УДК 001.57:519.876.5:004.942

ВЫБОР ОПТИМАЛЬНОГО ПУТИ ВОССТАНОВЛЕНИЯ В СИСТЕМАХ РЕЗЕРВНОГО КОПИРОВАНИЯ

В.Г. Казаков, С.А. Федосин

Аннотация. *В работе рассмотрена проблема построения универсального алгоритма поиска пути восстановления в современных системах резервного копирования данных. Исследована возможность применения аппарата теории графов. Рассмотрены различные алгоритмы теории графов и обоснован их выбор для исследуемой области. Представлены рекомендации по реализации предлагаемого метода.*

Ключевые слова. *Резервное копирование, восстановление данных, алгоритм, граф, нахождение кратчайших путей.*

Введение

Нетрудно понять, что для традиционных схем резервного копирования, вопрос выбора оптимального пути не стоит в принципе. Так, при работе, к примеру, инкрементной схемы создается единственная цепочка элементов репозитория, и при восстановлении к любой точке существует лишь единственный путь. Составление пути восстановления элементарно. Путь восстановления для полного резервирования редуцируется в единственный элемент репозитория – полную копию данных на момент времени восстановления.

Несмотря на то, что традиционные подходы применяются на данный момент повсеместно, они представляют крайности в выборе соотношений между основными характеристиками процессов резервного копирования: временем создания копий, временем восстановления данных и объемом репозитория для хранения копий. Это является поводом заставляющим сомневаться в эффективности применения традиционных подходов резервного копирования, и искать другие алгоритмические подходы. В то же время, вместе с быстрым ростом объемов хранимых данных все больше возрастает сложность их защиты, используя традиционные алгоритмы резервного копирования. О недостаточной эффективности свидетельствуют также данные исследований [1, 2, 3, 4].

Между крайностями традиционного полного резервного копирования, требующего неприемлемо большие объемы памяти для хранения, и инкрементного, с крайне низкой эффективностью восстановления существуют другие алгоритмы позволяющие найти нужный баланс между основными характеристиками процессов резервного копирования. См. [2, 5, 6].

При работе более сложных алгоритмов создания резервных копий, появляется потребность в нахождении оптимального пути восстановления из возможных. Для этого необходимо исследовать проблему оптимальности пути восстановления и способ эффективного их построения.

Постановка задачи

Рассмотрим некоторую систему резервного копирования, которая создает копии данных в последовательные запланированные моменты времени *{t_k, где $k=0,1,2,\ldots,T$}*. Систему данных для резервного копирования на момент времени t_j обозначим как D_j. Будем отныне всегда предполагать, что начальное состояние данных D_0 – пустое, а D_1 нет, то есть $D_0=\emptyset$, $D_1\neq\emptyset$. Каждая сделанная копия данных сохраняется в некотором хранилище данных – *репозитории*. Каждую такую копию будем называть *элементом репозитория*. Элемент репозитория, который содержит изменения между состояниями данных с момента t_l до момента t_{l+n}, то есть от D_l до D_{l+n} обозначим как R_l^n или $R(l,n)$, где l и n – целые, причем $n>0$, $l\geq 0$. Учитывая, что начальное состояние данных пустое, R_0^n фактически означает полную резервную копию состояния данных в момент t_n, то есть копию D_n.

Имеет смысл операция объединения элементов репозитория. Например, объединяя полную резервную копию системы данных на момент t_l, то есть R_0^l, с элементом репозитория содержащим изменения между состояниями данных с момента времени t_l до момента t_{l+n}, то есть с R_l^n, мы получаем полную резервную копию системы данных на момент t_{l+n}, то есть R_0^{l+n}.Следует заметить, что не все элементы могут быть объединены друг с другом, нет смысла, например, объединять элементы, содержащие изменения данных в не последовательные периоды времени. Для объединения не сочетаемых элементов репозитория введем элемент X, который будет обозначать некий “испорченный” элемент.

Множество всех элементов репозитория обозначим ***Rep***. ***Rep*** включает в себя все возможные элементы репозитория и введенный “испорченный” элемент X.

$$\boldsymbol{Rep} = \{X; R_i^j\}_{i=0,1,2\ldots;j=1,2,3\ldots} = = \{X; R_0^1; R_0^2; R_0^3;\ldots; R_1^1; R_1^2; R_1^3;\ldots; R_k^1; R_k^2; R_k^3;\ldots\}.$$

Операцию объединения элементов репозитория $\oplus$ на множестве всех элементов репозитория ***Rep*** введем следующим образом:

1. $X \oplus X = X;\ \ R_l^n \oplus X = X \oplus R_l^n = X;$

2. $$R_i^{n_i} \oplus R_j^{n_j} = \begin{cases} R_i^{n_i+n_j}, \text{если } i<j \ \ и \ \ j=i+n_i; \\ X, \text{в противном случае.} \end{cases}$$

Заметим, что введенный способ объединять элементы репозитория является алгебраической бинарной ассоциативной операцией, а, множество всех элементов репозитория с операцией объединения элементов репозитория, то есть ***Rep***$\oplus$ является полугруппой.

Выбор пути восстановления к некоторому моменту времени t_j для некоторого алгоритма резервного копирования означает выбор последовательности сочетаемых элементов из репозитория системы резервного копирования, объединение которых в любой последовательности, но в порядке следования, приводит к образованию копии данных на требуемый момент времени t_j.

Требуется исследовать проблему выбора оптимального пути восстановления для различных схем резервного копирования, подобрать алгоритм поиска таких путей максимально универсальный и в тоже время достаточно эффективный в смысле вычислительной сложности.

Представление работы схем резервного копирования в виде графов

Работу схем резервного копирования удобно представлять в виде ориентированных мультиграфов. При этом отображая в виде вершин состояния данных, а в виде дуг – элементы репозитория, хранящие данные об изменении между двумя состояниями данных. При этом удобно присваивать номера вершинам последовательно в соответствии с моментами резервного копирования t_k. Так, началом дуги, изображающей элемент репозитория $R(l,m)$ будет являться вершина, соответствующая D_l, а концом, соответствующая D_{l+m}.

Понятие пути восстановления в такой интерпретации становиться наглядным и совпадает с понятием пути в орграфах. При этом становится возможным применение аппарата теории графов для отыскания кратчайших путей, и путей соответствующих некоторым наперед заданным условиям.

Для примера проиллюстрируем работу инкрементного резервного копирования, используя представление в виде орграфа. См. рис. 1.

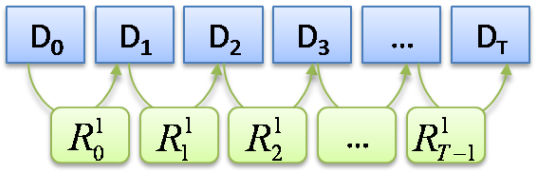

Рис. 1. Инкрементное резервное копирование

Полное резервное копирование в таком случае будет выглядеть так, как показано на рис. 2. При этом важно вспомнить требование пустоты начального состояния данных.

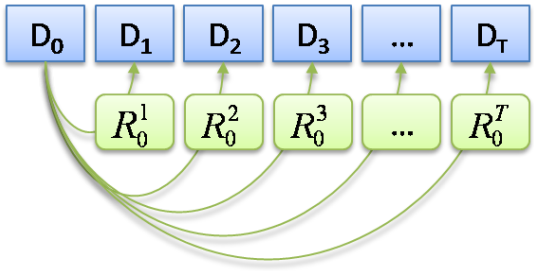

Рис. 2. Полное резервное копирование

Проиллюстрируем также проблему объединения не сочетаемых элементов репозитория простым примером. Начальное состояние данных D_0 пустое. Пусть D_1 включает в себя только некоторый файл A. Далее к моменту t_2 создаются файлы B и C, а к моменту t_3 остается только C, то есть $D_3=\{C\}$. Предположим, что были созданы элементы репозитория, изображенные на рисунке, а именно $R(0,1)$, $R(1,2)$ и $R(0,2)$. Причем, A^- в элементе $R(1,2)$ означает данные об удалении A. Рассмотрим объединение не сочетаемых элементов по данному выше определению: $R(0,2)$ и $R(1,2)$. Их объединение $R(0,2) \oplus R(1,2)$ не будет равносильно элементу $R(0,3)$, ибо хоть в них и содержится необходимый для $R(0,3)$ файл C, но $R(1,2)$ не содержит необходимой для верного результата информации об удалении B.

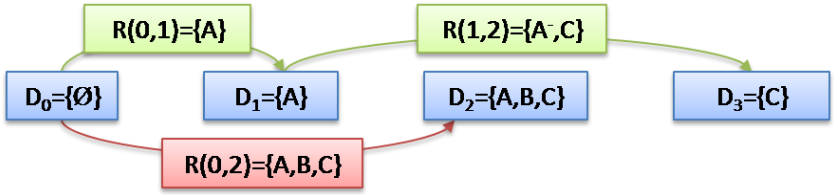

Рис. 3. Пример объединения не сочетаемых элементов репозитория

Мультиуровневое резервное копирование

Как первый пример нетривиальной схемы резервирования рассмотрим мультиуровневое резервное копирование (англ. *multi-level backup*).

Кратко опишем процедуру работы такого алгоритма. Копирование ведется в несколько уровней: на 0-ом уровне создаются полные резервные копии; на последующих – копируются файлы, модифицированные с момента предыдущего резервного копирования более низкого уровня.

На рис. 4 представлена иллюстрация работы некоторого мультиуровневого алгоритма за первые 12 периодов резервирования.

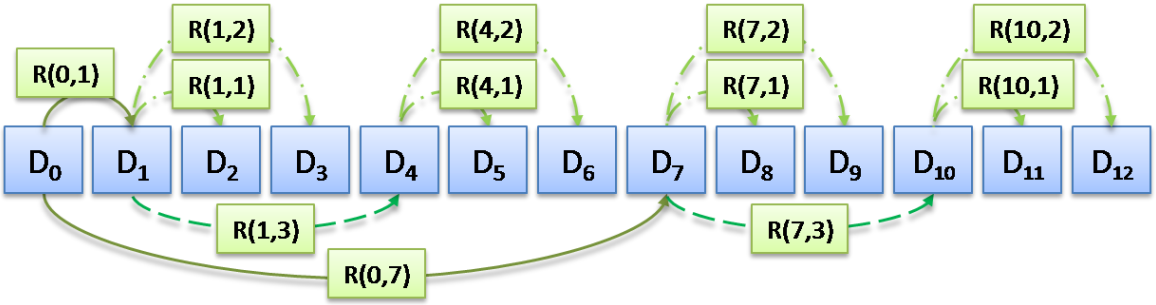

Рис. 4. Мультиуровневое резервное копирование

Для мультиуровневого резервного копирования при восстановлении на каждый момент времени также существует лишь единственный путь. Однако процедура составления такого пути несколько более сложна.

Схемы с явным дублированием данных

Есть возможность сократить количество элементов в пути восстановления, а тем самым и время восстановления, создавая на этапе создания резервных копий дополнительные избыточные элементы репозитория. Например, рассмотрим способ создания избыточных элементов так, как показано на рис. 5. Так используя избыточные элементы *R(0,2)* и *R(2,2)*, можно создать вдвое более короткий путь от D_0 к D_4.

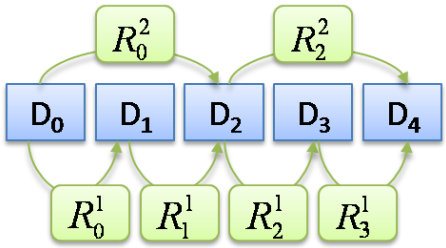

Рис. 5. Пример дублирования

Метод введения избыточных элементов может быть использован для значительного повышения эффективности процедуры восстановления. Один из таких алгоритмических подходов описан А.М. Костелло, К. Юмансом, Ф. Ву [5].

При этом, создание избыточного элемента может происходить как с непосредственным дублированием самих данных, так и без него. В случае, когда данные не копируются, избыточный элемент носит характер индекса со ссылками на данные в агрегированные элементы, и не занимает места под данные.

При использовании данного метода может иметься множество различных путей для восстановления к одной точке. Встает задача выбора критерия оптимальности пути и задача отыскания оптимального пути.

Другой метод дублирования

Имеет смысл операция исключения из создаваемого элемента репозитория изменений относящихся к некоторому периоду, если известно, что в последующем, при восстановлении эта часть данных не потребуется. Таким образом, исключая эти данные, можно будет снизить объем анализа при выполнении операции восстановления и тем самым повысить ее скорость. Данный подход можно использовать для построения своеобразной схемы резервного копирования [6].

Рассмотрим пример, иллюстрирующий данный подход (см. рис. 6). Обратимся к операции резервного копирования проходящей в момент t_3. В этот момент создается R_2^1. А также из уже созданного в предыдущий момент t_2 элемента R_1^1 исключаются данные об изменениях за период от t_2 до t_3, заключенных в R_2^1, при этом образуется новый элемент репозитория обозначенный на иллюстрации $\dot{R}_1^1$. Вновь образованный элемент $\dot{R}_1^1$ обладает тем свойством, что при объединении с R_2^1 даст тот же результат, что и $R_1^1 \oplus R_2^1$, однако $\dot{R}_1^1$ несет в себе меньшее количество данных, а значит более выгоден для использования чем R_1^1. Та же операции происходит и в следующий момент времени резервирования t_4. Только теперь из $\dot{R}_1^1$ исключается еще и данные об изменениях за период от t_3 до t_4, и образуется элемент, обозначенный $\ddot{R}_1^1$. $\dot{R}_2^1$ в свою очередь является R_2^1 без изменений заключенных в R_3^1.

Как и в случае с предыдущим методом дублирования образованные дополнительные элементы могут носить индексный характер.

Из иллюстрирующего данный подход орграфа, изображенного на рис. 6, видно, что имеется несколько возможных путей восстановления.

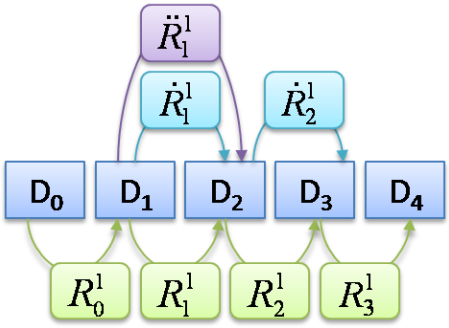

Рис. 6. Пример дублирования

Умозрительно, исходя и принципов создания рассматриваемых элементов репозитория видно, что наиболее оптимальным путем к D_4 является последовательность $\{R_0^1; \ddot{R}_1^1; \dot{R}_2^1; R_3^1\}$. Нужно, однако, заметить что в практической ситуации, когда вершин в орграфе и соединительных дуг будет сотни или даже тысячи, которые при этом будут переплетены особым образом, нахождение оптимального пути становиться не столь простой задачей.

Специализированные алгоритмы поиска пути восстановления

Для каждой схемы резервного копирования может быть составлен свой алгоритм нахождения оптимального пути с учетом ее специфических особенностей работы. Однако сплетение дублирующих элементов репозитория может быть сколь угодно сложным, и тем самым делать сложным построение специализированного алгоритма нахождения оптимального пути восстановления.

Практически необходимо уметь искать оптимальные пути восстановления при условии, что некоторые из элементов репозитория могут оказаться недоступными вследствие порчи или по другим причинам. Данное требование еще сильнее осложняет разработку специализированных алгоритмов.

Было бы удобно найти унифицированную процедуру поиска путей восстановления не зависящую от выбора алгоритма создания резервных копий.

Операция объединения элементов репозитория

Для понимания того какие приоритеты должны быть выбраны для поиска оптимальных путей, необходимо рассмотреть процедуру восстановления в целом.

Отыскав путь восстановления, система резервного копирования должна использовать информацию о поэтапном изменении резервируемых данных, хранящуюся в элементах репозитория, для создания копии состояния данных к моменту восстановления. В элементах репозитория хранятся различные версии файлов, изменяющихся со временем. При восстановлении необходимо определить, в каком элементе репозитория находится нужная версия каждого файла для восстановления. Фактически нужно копировать файлы с последними изменениями к моменту восстановления.

Нужно заметить, что скорость операции объединения зависит, прежде всего, от количества рассматриваемых элементов репозитория в пути восстановления. Менее очевидным является фактор количества содержащихся в них файлов: чем больше файлов в элементах репозитория, тем больше операций сравнения необходимо произвести.

Значительные временные расходы могут быть связаны с проблемой доступности различных элементов репозитория. На это влияет фактор выбора носителей: магнитные ленты с последовательным доступом, или жесткие диски. Имеет значение организация системы резервного копирования и хранения данных в целом. Об этих и других факторах, влияющих на производительность систем резервного копирования см. [7].

Однако вынесем эту совокупность факторов за пределы рассмотрения, и будем полагать, что все элементы репозитория равнодоступны. В этом случае непосредственно процесс копирования будет занимать одинаковое количество времени для любого корректного пути, ибо в любом случае будет скопировано одинаковое

количество файлов. Разница во времени работы процесса восстановления будет обусловлена вычислительной скоростью алгоритмов поиска пути восстановления и скоростью работы операции определения места нахождения нужных версий файлов.

Фактор объема элементов репозитория в пути восстановления

Приведем пример иллюстрирующий недостаточность учета одного лишь критерия минимальности количества элементов репозитория в пути восстановления.

Рассмотрим класс схем резервного копирования, в результате работы которых создаются элементы репозитория так, что для восстановления к некоторым моментам существует несколько путей восстановления одинаково минимальной длины.

Для иллюстрации рассмотрим следующий простотой случай.

Начальное состояние данных D_0 пустое. D_1 включает в себя только некоторый файл A, то есть $D_1=\{A\}$ и $R(0,1)=\{A\}$. Далее этот файл изменяется и $D_2=\{B\}$, создается элемент репозитория $R(0,2)$, содержащий изменение данных от D_0 до D_2 состоящее из B. Далее процесс продолжается так, как отображено на рис. 7. Причем, A^- в элементе $R(1,2)$ означает данные об удалении A. В результате для восстановления к моменту времени t_3 имеются два пути: $P_1=\{R(0,1);\ R(1,2)\}$ и $P_2=\{R(0,2);\ R(2,1)\}$. Оба пути состоят из двух элементов репозитория каждый. Однако для объединения элементов репозитория для P_1 требуется проанализировать больше на $\{A\}$ данных, чем для P_2. Если предположить что $\{A\}$ является не файлом, а совокупностью из нескольких тысяч файлов, то отличие времени работы процедуры восстановления для рассматриваемых путей может оказаться значительным.

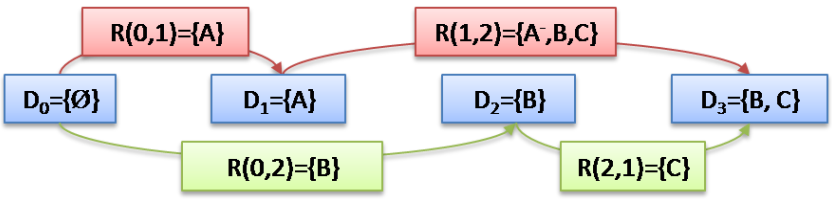

Рис. 7. Пример важности фактора объема элементов репозитория в пути

Поэтому, для того, чтобы иметь возможность находить действительно оптимальные пути, нужно учитывать не только критерий количества элементов репозитория входящих в путь, но и их суммарный объем.

Выбор алгоритма

Таким образом, нужен кратчайший путь, обладающий дополнительным свойством. Данную задачу можно рассматривать как задачу с некоторыми дополнительными ограничениями или как многоцелевую задачу, в которой учитывается не только длина, но и свойство объема файлов в элементах репозитория. Эти усложнения будут, вообще говоря, сильно увеличивать вычислительную работу, и с практической точки зрения более просто найти кратчайшие пути с минимальной длиной и выбрать среди них тот, который обладает нужным свойством [8].

Вообще говоря, необходимо получать простые цепи, ибо наличие циклов в пути означает непригодность для восстановления.

Существует много алгоритмов, в которых заложено требование получения простых цепей, однако, как показывают исследования, это требование значительно осложняет работу алгоритма, что влечет за собой значительное увеличение времени работы алгоритма [9].

Следует заметить, что, как правило, для алгоритмов резервного копирования нет возможности получить цикл в пути восстановления, однако в принципе это не исключено. Ведь существует принципиальная возможность создавать элементы репозитория хранящие изменения по отношению к предыдущему состоянию данных как, к примеру, в обратной инкрементной схеме (англ. *reverse incremental*). Нужно иметь в виду такую возможность и в действительно универсальном алгоритме восстановления это нужно учитывать.

Можно было бы создать процедуру восстановления с использованием алгоритмов исключающих циклы для тех схем резервного копирования, где наличие таких путей не исключено и другой алгоритм в противном случае. Но чтобы добиться действительной универсальности процедуры нахождения пути необходимо избавиться от требования излишних пользовательских параметров.

В таком случае имеет смысл использовать гибридные алгоритмы. При работе такого алгоритма, при поиске каждого последующего оптимального пути, изначально работает алгоритм, который не исключает появление циклов, найденный путь проверяется, и в случае обнаружения цикла, оптимальный путь находиться с использованием другого алгоритма, гарантирующего получение простого пути.

По данным исследований [10] гибридный алгоритм в составе с алгоритмом Йена превосходит по скорости поиска многие другие (все исследованные) алгоритмы поиска простых путей. В то же время для схем резервного копирования, исключающих наличие циклов, алгоритм Йена не выполняется.

Таким образом, применение такого гибридного алгоритма отлично подходит для поиска оптимального пути восстановления в системах резервного копирования.

Для поиска цепей с одним требованием минимизации длины пути в составе гибридного алгоритма и алгоритма Йена, будем использовать алгоритм Дейкстры.

Алгоритм Дейкстры

Наиболее эффективный алгоритм решения задачи об отыскании кратчайшего пути первоначально дал Дейкстра [11]. В общем случае этот метод основан на приписывании вершинам временных пометок, причем пометка вершины дает верхнюю границу длины пути от начала пути s к этой вершине. Эти пометки постепенно уменьшаются с помощью некоторой итерационной процедуры, и на каждом шаге одна из временных пометок становится постоянной. Последнее указывает на то, что пометка уже не является верхней границей, а дает точную длину кратчайшего пути к рассматриваемой вершине. Алгоритм останавливается, как только пометка, принадлежащая конечной вершине t искомого пути, станет постоянной. Если не завершать работу алгоритма на этом – будет найдено дерево кратчайших путей из начала искомого пути T_s [8].

Следует также заметить, что данный алгоритм может быть применен для отыскания дерева кратчайших путей T_t из всех вершин к конечной t. Для этого будет достаточно изменить направление всех дуг на прямо противоположное, и считать конечную вершину – началом пути [12].

Алгоритм Дейкстры строит кратчайший (s, t)-путь для графа G за время $O(n^2)$, где $n=|G|$ – количество вершин [11, 13, 14].

Замечания по применению алгоритма Дейкстры в отношении к задачи нахождения пути восстановления

При реализации алгоритма в случае поиска пути восстановления необходимо учесть возможность множественных дуг между двумя вершинами. Поэтому при проверке вершин с временными метками необходимо рассматривать все возможные варианты дуг. А при построении непосредственно пути учитывать не только последовательность прохождения вершин, но и использованные при этом дуги.

Удобно связывать с каждой вершиной v еще один список меток $\Theta(v)$, в который заносить идентификатор указывающей на v дуги в (s, v)-пути, имеющем минимальный вес среди всех (s, v)-путей, проходящих через вершины, получившие к данному моменту постоянные метки. Таких путей может быть несколько, поэтому $\Theta(v)$ в общем случае является списком.

После того как вершина t получила свою постоянную метку, с помощью меток Θ можно легко указать последовательность вершин и дуг составляющих кратчайший (s, t)-путь.

Важно не пропустить возможные кандидаты на оптимальный путь при учете двух рассмотренных выше факторов. Заметим, что для работы алгоритма Йена в купе с алгоритмом Дейкстры для верной работы не требовалось бы указания всех возможных дуг в кратчайшем пути в работе алгоритма Дейкстры, потому, что алгоритм Йена, перед тем как запускать каждый раз алгоритмом Дейкстры удаляет

уже просмотренные дуги. Таким образом, нет возможности пропустить нужный путь. Для гибридного же алгоритма условие нахождения всех возможных путей в T_t является основополагающим.

Идеально для задачи подходит представление мультиграфа списками смежности.

Так как требуется найти путь кратчайший в смысле количества использованных элементов репозитория, следует считать все веса равными единице.

Алгоритм Йена

Опишем алгоритм [15], представленный Йеном, позволяющий находить K кратчайших простых путей.

Рассмотрим ориентированный мультиграф $G(N, A)$, где N – множество вершин, а A – множество дуг. При этом $n=|N|$ – количество вершин, а $m=|A|$ количество дуг. При этом каждой дуге $(x,y) \in A$ поставлен в соответствие вес $c_{xy} \in \mathbb{R}$.

Пусть $P^k = \{s, v_2^k, v_3^k, ..., v_{q_k}^k, t\}$ – k-ый кратчайший (s, t)-путь, где v_i^k соответственно его вершины. Пусть P_i^k – «отклонение от пути P_i^{k-1} в точке i». Под этим понимается следующее: P_i^k – кратчайший из путей, совпадающий с P_i^{k-1} от s до i-й вершины, а затем идущий к $(i+1)$-й вершине по дуге, отличной от дуг к $(i+1)$-й вершине тех (ранее уже построенных) кратчайших путей P^j $(j=1,2,...,k-1)$, которые имеют те же самые начальные подпути от s к i-й вершине, что и P^{k-1}. P_i^k приходит в вершину t по кратчайшему подпути, не проходящему ни через одну из вершин $\{s, v_2^{k-1}, v_3^{k-1}, ..., v_i^{k-1}\}$, участвующих в формировании первой части пути P_i^k. Отсюда кстати следует, что путь P_i^k необходимо должен быть простой цепью.

Алгоритм начинает работу с нахождения кратчайшего пути P^1 с помощью алгоритма Дейкстры. Далее находят следующий оптимальный путь P^k, $k=2$. Для этого ищется набор лучших отклонений от заданного пути, используя на каждом шаге для этого алгоритм Дейкстры.

Затем ищется набор лучших отклонений от заданного пути. Кратчайший среди найденных кандидатов P_i^k и будет являться P^k.

Далее переходят к поиску P^k,для $k=3$. И так далее.

Чтобы найти более подробное описание алгоритма Йена см. например [8,15].

При поиске пути восстановления необходимо найти все кратчайшие пути, поэтому работу алгоритма следует остановить, как только следующий найденный путь будет обладать большей длиной, чем уже найденные. Затем из найденных путей следует выбрать оптимальный по второму фактору. Для этого достаточно найти путь с минимальным суммарным объемом входящих в него элементов репозитория.

Временная сложность алгоритма оценивается $O(Kn^3)$, с учетом того, что используется алгоритм Дейкстры с неотрицательными весами.

Гибридный алгоритм

Кратчайший *(s, t)*-путь, который отклоняется от P^k на вершине v_i^k, то есть P_i^k, ищется в виде $Root_{P^k}(s, v_i^k) \diamond (v_i^k, j) \diamond T_t(j)$, так, что дуга (v_i^k, j) не принадлежит никакому из кандидатов P^j найденных до этого. Где $Root_{P^k}(s, v_i^k)$ – подпуть P^k от s до v_i^k; операция объединения путей p и q, обозначена как $p \diamond q$, при этом объединенный путь образован путем p и следующим за ним q.

Создание каждого нового кандидата таким путем более эффективно, так как процедура отыскания кратчайшего пути на каждом этапе сводится к проблеме выбора нужной дуги (v_i^k, j), что может быть сделано за время $O(1)$. В отличие от применения алгоритма Дейкстры, который в нашем случае с неотрицательными весами выполняется за время $O(n^2)$. Однако такой алгоритм может вернуть путь с циклами, когда $Root_{P^k}(s, v_i^k)$ и $T_t(j)$ содержат одинаковые узлы. В таком случае гибридный алгоритм переключается на использование алгоритма Йена.

Для того чтобы выбор дуги (v_i^k, j) был скорым, веса c_{xy} должны быть заменены на уменьшенные: $c^*_{xy} = c_{xy} - c(T_t(x)) + c(T_t(y))$ для всех $(x, y) \in A$. Так $c_{xy} \geq 0$ для любых двух вершин $(x, y) \in A$, и $c_{xy} = 0$ для $(x, y) \in T_t$, и искомая дуга, начинающаяся с v_i^k будет та, что имеет минимальный уменьшенный вес.

Для работы гибридного алгоритма необходимо на предварительной стадии определить T_t, и заменить веса на уменьшенные.

В худшем случае время работы гибридного алгоритма будет идентично алгоритму Йена, то есть $O(Kn^3)$, в лучшем же случае, когда в отклонения циклов не будет, следует ожидать $\Omega(Kn + n^2 + m \log n)$.

Чтобы найти более подробное описание рассмотренного гибридного алгоритма см. [10].

Опыт применения и выводы

Рассмотренная процедура построения оптимальных путей была реализована в разработанной авторской системе резервного копирования. В данную систему были предварительно внедрены различные схемы резервирования: полное; инкрементное; дифференциальное; мультиуровневое; схемы, реализующие два метода дублирования, описанные выше; а также другие, экспериментальные схемы. Были разработаны специализированные процедуры поиска путей восстановления для каждой из реализованных схем. При разработке каждой из них требовалось спроектировать уникальных для данной схемы алгоритм, что по объему работы сравнимо со временем, затраченным на создание универсального алгоритма. Более того, универсальный алгоритм позволяет учитывать проблемы с недоступностью использования каких-либо из элементов репозитория и строить оптимальные пути с учетом данных обстоятельств.

В ходе проведения данной исследовательской работы был построен универсальный метод нахождения оптимального пути восстановления. Опыт использования показал его пригодность. Производительность универсальной процедуры поиска сравнима с работой специализированных алгоритмов. При внедрении в систему новых схем резервного копирования теперь не стоит задача разработки специализированных алгоритмов восстановления.

Стоит заметить что, с практической точки зрения, для тривиальных схем имеет смысл отключать процедуру гибридного алгоритма и ограничить использование алгоритмом Дейкстры, дабы максимизировать производительность.

Список литературы

1. A. L. Chervenak, V. Vellanki, Z. Kurmas, and V. Gupta. Protecting File Systems: A Survey of Backup Techniques. In Proceedings. Joint NASA and IEEE Mass Storage Conference, 1998. www.isi.edu/~annc/papers/mss98final.ps

2. Z. Kurmas, A. Chervenak. Evaluating backup algorithms. Proc. of the Eighth Goddard Conference on Mass Storage Systems and Technologies, 2000. http://www.cis.gvsu.edu/~kurmasz/papers/kurmas-MSS00.pdf

3. Казаков В.Г., Федосин С.А., Иконников С.Е. Моделирование операций резервного копирования для прогнозирования использования объема репозитория. Труды XV Всероссийской научно-методической конференции Телематика’2008. – Санкт-Петербург.: Санкт-Петербургский государственный университет точной механики и оптики, 2008. – С. 111 - 113.

4. Казаков В.Г., Федосин С.А. Анализ алгоритмов резервного копирования для получения оценок объема репозитория. Кибернетика и высокие технологии XXI века. IX международная научно-техническая конференция. –Воронеж.: НПФ «Саквоее» ООО, 2008. – С. 697-709 с.

5. A. Costello, C. Umans, and F. Wu. Online backup and restore. Unpublished Class Project at UC Berkeley, 1998. http://www.nicemice.net/amc/ research/backup/

6. Z. Kurmas. Reasoning Behind the Z Scheme. 2002. http://www.cis.gvsu.edu/~kurmasz /Research/BackupResearch/rbzs.html

7. Казаков В.Г., Федосин С.А. Технологии и алгоритмы резервного копирования / Всероссийский конкурсный отбор обзорно-аналитических статей по приоритетному направлению "Информационно-телекоммуникационные системы", 2008. - 49 с.

8. Кристофидес Н. Теория графов. Алгоритмический подход. – М.: Мир, 1978.

9. D. Eppstein. Finding the k shortest paths. 35th IEEE Symp. Foundations of Comp. Sci., Santa Fe, 1994, pp. 154-165. http://www.ics.uci.edu/~eppstein/pubs/Epp-TR-94-26.pdf

10. M. M. B. Pascoal, Implementations and empirical comparison for K shortest loopless path algorithms, The Ninth DIMACS Implementation Challenge: The Shortest Path Problem, November 2006
http://www.dis.uniroma1.it/~challenge9/papers/pascoal.pdf

11. Dijkstra E. W. (1959), A note on two problems in connection with graphs, Numerische Mathematik, 1, p. 269.

12. E. Q. V. Martins, M. M. B. Pascoal and J. L. E. Santos, The K shortest paths problem, CISUC, 1998. http://www.mat.uc.pt/~marta/Publicacoes/k_paths.ps.gz

13. Лекции по теории графов. В.А.Емеличев, О.И.Мельников, В.И.Сарванов, Р.И.Тышкевич. – М.: Наука, 1990.

14. Белов В.В., Воробьев Е.М., Шаталов В.Е. Теория графов. – М.: Высш. шк., 1976

15. J. Y. Yen. Finding the K shortest loopless paths in a network. Management Science, 17:712–716, 1971.