложено два дополнительных уловия:

1) Условия хранимости. Все имена и пути медиа файлов, которые находятся в данном B-поддереве хранятся в оперативной памяти до тех пор, пока мы работаем с данным B-поддеревом.
2) Условие ограниченности памяти. Для алгоритма установлено максимальное количество памяти, которое он может потреблять для хранения временных данных, в значение M.

Условие хранимости нам понадобится позднее для уменьшения времени анализа медиа файлов впоследствии.

Определим следующий рекурсивный алгоритм разбиение дерева на B-поддерева:

```
1    def РазбиениеДерева(Дерево):
2      РазбиениеДереваС(Дерево.КорневаяПапка)
3      СоздатьB-поддеревоИзПапки(Дерево.КорневаяПапка)
4
5    def РазбиениеДереваСПапки(Папка):
6      for Подпапка in Папка:
7        (Время,Память) = РазбиениеДереваС(Подпапка)
8        if Время >= МаксВремя or  Память >= МаксПамять:
9          СоздатьB-поддеревазПапки(Подпапка)
10         (Время,Память) = (0,0)
11       СуммарноеВремя += Время
12       СуммарнаяПамять += Память
13     Начало = ТекущееВремя()
14     ПамятьДляМедиа = СохранитьМедиаИз(Папка)
15     return (СуммарноеВремя+ПрошлоВремениС(Начало),
16             СуммарнаяПамять+ПамятьДляМедиа)
```

Данный алгоритм рекурсивно проходит по дереву, начиная с его корня, при этом в каждый момент времени он помнит сколько памяти требуется для

хранения медиа файлов текущего B-поддерево (**СуммарнаяПамять**) и сколько времени (**СуммарноеВремя**) ушло на чтение листингов файлов и сохранение медиа из их в список медиа файлов (**СохранитьМедиаИз**). После того, как какое-либо из ограничений на память и время сканирования будет превышено из текущей папки создаётся новая B-поддерево (**СоздатьB-поддеревоИзПапки**). Поскольку теперь **Подпапка** более не принадлежит текущей B-поддереве, то и времени и памяти при повторном сканировании B-поддереваи на неё потрачено не будет. При этом список запомненых медиа файлов обрабатывается, заносится в базу данных поискового сервера и очищается. При этом все дочерние B-поддерева уже созданы, что следует из рекурсивности функции. В конце алгоритма создаётся корневая B-поддерево. Из рекурсивности следует, что все B-поддерева имеют ровно одного родителя, за исключением корневой B-поддерева. Следовательно мы создали правильное разбиение.

Покажем алгоритм для повторного сканирования B-поддерева на предмет медиа файлов:

```
1  def СканированиеB-поддеревоНаПредметМедиаФайлов(B-поддерево):
2    СканироватьПапку(B-поддерево.КорневаяПапка, B-поддерево)
3
4  def СканироватьПапку(Папка, B-поддерево):
5    if not КорневаяПапкаB-поддерева(Папка, B-поддерево.ДочерниеB-поддерева):
6      for Подпапка in Папка:
7        СканироватьПапку(Подпапка, B-поддерево)
8      СохранитьМедиаИз(Папка)
```

Этот алгоритм при своей работе будет удовлетворять требованиям условий для алгоритма разбиения дерева на B-поддерева по памяти и времени работы, если дерево папок и файлов не изменилось со времени разбиения на B-поддерева.

3.2 Свойства B-поддеревьев*

Однако такое простое определение B-поддеревьев, как поддеревья не всегда будет работать. Представим себе, как нижеследующее дерево надо разбить на B-поддеревья:

Дерево Натали. У корня дерева миллион потомков, каждый из которых хранит один медиа файл. (Дерево названо в честь музыкального сервера **\\Natalie** локальной сети МФТИ, который имеет схожую структуру)

Проблема Натали. У дерева Натали нельзя получить разбиение на B-поддерева, где каждое B-поддерево – поддерево.

Доказательство. Докажем от противного. Предположим, что мы нашли такое разбиение на B-поддерева. В таком дереве находится миллион папок с миллионом медиа файлов. По выводам из свойств B-поддеревьев их не может быть сильно больше, чем 10 000, поэтому из миллиона папок не более 100 000 могут стать B-поддеревьями. Поэтому в корневое B-поддерево попадёт больше 100 000 медиа файлов, его сканирование займёт около часа и это нарушит свойство релевантности сканирования. *Что и требовалось доказать*.

Поэтому расширим определение B-поддерево до *B*-поддерева* следующим образом:

Некорневое B*-поддерево определяется корневой вершиной и двумя границами имён, каждая из которых может быть не задана. Он включает в себя все поддеревья, чьими корневыми вершинами являются подпапки корневой папки, имена которых находятся от первой границы (включительно) до второй (исключительно), где одно или оба этих условия снимаются если соответствующая граница не задана. Некорневое B*-

поддерево не может одновременно иметь корневую вершину совпадающую с корневой вершиной дерева и не иметь ни одной заданой границы.

Корневое B*-поддерево включает в себя корневое поддерево. Фактически оно является некорневым B*-поддеревом с корневой вершиной совпадающей с корневой вершиной дерева, не заданными границами и включает файлы находящиеся в корневой вершине.

Теорема расширения B-поддеревьев B-поддеревьями*. Каждому разбиению на B-поддеревья можно сопоставить в соответствие разбиение на B*-поддеревья, которое будет обладать тем же набором вершин для каждого элемента разбиения.

Доказательство. Поставим каждому корневому B-поддереву корневое B*-поддерево. Каждому некорневому B-поддереву B с именем корневой папки F, сопоставим B*-поддерево B*. Корень B* будет родителем F, который будет существовать так как B не корневое, левая граница будет F, если существует папка V следующая по алфавиту за папокй F, то правая граница будет V, если нет такой папки, то правая граница будет неопределена. Согласно определению B* папка F и её подпапки будут принадлежать B*. Так как границы B* определяют только одну папку F, которая попадает между ними, то никакие папки не принадлежащие B не попадут в B*. Итак, для любой папки принадлежащей B, эта папка будет принадлежать и B*, а для любой не принадлежащей не будет. *Что и требовалось доказать*.

Новое определение куда более гибкое и, в частности, позволяет разрешить проблему Натали. К примеру, можно задать B*-поддеревья так, чтобы каждые 1000 папок попали в своё B*-поддерево. Это разобьёт дерево

на 1001 B*-поддерево, каждое из которых, кроме корневого, будет содержать 1000 медиа файлов и папок. Такое разбиение не будет нарушать ни одно свойство B-поддерева, если учитывать оговорку корневого B-поддерева.

3.2.1 Алгоритм разбивания дерева на B-поддеревья*

Определим следующий рекурсивный алгоритм разбиение дерева на B-поддерева:

```
1  def РазбиениеДерева(Дерево):
2    РазбиениеДереваС(Дерево.КорневаяПапка)
3    СоздатьB*-поддеревоИз(Дерево.КорневаяПапка, null, null)
4
5  def РазбиениеДереваС(Папка):
6    ЛеваяГраница = null
7    for Подпапка in Папка:
8      (Время,Память) = РазбиениеДереваС (Подпапка)
9      СуммарноеВремя += Время
10     СуммарнаяПамять += Память
11     if СуммарноеВремя >= МаксВремя or СуммарнаяПамять >= МаксПамять:
12       СоздатьB*-поддеревоИз(Папка, ЛеваяГраница,
13           СледующаяПапка(Подпапка))
14       ЛеваяГраница = СледующаяПапка(Подпапка)
15       (СуммарноеВремя,СуммарнаяПамять) = (0,0)
16   Начало = ТекущееВремя()
17   ПамятьДляМедиа = СохранитьМедиаИз(Папка)
18   return (СуммарноеВремя+ПрошлоВременисС(Начало),
19           СуммарнаяПамять+ПамятьДляМедиа)
```

В отличии от предыдущего алгоритма создаваемое B*-поддерево требует уже три параметра: папку, для потомков которой будут указаны диапазоны и два диапазона, левый и правый. При этом каждый из диапазонов может принимать особое значение **`null`**, которое отвечает за тот случай, когда граница не задана. Также появилась функция **`СледующаяПапка`**, которая для папки F у который два потомка A и B для потомка A возвращает B, а для B возвращает **`null`**. Таким образом она возвращает следующуюв лексикографическом порядке папку, которая находится на том же уровне вложенности. При этом корневое B*-поддерево создаётся всегда с корневой папкой и отсутствием обеих границ.

В процессе сканирования **Папки** ведётся учёт памяти и времени необходимого для сканирования всех папок, кроме тех, которые уже стали членами дочерних B*-поддеревьев. При этом после создания каждого нового B*-поддерева мы запоминаем его правую границу, как левую границу для возможно создаваемого следующего B*-поддерева для той же **Папки**. Таким образом для всех B*-поддеревьев у которых общая корневая папка не пересекающиеся границы, если учитывать то, что левая граница включающая, а правая исключающая. Из этого следует то, что алгоритм не создаст пересекающихся B*-поддеревьев. Поэтому каждая папка и каждый файл будут принадлежать ровно одному B*-поддереву, поэтому свойство импликации файлов и папок не будет нарушено.

Данный алгоритм создаёт более сложное разбиение дерева и позволяет работать правильно в куда более большом количестве случаев, к примеру, он решает проблему Натали. Однако существуют деревья в которых и этот алгоритм будет превышать требуемое количество памяти:

Правовзвешенное дерево. Каждый узел Vi содержит папки Fi, которая содержит 100 медиа файлов и имеет имя «A», и Vi+1, которая имеет имя «B». При этом i находится в диапазоне от 1 до 100. V101 не содержит файлов и папок.

При анализе этого дерева для разбиения на B*-поддеревья при анализе Vi папка Fi не будет содержать достаточно медиа файлов, чтобы образовать отдельное B*-поддерево, поэтому будет продолжен рекурсивный анализ папки Vi+1. Но все медиа файлы папки Fi по свойству хранимости будут сохранены в память, поэтому в момент анализа V100 в памяти будут все медиа файлы папок Fi, где i от 1 до 100, что даст нам 10 000 медиа файлов

находящихся в памяти. Надо заметить, что это число не зависит от ограничений на память и время для одного B*-поддерева, поэтому при некоторых наборах параметров алгоритм будет потреблять больше оперативной памяти, чем было указано в **МаксПамяти**.

Однако при исследовании реальных локальных сетей на построенном прототипе конфигурации деревьев, которые бы требовали заметно больше памяти для анализа, чем было указано константой **МаксПамяти**, не встретились.

3.3 Переразбиение дерева

Мы рассмотрели случай разбиения дерева на B*-поддеревья, но после того, как дерево разбито на B*-поддеревья сеть изменится и надо будет делать новое разбиение на B*-поддеревья. Наложим следующие условия на алгоритм переразбиения на B*-поддеревья:

1) Количество изменений в дереве B*-поддеревьев линейно зависит от количества изменений.

2) Если со времени предыдущего разбиения было добавлено или удалено несколько медиа файлов, то это не должно вызывать переразбиения на другие B*-поддеревья

Если алгоритм переразбиения при изменениях дерева свести к операции разбиния одного B*-поддерева на несколько и операции объединения нескольких B*-поддеревьев в одно, и минимизировать количество операций, то оба условия будут выполнены. Особо надо выделить операцию объединения родительского B*-поддерева и его потомка, при этом операция их объединения будет состоять в удалении дочернего B*-

поддерева, при этом дерево B*-поддеревьев будет оставаться деревом, а все медиа файлы, которые принадлежали ранее потомку станут принадлежать родителю.

Алгоритм будет следующим:

1) Если корневую папку B*-поддерева удалили, то B*-поддерево надо удалить.
2) Если размер B*-поддерева оказался менее половины максимального ограничения размера для первичного разбиения, то его нужно объединить с родительским B*-поддеревом.
3) Если размер B*-поддерева оказался более двух размеров максимального ограничения размера для первичного разбиения, то его нужно заново переразбить на два B*-поддерева.

Теперь размер любого B*-поддерева после выполнения этих двух операций будет в пределах от половины до двух максимальных ограничений размеров для ервичного разбиения.

Полученный алгоритм не будет выполнять операции переразбиения поддерева при малых изменениях сети, а при больших изменениях количество операций изменения дерева B*-поддеревьев будет пропорционально количеству изменения в локальной сети, так как для каждого изменения требуется создать или удалить медиа файлов не менее половины максимального ограничения размера для первичного разбиения.

3.4 Анализ сканирования неизменных частей сети

Однако чаще всего сеть не изменяется, поэтому при проверке на изменения отдельного B*-поддерева изменений не будет найдено. При этом, если считывать из базы данных все медиа файлы, которые принадлежат данному B*-поддереву, то получается неоправдано большое количество чтений из базы данных, когда всё, что требуется – проверить произошло ли измнений B*-поддерева.

Поскольку сканирование одной папки требует в 10 раз меньше времени, чем сканирование одного медиа файла, то сканирование только папок и файлов игнорирующее медиа файлы работает в 10 раз быстрее, чем полное сканирование всех данных. Построим алгоритм так, чтобы быстрое сканирование папок не требовало слишком много операций с базой данных.

3.4.1 Хэширование B-поддеревьев*

Для этого введём понятие хэша B*-поддерева. Определим его так, чтобы при любом изменении папки или файла внутри этого B*-поддерева хэш менялся. При этом его можно рассчитать при сканировании только файлов и папок. Поскольку хэш учитывает только информацию собираемую при сканировании папок и не учитывает содержимое медиа файлов, то в него попадут имена папок и файлов, размеры файлов и даты изменения папок и файлов. Поскольку в хэш попадают только те файлы и папки, которые входят в B*-поддерево, а разбиение на поддеревья проходит рекурсивно, то хэш тоже должен быть рассчитываемым рекурсивно. Введём H – фукнцию хэша и определим её следующим образом:

H(папка без подпапок) = Hash(имя папки, имена файлов, даты изменения)

H(папка) = Hash(H(подпапка1), H(подпапка2), … , имена файлов, даты)

Где Hash – хеширующая функция, которая на входе принимает массив из массивов бит произвольной длинны, а на выходе даёт последовательность бит длинны 2N. От неё требуется, чтобы вероятность того, что для разных данных будет получена один и тот же хэш была достаточно мала. Для того, чтобы хэши для всех папок были различны и не повторялись при изменениях папок требуется, чтобы 2^N было много больше количества банчей которые могут образоваться за большое количество времени, скажем за год, поэтому требуется чтобы 2^N было много больше чем 10 000 * 1000 000, или N>>33. Поэтому для функции Hash подходит широко применимая функция md5, которая обладат требуемыми свойствами и которая даёт хэш длинной 128 бит.

3.5 Оптимизация релевантности

После того как мы описали наши алгоритмы разбиения сети на B*-поддеревья и обсуждили детали того как именно работает сканирование отдельного B*-поддерева осталось только выбрать правильное B*-поддерево, чтобы её можно было начать сканировать. У нас есть B*-поддеревья, более того, про каждое у нас есть собранная статистическая информация и мы знаем когда в прошлом B*-поддерево менялось, а когда не менялось.

Наше предположение состоит в том, что на основании данных о прошлых сканированиях B*-поддерева можно получить информацию о том, как оно будет меняться в будущем.

Примеры:

Папка входящих файлов файлового сервера. В прошлом она менялась

каждые несколько минут на протяжении месяца, можно ожидать, что она и следующий день будет меняться каждые несколько минут.

Папка с хорошими фильмами. Меняется нерегулярно, но в среднем каждые несколько дней. Вполне можно ожидать, что она за неделю уж точно поменяется, а за день вряд ли.

3.5.1 Ожидаемая вероятность изменения

Представим, что у нас есть две B*-поддерева и одна уже изменилась со времени последнего сканирования с вероятностью 80%, хотя прошло всего 5 минут, а другая изменилась с вероятностью 40%, хотя мы сканировали её день назад. Оценки процентов мы сделали изучая историю изменений B*-поддерева за длительный период. Раз какой-то кусочек сети уже почти гарантированно изменился, то его и надо проверять на изменения. Поэтому если надо надо запустить очередное сканирование B*-поддерева, то на изменения должна быть проверена первая.

В целом принцип выбора следующей B*-поддерева которую мы начнём сканировать – надо взять B*-поддерево у которой на текущий момент вероятность того, что она изменилась наибольшая.

Примем модель обычного пользователя, когда он не регулярно выкладывает что-то в общий доступ, а время от времени и как получится. При этом есть некоторая общая активность пользователя, которая вариируется от пользователя к пользователю.

Такой случайный процесс обновления файлов в общем доступе хорошо моделируется марковскими процессами, в частности нам хорошо подходит Пуассоновский процесс, который моделирует наступление дискретных

событий.

$$P[X(t)=k]=\frac{\lambda^k t^k}{k!}e^{-\lambda t},$$

где P вероятность того, что через время t B*-поддерево успела поменяться k раз. Поскольку нам интересно изменилась ли B*-поддерево за время t с момента последней проверки, то вероятность того, что данная B*-поддерево поменялась за время t равняется

$$P[X(t)>0]=1-e^{-\lambda t}.$$

Где l определяется из истории изменеий B*-поддерева, как мат. ожидание времени проходящего между событиями и определяется из количества изменений B*-поддерева за характерный период. Характерный период определяется эмпирически, как время за которое привычки пользователя могут измениться, мы оцениваем его в месяц.

Итак, для каждой B*-поддерева мы храним среднее время проходящее между её изменениями l, время последней проверки B*-поддерева (не важно нашедшее или не нашедшее изменения) t0 и можем оценивать вероятность того, что B*-поддерево уже изменилась как

$$P=1-e^{-\lambda(t-t_0)},$$

где t – текущее время. Рассчитав вероятности для всех B*-поддеревьев и выбрав ту, у которой эта вероятность наибольшая мы начинаем её сканирование.

Таким образом выбор следующей B*-поддерева для сканирования Bn из набора B*-поддеревьев B происходит так:

$$B_n=\operatorname{argmax}_{b\in B}\left(1-e^{-b\lambda(t-b_t)}\right),$$

где bλ – лямбда параметр B*-поддерево b, а bt – время предыдущего скана B*-поддерева.

3.5.2 Оценка общих потерь

Альтенативным методом оценки выбора следующей B*-поддерева для сканирования является метод оценки потерь от сканирования. Будем считать суммарные потери поискового движка от того, что он начнёт сканировать очедерную B*-поддерево и потери пользователя от того, что сканирование B*-поддерева будет позже её изменения и поэтому в индексе будут нерелеватные результаты. Поисковому движку всё равно какую B*-поддерево сейчас сканировать и его потери мало зависят от того изменилась ли она, а пользователь малочувствителен к малым задержкам индекса по отношению к файловой системе и сильно чувствителен к большим временам, более того, существует некоторое критическое время T после которого пользователь начинает очень быстро становится недовольным задержкой. Для простоты модели положим, что потери от сканирования B*-поддерева до его изменения s, после изменения d, а потери пользователя при сканировании B*-поддерева после её момента изменения t0 экспоненциально возрастают от времени. Итак, суммарные потери от сканирования B*-поддерево в момент времени t, когда он изменился в момент времени t0 это

$$R(t)=\begin{cases} d+e^{\frac{t-t_0}{T}}-\frac{t-t_0}{T}-1 & t>t_0 \\ s & t\le t_0 \end{cases},$$

где функция R(t) обладает требуемым характером:

$$R(t)=s,\ при\ t<t_0,\ \dot{R}(t)=0,\ при\ t=t_0\ и\ R(t)>>1+s,\ при\ t>>t_0.$$

Пускай мы проверили B*-поддерево последний раз в момент времени t0,

пускай вероятность того, что B*-поддерево изменилась определяется пуассоновским процессом P[X(t)>0]=1-exp(-lt). Известно есть распределение плотности вероятности изменения B*-поддерева в каждый момент времени, также известно значение функции потерь для любого момента t' изменения B*-поддерева, поэтому потери в момент времени t, если предыдущая проверка была в момент времени t0 представляют собой случайную величину. Посчитаем математическое ожидание этой величины:

$$\mathrm{E}R_{t_0}(t)=\int_{t_0}^{+\infty} P[X(t-t_0)>0]R(t-t_0)dt$$

.

Теперь мы для каждой B*-поддерева знаем величину её потерь от сканирования в любой момент времени. Причём поскольку

$$\mathrm{E}R_{t0}(t_1)+\mathrm{E}R_{t1}(t_2)<\mathrm{E}R_{t0}(t_2),\ при\ t_0<t_1<t_2,$$

то всегда выгодно проверить какую-то B*-поддерево теперь, а не ждать у моря погоды. Поэтому наиболее выгодно проверять сейчас B*-поддерево с наибольшим ER, чтобы он не стал ещё больше. Поэтому задача выбора B*-поддерева для повторного сканирования становится следующей:

$$B_n=\mathrm{argmax}_{b\in B}\mathrm{E}R(t,b_\lambda,b_{t0}),$$

где B – множество B*-поддеревьев, Bn – следующая B*-поддерево для сканирования, t – текущее время, bl –лямбда параметр B*-поддерева b, bt0 – время последнего сканирования B*-поддерева b. Таким образом получаем следующую формулу для выбора B*-поддерево для последующего сканирования:

$$B_n=\mathrm{argmax}_{b\in B}\left(\int_{b_{t0}}^{t} b_\lambda e^{-b\lambda t'}\left(d+e^{\frac{t'-b_{t0}}{T}}-\frac{t'-b_{t0}}{T}-1\right)dt'+\int_{t}^{+\infty} b_\lambda e^{-b\lambda t'}sdt'\right)$$

4. Программная реализация

На основе рассмотренной архитектуры индекса и поисковых алгоритмов был разработан прототип поисковой системы. Система получила название «Paddle».

Рис. 3. Главная страница поисковой системы «Paddle».

Запуски и испытания поисковой системы проводились на подсети локальной сети МФТИ на довольно устаревшем персональном компьютере:

Pentium 800, 512 Mb PC133, 40Гб жесткий диск.

Индексирующим механизмом было проиндексировано достаточно большое количество медиаресурсов, после чего были проверены поисковые механизмы, иследована структура собранного индекса и найдены закономерности.

4.1 Опытные храктеристики разбиения

Из 256 мегабайт памяти 100 было отдано для памяти 10 поисковых роботов, поэтому на каждого из них приходилось ограничение в 10 мегабайт. Временным ограничением было выбрано 90 секунд на сканирование B*-поддиапазона.

После запуска тестового поискового сервера характеристики разбиений на B*-поддеревья оказались в среднем одинакового размера от 1000 до 100 000 файлов, и содержат не более 800 медиа файлов. Надо заметить, что скорость сканирования различных компьютеров в сети отличичалась в десятки раз, поэтому размер B*-поддеревьев, когда они ограничивались временем повторного сканирования, а не объёмом оперативной памяти, был различен. Порядка 30% B*-поддеревьев были ограничены памятью, а порядка 60% были ограничены временем повторного сканирования, примерно 10% B*-поддеревьев получились существенно меньше ограничений и памяти и времени.

4.2 Опытные характеристики ускорения из-за хэширования B-поддеревьев*

Скорость сканирования файловой системы без изучения мета информации медиа файлов была в 5-20 раз быстрее, чем полное изучение B*-поддеревьев. После начального сканирования сети в установившемся режиме примерно каждое десятое сканирование обнаруживало какое-либо изменение файловой системы. Поэтому каждое отдельное сканирование стало в 3-7 раз быстрее.

Поскольку если не было найдено изменений файловой системы, то производилось только одно чтение из базы данных – хэша B*-поддерева, в

случае нахождения изменений происходит чтение медиа файлов из B*-поддерева, а количество записей пропорционально количеству изменений в медиа файлах.

В среднем на одно B*-поддерво приходится 100 медиа файлов, а за одно изменений находится 5 изменённых медиа файлов, поэтому количество операций с базой данных на одно сканирование B*-поддерева. Если не было изменений, то 1 операция чтения, если было n изменений и было b медиа файлов, то 1+b чтений и n записей. С учётом полученных данных в среднем получается: 11 операций чтения и 0.5 операций записи на одно сканирование B*-поддерева.

Если не было изменений сканирование в среднем занимает 30 секунд, а если изменения были, то порядка 5 минут, поэтому получаем среднее время сканирования одна минута.

Итак, в среднем на каждого поискового робота требуется одна операция чтения в 5 секунд и одна опреация записи в две минуты. Это делает операции сканирования сети не ограниченными от скорости жёсткого диска.

Вся дополнительная информация для хранения B*-поддеревьев потребовала порядка двух мегабайт данных.

Заключение

Цель данной работы была определена так: необходимо разработать алгоритмы и построить робота поисковой системы мультимедийной информации для локальных сетей.

В ходе выполнения этой задачи была подробно изучена предметная область: определены особенности объектов поиска – медиа-ресурсов; описана структура локальной сети, как источника мультимедийной информации; сформулированы потребности пользователя при работе с поисковой системой и выбран наиболее удобный и привычный интерфейс взаимодействия – текстовый поиск по словам; проанализированы существующие поисковые системы для локальных сетей и интернет, выделены их сильные и слабые позиции; описана технологическая среда, в которой будет необходимо создавать поисковую систему, отмечены самые важные накладываемые компьютерной средой ограничения; переформулирована задача с учетом контекста предметной области.

Далее была построена математическая модель поисковой инфраструктуры: были формализованы и описаны операции доступа к локальной сети, выбраны структуры данных и алгоритмы, позволяющие оптимизировать критический для поискового индекса фактор – релевантность, математически описаны все структуры данных и объекты, с которыми будет работать поисковый робот, описан механизм позволяющий численно измерять состояние поискового индекса.

При помощи построенной математической модели был разработан, и формально описан алгоритм, который принимает на входе представление локальной сети в виде дерева папок и последовательными шагами при

помощи особого представления данных разбивает локальную сеть на отдельные подобласти, которые проверяются на предмет изменений, обновляя индекс. Был теоретически обоснован тот факт, что алгоритм удовлетворяет требованиям пространственной и временной эффективности. Построено несколько моделей изменения данных в сети и необходимой реакции на них поискового робота, созданы метрики оценки меры соответствия поискового индекса и файловой сети.

Был запрограммирован действующий прототип, на практике проверена его работоспособность, поведен анализ собранной в индексе структуры данных и установлены некоторые эмпирические закономерности.

Безусловно, на данном этапе работа над поисковой системой и поисковым роботом в частности далека от завершения. Самыми перспективными направлениями для продлжения работы являются:

- построение архитектуры, масштабируемой на серверный кластер;
- создание «толстого» клиентского приложения;
- присоединение внешних баз знаний, содержащих дополнительную метаинформацию о медиа-ресурсах;
- внедрение системы ранжирования на основе PageRank.

Библиография

1. *[Кормен] Т. Кормен, Ч. Лейзерсон, Р. Ривест*, Алгоритмы: построение и анализ // ISBN: 5-900916-37-5
2. *[Ахо] А. Ахо, Дж. Хопкрофт, Дж. Ульман*, Построение и анализ вычислительных алгоритмов. // ISBN 5-8459-0857-4, 0-07-013151-1
3. *[Кнут] Д. Кнут*, Искусство программирования // ISBN: 5-8459-0080-8, 0-201-89683-4
4. *[Чернецов] М. Чернецов*, Разработка алгоритмов и построение индекса поисковой системы мультимедийной информации для локальных сетей
5. *[Экштайн] R. Eckstein, D. Collier-Brown*, Using Samba // ISBN: 0-596-00256-4
6. *[Хертель] C. Hertel*, Implementing CIFS - The Common Internet FileSystem // ISBN: 0-13-047116-X
7. *[RFC959] J. Postel*, File Transfer Protocol (FTP) // RFC 959
8. *[RFC792] J. Postel*, Internet Control Message Protocol (ICMP) // RFC 792
9. *[Натан] А.А. Натан, О.Г. Горбачев, С.А.Гуз*, Случайные процессы. Математическая статистика. // ISBN: 978-5-94073-102-3
10. *[Бирюков] С.И. Бирюков*, Оптимизация. // М.: МФТИ, 1995.
11. *[Сегалович] И.В. Сегалович*, Как работают поисковые системы // http://company.yandex.ru/articles/article10.html
12. *[Зеленков] И.В. Сегалович, Ю.Г. Зеленков, Д.О. Нагорнов*, Методы сравнительного анализа современных поисковых систем и определения объема Рунета // Труды 8ой Всероссийской научной конференции

«Электронные библиотеки: перспективные методы и технологии, электронные коллекции» - RCDL’2006, Суздаль, Россия, 2006.

13. *[Брин] S. Brin, L. Page*, The Anatomy of a Large-Scale Hypertextual Web Search Engine // WWW7, 1998

14. *[Лианг] S. Liang*, Java Native Interface: Programmer's Guide and Specification (JNI) // http://java.sun.com/docs/books/jni/

15. *[Бишоп] T. Bishop, G. Mitchell, J. Bell, B. Holm, D. Ayers, C.C. Bettis, S.R.T. Loton, M. Bogovich, M. Wilcox, L.K. Poon, N. Nanda, R. Grehan, M. Ferris, K.L. Poon*, Professional Java Data: RDBMS, JDBC, SQLJ, OODBMS, JNDI, LDAP, Servlets, JSP, WAP, XML, EJBs, CMP2.0, JDO, Transactions, Performance, Scalability, Object and Data Modeling // ISBN-10: 1861004109, ISBN-13: 978-1861004109

16. *[Журавлёв] Ю.И. Журавлев, Ю.А. Флеров*, Дискретный анализ. Ч.1, Учебное пособие. // М.: Изд-во МФТИ, 1999

17. *[Серебряков] В.А. Серебряков, М.П. Галочкин, Д.Р. Гончар, М.Г. Фуругян*, Теория и реализация языков программирования. // М.: МЗ-Пресс, 2003

18. *[Ламль] T. Lammle*, CCNA: Cisco Certified Network Associate Study Guide, Third Edition // ISBN-10: 0782141676ISBN-13: 978-0782141672

19. *[Шварц] B. Schwartz, P. Zaitsev, V. Tkachenko, J. Zawodny, A. Lentz, D. Balling*, High Performance MySQL: Optimization, Backups, Replication, and More // ISBN-10: 0596101716, ISBN-13: 978-0596101718