*Моделирование и анализ информационных систем.* Т. 23, № 4 (2016), с. 479–491
*Modeling and Analysis of Information Systems.* Vol. 23, No 4 (2016), pp. 479–491



**DOI:** 10.18255/1818-1015-2016-4-479-491

УДК 004.056.3

# Резервное копирование с хранением в базе данных

**Таранин С. М.**

*получена 9 сентября 2015*

**Аннотация.**
В данной работе представлен обзор некоторых технологий, которые используются в современных системах резервного копирования, кратко описаны их преимущества и недостатки. Далее рассматривается подход к реализации системы резервного копирования с сохранением файлов в базе данных. Предлагается разбивать копируемые файлы на блоки фиксированной длины. Каждый блок представляет собой последовательность байт. Длина блока может быть адаптивной, т.е. меняться в зависимости от типа или размера файла. В таком виде содержимое файлов предлагается хранить в одной таблице, а информацию о них: имена, атрибуты и связи между ними – хранить в другой таблице. Сведения о сохраненных файлах и папках предлагается хранить не только в базе данных на сервере, но и на стороне клиента в некоторой иерархической структуре. Она содержит набор записей и представляет собой модель копируемой директории. Наличие такой модели позволяет отслеживать изменения в файловой системе клиента без выполнения дополнительных запросов к базе данных. В случае если файл изменен, в базу копируются только его изменившиеся блоки. При этом в модели на стороне клиента также обновляется информация, например дата изменения отредактированного документа. Удаляются записи об удаленных файлах и папках. Таким образом, уменьшается нагрузка на канал передачи данных. В статье описаны алгоритмы сохранения и восстановления данных, а также рассмотрены факторы, влияющие на скорость их работы. Наглядно показана зависимость скорости сохранения и восстановления данных от мелкости разбиения файлов, а также от структуры копируемой директории.

**Ключевые слова:** файл, данные, резервное копирование, база данных, блок, модель

**Для цитирования:** Таранин С. М., "Резервное копирование с хранением в базе данных", *Моделирование и анализ информационных систем*, **23**:4 (2016), 479–491.

**Об авторах:**
Таранин Сергей Максимович, orcid.org/0000-0001-8117-7358, аспирант,
Ярославский государственный университет им. П.Г. Демидова,
ул. Советская, 14, г. Ярославль, 150003 Россия, e-mail: staranin0208@yandex.ru

## Введение

Несмотря на широкое применение цифровых устройств для обработки документов, еще рано говорить о полном отказе от бумажных носителей. И если защите информации, представляющей собой персональные данные или коммерческую тайну, уделяется много внимания, то защита «иных» форм данных – это личное дело их

владельца. Например, для студентов это могут быть лекции, рефераты, незавершенные курсовые работы. Сюда же можно отнести информацию, содержащуюся в записных книжках и ежедневниках, личные архивы и многое другое. Вся эта информация представляет определенную ценность для ее владельца.

Существует проблема обеспечения целостности такой информации на автоматизированном рабочем месте (АРМ) с надежностью, близкой к надежности бумажных носителей. Разумеется, надежность бумажного носителя – это, в данном случае, идеал, к которому нужно стремиться. Автоматизированное рабочее место – это программно-технический комплекс, предназначенный для автоматизации деятельности определенного вида. Он представляет собой профессионально-ориентированную малую вычислительную систему, расположенную непосредственно на рабочем месте специалиста и предназначенную для автоматизации его работы. Обычно в качестве АРМ выступает персональный компьютер (ПК) с предустановленным программным обеспечением (ПО). Это может быть система проектирования, система электронного документооборота, справочная система и др.

Обеспечение целостности данных на АРМ в основном достигается путем резервного копирования. Это процесс создания копии данных на носителе, предназначенном для восстановления данных в оригинальном или новом месте их расположения в случае их повреждения или разрушения. Копии записываются либо на магнитную ленту, либо на жесткий диск. Скорость записи на последний выше, однако его надежность ниже, поэтому еще рано говорить о полном отказе от ленточных носителей. Для того, чтобы объединить в одной системе резервного копирования скорость сохранения информации на диски с надежностью ленточных накопителей, была придумана схема D2D2T [1], при которой данные сначала копируются на жесткий диск, затем на ленту. Во время резервного копирования основные приложения приостанавливают свою работу. Применение технологии D2D2T позволяет свести к минимуму время простоя системы, поскольку приложения останавливаются на время, необходимое для копирования данных на жесткий диск. Потом приложения возобновляют работу, а данные копируются с диска на ленту. Однако применение данной технологии не решает следующие проблемы традиционного резервного копирования:

- избыточность копируемой информации;
- непредсказуемое время восстановления данных;
- риск потери данных из-за невозможности восстановления.

Последнее время активно развивается технология непрерывной защиты данных (CDP) [3, 6, 7]. В отличие от традиционного резервного копирования, здесь не требуется определять время создания копий. Специальный сервис постоянно отслеживает изменения в файловой системе и копирует только изменившиеся данные. Данная статья посвящена вопросам разработки подобной системы.

# 1. Современные технологии в системах резервного копирования

## 1.1. Непрерывная защита данных

Система резервного копирования, реализующая технологию CDP, должна иметь возможность восстановить любую версию каждого хранимого файла на любой момент времени. Существуют системы, которые в отличие от традиционных производят операции копирования очень часто и позиционируются разработчиками как CDP-решения. В связи с этим системы стали делить на «настоящие» CDP (англ. True CDP), и на «почти непрерывные» CDP решения (англ. near-Continuous CDP). Технология True CDP отличается от традиционного резервного копирования и near-Continuous CDP прежде всего тем, что операции копирования проходят сразу при изменении данных, а не по заданному расписанию.

Существуют споры вокруг преимуществ True CDP. Данная технология имеет ограниченное применение, и можно привести примеры, когда ее использование является нежелательным. Современные операционные системы для ускорения файловых операций используют кэширование. Измененные блоки файла, предназначенные для записи на диск, некоторое время хранятся в кэше, и лишь в свободное от других операций время сохраняются физически. Такой подход называют методом отложенной записи (англ. lazy write) [5]. По этой причине система резервного копирования, реализующая технологию True CDP, не может видеть все изменения в файлах, пока операционная система не сбросит их на диск из кэша. Одним из недостатков также является негативное влияние на производительность сети при обработке файлов большого размера, например видеозаписей или файлов систем проектирования.

## 1.2. Исключение дублирования данных

Для исключения повторяющихся данных применяется механизм дедубликации (англ. de-duplication) [8, 9]. Он представляет собой метод сжатия информации, когда поиск копий производится по всему массиву данных, а не в пределах одного файла. Главным преимуществом использования данной технологии является существенная экономия дискового пространства. В процессе дедубликации данные разбиваются на блоки. Каждый блок идентифицируется уникальным числом (хэш, номер корректировки). Каждый новый хэш сравнивается с вычисленными ранее. Если он уникален, то добавляется в список, иначе блок заменяется ссылкой на уже существующий хэш. Сложность реализации заключается в том, чтобы не снизить производительность системы резервного копирования. Так, использование дедубликации может существенно ухудшить показатели скорости восстановления данных. Дедубликация данных может быть выполнена до того, как данные будут записаны в целевое хранилище (англ. in-line de-duplication), или уже после записи (англ. post-process de-duplication). Системы, в которых процесс дедубликации выполняется после записи, требуют хранилища большего объема, так как вначале данные сохраняются полностью.

### 1.3. Сервисы удаленного резервного копирования

Удаленное резервное копирование (англ. remote backup или online backup) представляет собой сервис, предлагающий пользователям систему хранения и резервного копирования данных [1, 2]. Пользователю предоставляется клиентская часть системы, которая собирает, сжимает, шифрует и отправляет файлы на сервер поставщика услуг резервного копирования. Доступ к файлам можно получить в любом месте, где есть подключение к сети Интернет. Основные преимущества сервисов удаленного резервного копирования в том, что копии хранятся отдельно от оригиналов, а участие пользователя практически не требуется. Использование таких сервисов избавляет пользователя от необходимости устанавливать и поддерживать собственные системы хранения. Однако скорость создания резервных копий и восстановления значительно ограничена и зависит от скорости доступа к сети Интернет.

## 2. Резервное копирование с хранением информации в базе данных

Рассмотрим подход к реализации системы резервного копирования на клиент-серверной архитектуре [10], где клиентская часть генерирует запросы на сохранение или восстановление данных, а серверная часть обрабатывает запросы от клиентов и упорядочивает сохраненные данные.

Для хранения пользовательских данных на сервере и управления ими можно ввести две таблицы Model и Data. Первая таблица представляет собой модель копируемой директории. Каждая запись в модели хранит информацию об объекте файловой системы. Далее под таким объектом будем понимать файл или папку копируемой директории. Связь между таблицами Model и Data – «один ко многим» соответственно.

Таблица Model содержит следующие поля:

1. ID – первичный ключ записи
2. NAME – имя объекта файловой системы. Имя файла должно быть указано с расширением.
3. PARENT – идентификатор «родителя» – записи, соответствующей некоторой папке.
4. ISFOLDER – запись соответствует папке.
5. HASH – хэш-код файла для проверки целостности данных после восстановления из резервной копии. Для ускорения обработки файлов вместо вычисления хэш-кода можно хранить номер корректировки или дату последнего изменения.

Поля NAME и PARENT представляют собой естественный ключ записи таблицы, поскольку в одной папке не может быть 2-х файлов с одинаковым именем и расширением.

Таблица Data хранит данные файлов в виде блоков и содержит следующие поля:

1. ID – первичный ключ записи
2. FILEID – идентификатор записи таблицы Model.
3. BLOCK – последовательность байт некоторого файла копируемой директории в виде строки символов.
4. HASH – хэш-код. Для проверки целостности данных после восстановления из резервной копии.

Модель копируемой директории предлагается хранить не только на сервере, но и на клиентской части в некоторой иерархической структуре. Это позволяет отслеживать изменения без выполнения дополнительных запросов к БД [11, 12, 13]. В случае если файл изменен, то копируются только его изменившиеся блоки [14, 15]. Размер блоков может быть фиксированным, а может выбираться в зависимости от типа или размера файла (адаптивная дедубликация) [4]. Таким образом, уменьшается нагрузка на канал передачи данных.

## 2.1. Сохранение данных

Предлагается следующий рекурсивный алгоритм резервного копирования:

1 Для каждого файла из текущей директории:

- 1.1 Создаем запись для таблицы Model и сохраняем ее insert-запросом к БД.
- 1.2 Получаем идентификатор сохраненной записи.
- 1.3 Читаем файл с диска блоками, формируя массив записей для сохранения в таблицу Data.
- 1.4 Сохраняем записи в таблице Data insert-запросом к БД.

2 Для каждой папки из текущей директории:

- 2.1 Формируем запись для таблицы Model и сохраняем ее insert-запросом к БД.
- 2.2 Запускаем данный алгоритм для файлов в этой директории.

3 Считаем хэш-код для текущей директории и сохраняем его в поле HASH соответствующей записи таблицы Model update-запросом.

### 2.1.1. Оценка трудоемкости алгоритма

Глубина дерева рекурсии равна глубине копируемой директории. При обработке очередного узла сканируемой директории наиболее трудоемкой задачей является обработка файла, т.к. от его размера зависит количество обращений к БД. Поэтому необходимо свести к минимуму количество запросов при сохранении приемлемого уровня обеспечения целостности данных за счет разбиения файлов на блоки. Оценим количество запросов к БД при сканировании очередного узла дерева рекурсии.

Пусть n – количество файлов в текущей директории, m – количество папок в текущей директории, sgm – длина блока. Тогда количество запросов можно выразить следующей формулой:

$$F(m,n) = \sum_{i=1}^{n} \left( \frac{l_i}{sgm} + 2 \right) + m + 1,$$

где $sgm < l_i, l_i$ – длина i-го файла в текущей директории.

Обозначим $s_i = \frac{l_i}{sgm} + 1$ – число блоков i-го файла. Тогда формула примет более компактный вид:

$$F(m,n) = \sum_{i=1}^{n} (s_i + 1) + m + 1.$$

Для каждого файла выполняется $s_i$ запросов для записи блоков в таблицу Data, один запрос для записи информации о файле в таблицу Model, m запросов для записи информации о папках в таблицу Model и один запрос для обновления записи в таблице Model (сохранение хэша текущей директории). Каждой директории ставится в соответствие уникальный идентификатор – первичный ключ записи в таблице Model. Общее количество запросов к БД при обработке директории можно представить следующей формулой:

$$F_k = F(m_k, n_k) = \sum_{i=1}^{n_k} \left( \frac{l_{k_i}}{sgm} + 2 \right) + \sum_{j=1}^{m_k} F(m_{k_j}, n_{k_j}) + m + 1 =$$

$$\sum_{i=1}^{n_k} (s_{k_i}) + \sum_{j=1}^{m_k} F(m_{k_j}, n_{k_j}) + m + 1,$$

где $k$ – идентификатор текущей директории. Здесь мы учитываем результаты обработки вложенных папок. Переменную $l_{k_i}$ мы понимаем здесь как «длина $i$-го файла директории с идентификатором $k$». Таким образом, скорость работы алгоритма зависит от глубины вложенности папок и от отношения $\frac{l_{k_i}}{sgm}$. Длина блока может быть константой, а может меняться в зависимости от типа или объема файла.

Посмотрим, как меняется скорость обработки данных в зависимости от длины блока. Тестовый стенд представляет собой два компьютера, объединенные в сеть. Один из них играет роль сервера с установленной СУБД Linter 6.0.18.9 (процессор: Intel Celeron 2.8 GHz, память: 3 GB) [11, 16]. Второй ПК представляет собой клиентскую часть системы, которая содержит пользовательские данные (процессор: Intel Core i5-2500K, 3.3 GHz, память 8 GB ). На клиентском ПК запускается приложение, которое обрабатывает данные пользователя и отправляет их на сервер. В качестве тестовых данных возьмем папку с файлами. близкими по размеру (360 фото 1600x1200, 0,8–1,5 МБ каждая) без вложенных директорий. Результат обработки данных по описанному алгоритму с разными размерами блока представлен на рисунке 1:

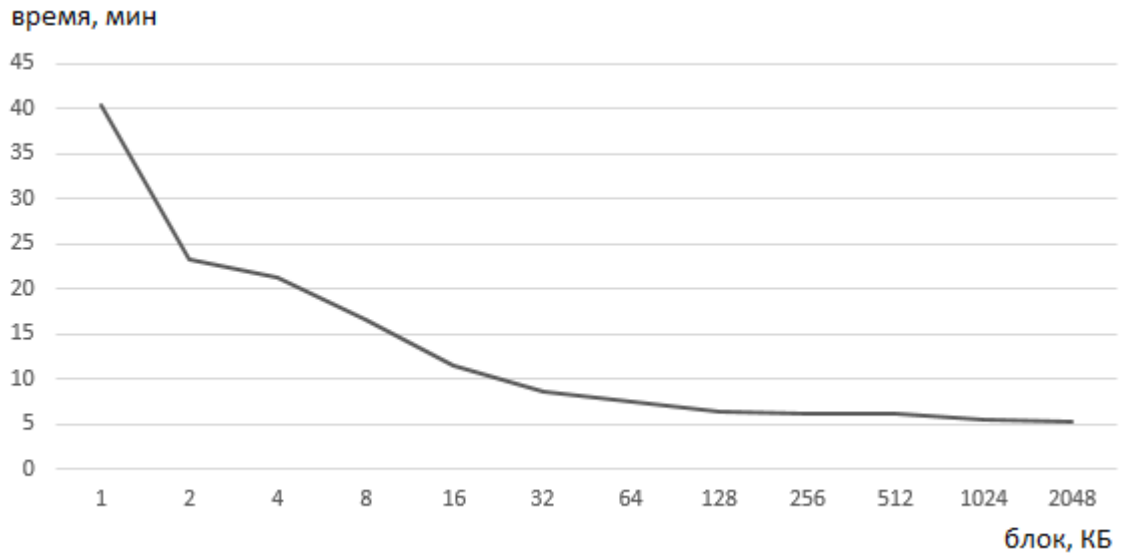


Рис. 1. Зависимость времени сохранения от размера блока
(все файлы в одной папке)
Fig. 1. The dependence of the saving time of the block size
(all the files in the one folder)

Наибольший рост производительности достигается после увеличения размера блока с 1 до 2 КБ (42%). При последующем увеличении его размера производительность растет медленнее. Поэтому оптимальный размер блока для данного случая составляет 64 – 128 КБ, т.к. дальнейшее его увеличение незначительно влияет на скорость работы. При размере блока 64 КБ каждый файл тестового стенда был разбит в среднем на 22 блока, а при 128 КБ – на 11 блоков.

Посмотрим теперь, как изменится скорость работы, если распределить файлы по папкам. Пусть каждая папка имеет 2 вложенные папки. В каждой папке, кроме корневой, по 12 файлов. Тогда директория будет иметь структуру как на рисунке 2.

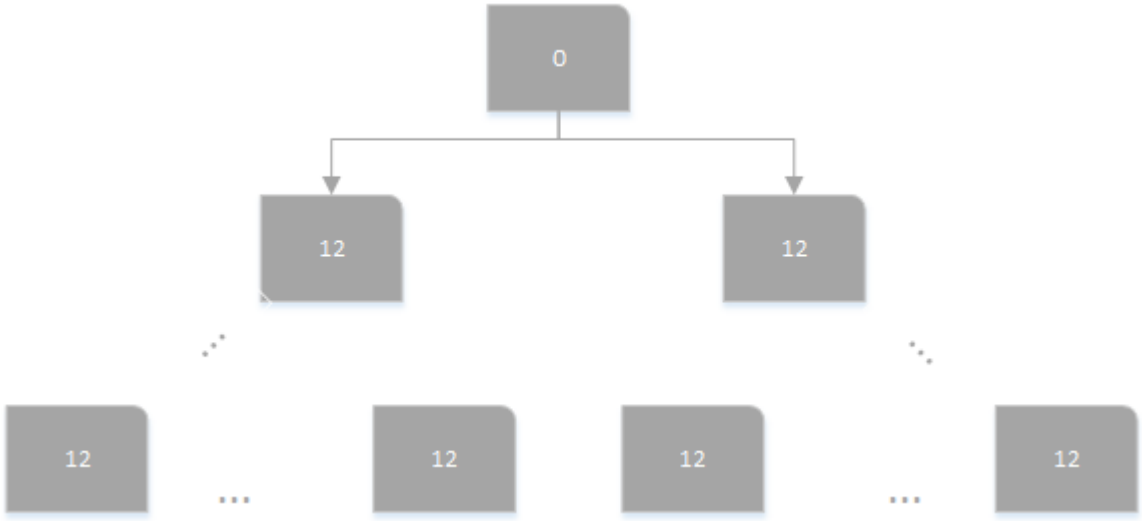


Рис. 2. Структура тестовой директории
Fig. 2. The structure of the test directory

Результат обработки данных представлен на рисунке 3:

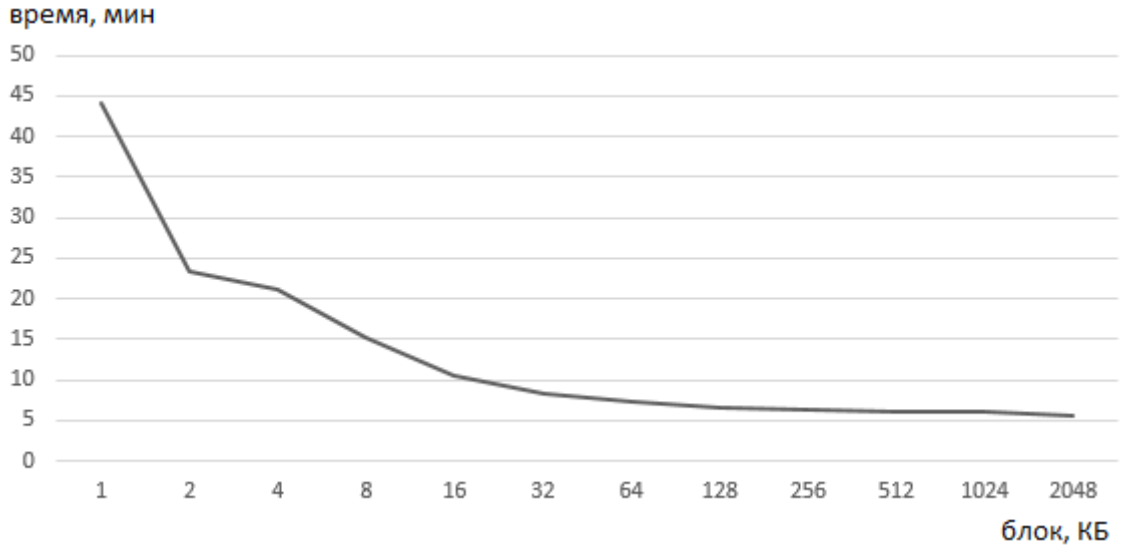


Рис. 3. Зависимость времени сохранения от размера блока
(все файлы равномерно распределены по вложенным папкам)
Fig. 3. The dependence of the saving time of the block size
(all the files are evenly distributed in the subfolders)

Исходя из опытных данных можно сделать предположение о том, что оптимальный размер блока 64 – 128 КБ. Ветвление структуры файловой системы практически не повлияло на скорость работы.

## 2.2. Восстановление данных

Для восстановления файлов из БД можно использовать рекурсивный алгоритм, входными параметрами которого являются путь до корневой директории – source, а также идентификатор соответствующей записи таблицы Model – parentID.

1 По parentID получим набор записей из таблицы Model с информацией о файлах и папках корневой директории.

2 Для каждой записи из этого набора:

- 2.1 Формируем путь к объекту файловой системы (source плюс значение поля NAME)
- 2.2 Если значение поля ISFOLDER равно «истина», то
  - 2.2.1 Создаем директорию, если такой не существует
  - 2.2.2 Восстанавливаем содержимое созданной директории, передав на вход данного алгоритма путь до нее и идентификатор записи.
- 2.3 Иначе:
  - 2.3.1 Создаем файл.
  - 2.3.2 По ключу записи получаем содержимое этого файла из таблицы Data в виде набора записей.

2.3.3 Записываем последовательно содержимое полей BLOCK в созданный файл.

Наиболее трудоемкой процедурой является последовательная запись данных в файл. Она зависит от размера блоков, на которые файл был разбит перед сохранением. Чем меньше размер блока, тем больше последовательных операций записи в файл будет выполнено. Зависимость времени восстановления данных от размера блока представлена на рисунке 4.

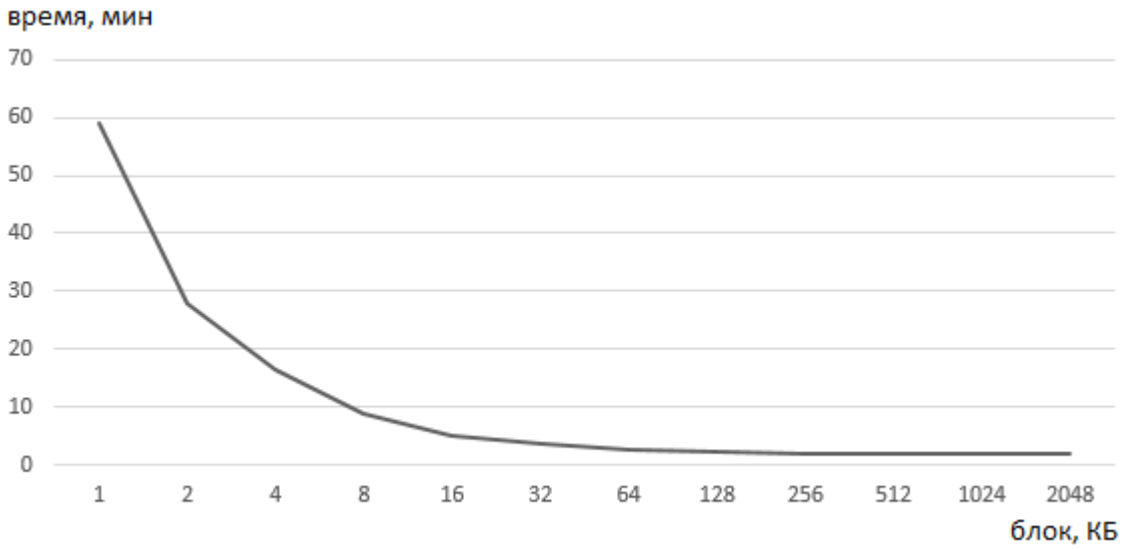


Рис. 4. Зависимость времени восстановления от размера блока

Fig. 4. The dependence of the recovery time of the block size

Как и в случае сохранения, оптимальный размер блока составляет 64 – 128 КБ, поскольку дальнейшее его увеличение незначительно влияет на скорость работы алгоритма.

Мелкость разбиения файлов влияет не только на скорость сохранения и восстановления. В каждой записи таблицы Data хранится хэш-код и идентификатор записи таблицы Model. Чем меньше размер блока, тем больше дополнительной информации, кроме пользовательских данных, приходится хранить в БД. Зависимость размера резервной копии от размера блока для десяти файлов размером 0.8 МБ представлена на рисунке 5.

Таким образом, оптимальное разбиение для сохранения и восстановления данных составляет около 11–22 блоков на файл. Такое разбиение незначительно влияет на размер БД, а также позволяет сохранить приемлемую скорость обработки данных.

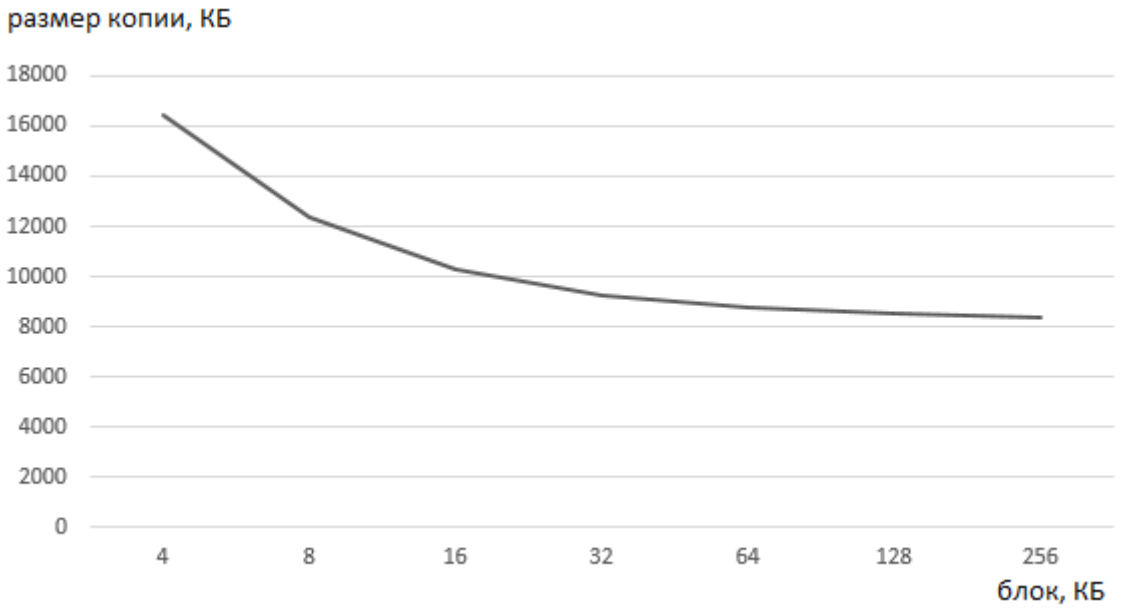


Рис. 5. Зависимость размера резервной копии от размера блока
Fig. 5. The dependence of the backup size of the block size

## 2.3. Поиск изменившихся данных

После полного резервного копирования данных запускается непрерывный процесс сканирования файловой системы на наличие изменений. Как только процесс обнаружит, что файловая система изменена, он должен сравнить новые данные с данными в модели, обновить модель и отправить изменения на сервер.

Ниже предлагается рекурсивный алгоритм сканирования файловой системы, состоящий из двух этапов. Сначала производится поиск удаленных файлов и папок. Далее происходит поиск изменений в существующих файлах, и сохранение новых данных.

На первом этапе выполняется следующая последовательность действий:

1 По parentID получим набор записей из таблицы Model с информацией о файлах и папках корневой директории.

2 Для каждой записи из этого набора:

2.1 Формируем путь к объекту файловой системы (source плюс значение поля NAME)

2.2 Если значение поля ISFOLDER равно «истина», то

2.2.1 Если такой директории не существует, удаляем данные вложенных файлов из таблицы Data и соответствующие записи из таблицы Model.

2.2.2 Иначе, выполняем аналогичные действия для данной директории.

2.3 Иначе (ISFOLDER равно «ложь»):

2.3.1 Если файл не существует, удаляем соответствующие данные из таблиц Data и Model.

2.3.2 Иначе, переходим к следующей записи выборки.

На втором этапе выполняется следующая последовательность действий:

1 Для каждого файла из текущей директории:

2 Если в модели на стороне клиента нет записи о файле, то:

2.1 Создаем запись для таблицы Model и сохраняем ее insert-запросом к БД.

2.2 Получаем идентификатор сохраненной записи.

2.3 Читаем файл с диска блоками, формируя массив записей для сохранения в таблицу Data.

2.4 Сохраняем записи в таблице Data insert-запросом к БД.

3 Иначе (в модели на стороне клиента есть запись о файле):

3.1 Вычисляем его хэш-код и сравниваем с хэш-кодом клиентской модели.

3.2 Если коды совпадают, переходим к следующему файлу.

3.3 Иначе:

3.3.1 Читаем файл с диска блоками.

3.3.2 Синхронизация данных файла. От каждого блока считаем хэш-код и сравниваем с вычисленным ранее. Если хэш-коды очередного блока не совпадают, сохраняем этот блок в БД. Лишние блоки удаляем, недостающие – сохраняем.

4 Для каждой папки из текущей директории:

4.1 Если в таблице Model нет соответствующей записи, то

4.1.1 Сохраняем информацию о данной папке в БД.

4.1.2 Обрабатываем данные в новой директории по рассмотренному выше алгоритму сохранения.

4.2 Иначе, выполняем аналогичные действия для данной директории.

Чтобы сократить количество запросов к БД при сканировании, можно хранить список хэш-кодов блоков данных на стороне клиента и поддерживать в актуальном состоянии, так же как модель данных сканируемой директории.

# Заключение

В статье были рассмотрены современные технологии резервного копирования данных. Далее был предложен подход к реализации системы резервного копирования с хранением файлов в базе данных. Одним из преимуществ такого подхода является его универсальность. Решение на его основе можно адаптировать под разные

схемы сети. Предлагаемый подход может служить основой для разработки сервиса удаленного резервного копирования, решения для локальной сети небольшой организации или распределенной сети крупной фирмы с филиалами. В статье были предложены варианты алгоритмов сохранения и восстановления резервной копии из БД, дана оценка их производительности и приведены результаты соответствующих экспериментов. Далее был предложен алгоритм поиска изменившихся данных. Также была показана зависимость скорости сохранения и восстановления данных от величины разбиения файлов.

Направлениями дальнейшего исследования являются доработка предложенных алгоритмов с применением технологии адаптивной дедубликации данных, экспериментальное исследование влияния данной технологии на скорость сохранения и восстановления, оценка границ применимости технологии адаптивной дедубликации.

## Список литературы / References

[1] Казаков В. Г., Федосин С. А., “Технологии и алгоритмы резервного копирования”, *Всероссийский конкурсный отбор обзорно-аналитических статей по приоритетному направлению «Информационно-телекоммуникационные системы»*, 2008, 1–49; [Kazakov V. G., Fedosin S. A., “Technologii i algoritmi reservnogo kopirovania”, *Vserossiyskiy konkursniy otbor obzorno-analiticheskih statey po prioritetnomu napravleniu ”Informacionno-telekommunikacionnie sistemi”*, 2008, 1–49, (in Russian).]

[2] Алиев А. А., Самедов Р. Б., “Алгоритм создания полного резервного копирования в облачных вычислениях”, *Вестник Бакинского Университета*, 2013, № 4, 120–127; English transl.: Aliev A. A., Samadov R. B., “An algorithm of a full backup in cloud computing”, *Baku State University News*, 2013, № 4, 120–127.

[3] Mugoh L., Ateya I. L., Shibwabo B. K, “Continuous Data Protection Architecture as a Strategy for Reduced Data Recovery Time”, *Journal of Systems Integration*, **2**:4 (2011), 54–69.

[4] Казаков В. Г., Федосин С. А., Плотникова Н. П., “Способ адаптивной дедупликации с применением многоуровневого индекса размещения копируемых блоков данных”, *Фундаментальные исследования*, 2013, № 8 (часть 6), 1322–1325; Kazakov V. G., Fedosin S. A., Plotnikova N. P., “Method of adaptive dedublication with multilevel block indexing”, *Fundamental research*, 2013, № 8 (part 6), 1322–1325.

[5] Kathuria V., Dhamankar R., Kodavalla H., “Transaction Isolation and Lazy Commit”, *IEEE 23rd International Conference on Data Enginering*, 2007, 1204–1211.

[6] Curtis Preston Mugoh W., “Data Protection Strategies In Today’s Data Center”, *Oracle White Paper*, 2012, 1–8.

[7] Zhu N., Chiueh T., “Portable and Efficient Continuous Data Protection for Network File Servers”, *Stony Brook University*, 2007, 1–17.

[8] Meyer D. T., Bolosky W. J., “A Study of Practical Deduplication”, *ACM Transactions on Storage*, **7**:4 (2012), 1–13.

[9] Storer M. W., Greenan K., Long D. D. E., Miller E. L., “Secure Data Deduplication”, *Proceedings of the 4th ACM international workshop on Storage security and survivability*, 2008, 1–10.

[10] Renzel K., Keller W., “Client/Server Architectures for Business Information Systems”, *A Pattern Language*, 1997, 1–25.

[11] Дейт К. Дж., *Введение в системы баз данных*, **8**, Издательский дом "Вильямс", 2005; Date C. J., *An Introduction to Database Systems*, **8**, Pearson Education, Inc., 2004.

[12] Гроff Д., Вайнберг П., Оппель Э., *SQL: полное руководство*, **3**, Издательский дом "Вильямс", 2015; Groff J., Weinberg P., Oppel A., *SQL The Complete Reference*, **3**, The McGraw-Hill Companies, 2010.

[13] Дейт К. Дж., *SQL и реляционная теория. Как грамотно писать код на SQL*, Символ-Плюс, 2010; Date C. J., *SQL and Relational Theory. How to Write Accurate SQL Code*, O'Reilly Media Inc., 2009.

[14] Sebastian J., Aelterman S., *The Art of SQL Server FILESTREAM*, Simple Talk Publishing, 2012.

[15] Sears R., Catharine van Ingen, Gray J., "To BLOB or Not To BLOB: Large Object Storage in a Database or a Filesystem?", *Technical Report MSR-TR-2006-45*, 2006, 1–11.

[16] Максимов В., Козленко Л. А., Маркин С. П., Бойченко И. А., "Защищенная реляционная СУБД Линтер", *Открытые системы. СУБД*, 1999, № 11–12; English transl.: Maksimov V., Kozlenko L. A., Markin C. P., Bojchenko I. A., "Zashchishchennaya relyacionnaya SUBD Linter", *Otkrytye sistemy. SUBD*, 1999, № 11–12, (in Russian).]

---

**Taranin S. M.**, "Backup with Storage in a Database", *Modeling and Analysis of Information Systems*, **23**:4 (2016), 479–491.

**DOI:** 10.18255/1818-1015-2016-4-479-491

**Abstract.** This paper presents an overview of some technologies that are used in modern backup systems. We consider their advantages and disadvantages. Next, we consider an example of the realisation of the backup system with files store in the database. We propose to divide the copied files into blocks of fixed length. Each block is a sequence of bytes. The block length may be adaptive, i.e. it can vary depending on the type or file size. We can store the file content in one table, and information of them such as names, attributes, and relationships between them, store in another table. The information of retained files and folders can be stored also on the client side in a hierarchical structure. It is a set of records and a model of the copied directory. The presence of such a model allows to find changes of the copied directory without additional queries to the database. If a file is modified, it is copied only the changed blocks. The model is also updated on the client side. Thus, the load on the data channel reduces. This paper presents the algorithms of saving and restoring data, and describes the factors that affect to the speed of their work. It demonstrates the dependence of the rate of saving and recovery of the fineness of the partition files, as well as the structure of the copied directory.

**Keywords:** file, data, backup, database, block, model

**On the authors:**
Taranin Sergey Maksimovich, orcid.org/0000-0001-8117-7358, graduate student,
P.G. Demidov Yaroslavl State University,
Sovetskaya str., 14, Yaroslavl, 150003, Russia, e-mail: staranin0208@yandex.ru