МИНОБРНАУКИ РОССИИ

Федеральное государственное автономное образовательное
учреждение высшего образования
«Южный федеральный университет»

Институт математики, механики
и компьютерных наук им. И. И. Воровича

Кафедра алгебры и дискретной математики

Михайлишин Андрей Сергеевич

РАСПРЕДЕЛЕННАЯ ОТКАЗОУСТОЙЧАЯ СИСТЕМА ХРАНЕНИЯ ДАННЫХ НА ОСНОВЕ СИСТЕМЫ ОСТАТОЧНЫХ КЛАССОВ

КУРСОВАЯ РАБОТА
по направлению подготовки
01.03.02 – Прикладная математика и информатика

Научный руководитель –
доцент, к.т.н. Могилевская Надежда Сергеевна

Отлично (100 баллов) _____
оценка (рейтинг) подпись руководителя

Ростов-на-Дону – 2023

Задание к курсовой работе

«РАСПРЕДЕЛЕННАЯ ОТКАЗОУСТОЙЧИВАЯ СИСТЕМА ХРАНЕНИЯ ДАННЫХ НА ОСНОВЕ СИСТЕМЫ ОСТАТОЧНЫХ КЛАССОВ»

студента факультета математики, механики и компьютерных наук Южного федерального университета

Михайлишина Андрея Сергеевича

В курсовой работе необходимо:

создать распределенную отказоустойчивую систему хранения данных на основе системы остаточных классов (СОК). Построить программное средство (имитационную модель на языке программирования C++), реализующую метод разделения данных, провести эксперименты с реализацией метода.

Отзыв научного руководителя
на курсовую работу «Распределенная отказоустойчивая система хранения данных на основе системы остаточных классов»
Михайлишина Андрея Сергеевича

В ходе выполнения курсовой работы Андрей Михайлишин работал регулярно, согласно предварительно составленному плану. Проявлял разумную инициативу, продемонстрировал отличные навыки программирования, умение читать и понимать научные тексты. Цель курсовой работы достигнута в полном объеме, намечены дальнейшие направления работы.

Рекомендуемая оценка **отлично** (100 баллов).

Руководитель
Могилевская Надежда Сергеевна

АНТИПЛАГИАТ
ОБНАРУЖЕНИЕ ЗАИМСТВОВАНИЙ

СПРАВКА

Южный Федеральный Университет

о результатах проверки текстового документа
на наличие заимствований

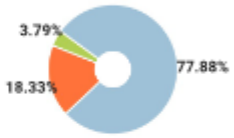
ПРОВЕРКА ВЫПОЛНЕНА В СИСТЕМЕ АНТИПЛАГИАТ.ВУЗ

Автор работы: Михайлишин Андрей Сергеевич
Самоцитирование рассчитано для: Михайлишин Андрей Сергеевич
Название работы: КУРСОВАЯ РАБОТА
Тип работы: Курсовая работа
Подразделение: Институт механики, математики и компьютерных наук ЮФУ

РЕЗУЛЬТАТЫ

СОВПАДЕНИЯ	18.33%
ОРИГИНАЛЬНОСТЬ	77.88%
ЦИТИРОВАНИЯ	3.79%
САМОЦИТИРОВАНИЯ	0%

ДАТА ПОСЛЕДНЕЙ ПРОВЕРКИ: 02.06.2023



Структура документа: Проверенные разделы: титульный лист с.1, основная часть с.2-19, библиография с.20

Модули поиска: ИПС Адилет; Библиография; Сводная коллекция ЭБС; Интернет Плюс*; Сводная коллекция РГБ; Цитирование; Переводные заимствования (RuEn); Переводные заимствования по eLIBRARY.RU (EnRu); Переводные заимствования по коллекции Гарант: аналитика; Переводные заимствования по коллекции Интернет в английском сегменте; Переводные заимствования по Интернету (EnRu); Переводные заимствования по коллекции Интернет в русском сегменте; Переводные заимствования издательства Wiley ; eLIBRARY.RU; СПС ГАРАНТ: аналитика; СПС ГАРАНТ: нормативно-правовая документация; Медицина; Диссертации НББ; Коллекция НБУ; Перефразирования по eLIBRARY.RU; Перефразирования по СПС ГАРАНТ: аналитика; Перефразирования по Интернету; Перефразирования по Интернету (EN); Перефразированные заимствования по коллекции Интернет в английском сегменте; Перефразированные заимствования по коллекции Интернет в русском сегменте; Перефразирования по коллекции

Работу проверил: Могилевская Надежда Сергеевна

ФИО проверяющего

Дата подписи: 29.05.2023

Подпись проверяющего

Чтобы убедиться
в подлинности справки, используйте QR-код,
который содержит ссылку на отчет.

Ответ на вопрос, является ли обнаруженное заимствование корректным, система оставляет на усмотрение проверяющего. Предоставленная информация не подлежит использованию в коммерческих целях.

Оглавление

Введение

Одним из способов организации отказоустойчивых хранилищ является разделение данных на множество частей с дополнительной избыточностью, затем восстановление всех данных из неполного набора частей.

В этой работе рассматривается такой способ – избыточная система остаточных классов (ИСОК). Важнейшая особенность этого метода – возможность контроля целостности информации. Целостность проверяется восстановлением закодированного в остатках изначального числа и проверки на то, попадает ли это число в допустимый диапазон. Если ошибка обнаруживается, то существуют различные алгоритмы способные локализовать искажение, если оно произошло не более чем в определенном заранее количестве остатков.

Цель курсовой работы

Создание распределенной отказоустойчивой системы хранения данных на основе системы остаточных классов (СОК). Для этого подойдет усовершенствованный метод – избыточная система остаточных классов (ИСОК).

Для достижения цели необходимо построить программное средство (имитационную модель на языке программирования C++), реализующую метод разделения данных, и провести эксперименты с реализацией метода.

Структура работы

в разделе 1 по материалам [1] описан метод СОК и его производное - ИСОК;

в разделе 2 описана реализация в C++

в разделе 3 описаны проведенные эксперименты и их результаты

1. Метод ИСОК

1.1. Базовые определения

В СОК каждое число A представляется в виде набора из k остатков от деления a_i этого числа на p_i, входящие в набор оснований:

$$A = (a_1, a_2, \ldots, a_k),\ a_i = A\ mod\ p_i,\ \text{где } i = 1, \ldots, k.$$

Согласно китайской теореме об остатках [теория чисел учебник] (КТО) такое представление A из промежутка $[0, P_k)$ уникально лишь в том случае, если все p_i попарно простые. P_k - рабочий диапазон представления чисел в СОК. Добавив к системе оснований $\{p_1, p_2, \ldots, p_k\}$ основания $p_{k+1}, \ldots, p_n$ и расширив представление A остатками этих оснований $a_{k+1}, \ldots, a_n$ получаем избыточную СОК, которая приобретает новые свойства, которые требуются для создания отказоустойчивой системы хранения данных. А именно, потеря любых $n - k$ остатков не лишает возможности восстановить исходное число A.

На рисунке 1 сведены вместе введенные выше определения и обозначения (рис. 1).

Введенные определения

СОК- система остаточных классов;
ИСОК - избыточная система остаточных классов;
КТО - китайская теорема об остатках;
k – число рабочих оснований;
n – число всех оснований (рабочие + избыточные);
$P_k = p_1 \cdot p_2 \cdot \ldots \cdot p_k$ – рабочий диапазон (объем диапазона системы);
$P = p_1 \cdot p_2 \cdot \ldots \cdot p_n$ – полный диапазон;
$A \in [0, P_k)$ – число (кодируемая информация);
$P: \{p_1, p_2, \ldots, p_n\}$ – система оснований;
$(a_1, a_2, \ldots, a_n)$ - остатки по набору оснований;

Рис. 1. Введенные определения в 1.1.

1.2. Подбор оснований для представления числа A в ИСОК

Пусть A - имеет разрядность b бит**,** тогда для корректного представления в СОК рабочий диапазон должен быть больше или равен 2^b:

$$P_k \geq 2^b$$

или, другими словами,

$$p_1 \cdot p_2 \cdot \ldots \cdot p_k \geq 2^b.$$

Чем меньше разрядность каждого основания, тем меньше аппаратных затрат для реализации операций по данному основанию. Также сократить аппаратные затраты можно за счет использования модуля, равного степени двойки 2^l, так как для нахождения остатка по данному модулю достаточно взять l младших разрядов исходного числа.

Таким образом необходимо подобрать набор k взаимно простых чисел и минимально возможной разрядностью, достаточных для перекрытия диапазона $[0, 2^b)$.

Сам алгоритм подбора набора оснований P можно разделить на 2 типа - набор без степени двойки и со степенью двойки - при выборе набора со степенью двойки операции с одним из основанием сильно упрощаются для вычисления, что положительно влияет на скорость работы программы.

В первом случае берется округление сверху от $(2^{b/k}) - 1$ за условное среднее значение рабочих оснований, затем заполняются все остальные рабочие основания (первые k) по принципу первое попарно простое для всех уже выбранных оснований слева в p[k/2 - i], первое попарно простое для всех уже выбранных оснований справа в p[k/2 + i]. Далее весь массив P структурируется для того, чтобы все рабочие основания оказались по порядку в начале массива. Далее заполняются избыточные основания - от k

до n, берутся первые попарно простые (для всех уже выбранных оснований) числа справа. Получаем первый набор оснований - $\{p_i\}_{i=1}^{n}$.

Для второго набора оснований $\{p_i^*\}_{i=1}^{n}$, где степень двойки является обязательной сначала считается количество занимаемых бит для самого большой основания из первого набора $b(p_n)$, затем условному среднему рабочих оснований присваивается значение $2^{b(p_n)-1}$ и повторяются те же шаги что и при нахождении первого набора оснований.

Если разрядности наибольших элементов наборов совпадает $b(p_n) = b(p_n^*)$, тогда используем диапазон со степенью двойки $\{p_i^*\}_{i=1}^{n}$, иначе: используем первый набор оснований $\{p_i\}_{i=1}^{n}$. На рисунке 2 сведены вместе введенные выше определения и обозначения (рис. 2).

Введенные определения

b – требуемая разрядность числа A (число бит)

P: $\{p_i\}_{i=1}^{n}$, $\{p_i^*\}_{i=1}^{n}$ – наборы оснований без степени двойки и со степенью двойки

p_n – последнее основание в наборе (самое большое т.к. P структурирован по размеру)

$b(p_n)$ – число занимаемых бит основанием p_n

Рис. 2. Введенные определения в 1.2.

1.3. Преобразование числа A в ИСОК

Разбиение A на остатки наборов оснований методом деления является довольно неэффективным из-за сложности операции. Поэтому используется метод непосредственного суммирования с использованием табличной арифметики для числа A.

Пусть число A записано в позиционной системе счисления с основанием N, то есть:

$A = A_0 \cdot N^0 + \ldots + A_m \cdot N^m$ или $A = \sum_{i=0}^{m} A_i N^i$, где $0 \leq A \leq N - 1$.

Представим степени основания N^i и коэффициенты A_i в системе остаточных классов с основаниями $\{p_1, p_2, \ldots, p_n\}$, тогда:

$$N^i = \left\{N_1^{(i)}, \ldots, N_n^{(i)}\right\}, A_i = \left\{A_1^{(i)}, \ldots, A_n^{(i)}\right\}$$

получим:

$$A = \left(\sum_{i=0}^{m} A_i^{(1)} N_i^{(1)} \bmod p_1, \ldots, \sum_{i=0}^{m} A_i^{(n)} N_i^{(n)} \bmod p_n\right) = (a_1, \ldots, a_n)$$

Пример. Переведем число $A = 1446$ в СОК с основаниями

$$\{p_1, p_2, \ldots, p_6\} = \{4, 5, 7, 9, 11, 13\}.$$

A представимо в виде:

$$1446 = 1 \cdot 10^3 + 4 \cdot 10^2 + 4 \cdot 10^1 + 6 \cdot 10^0$$

В таблице 1 представлены остатков степеней $10^i, i = 0..4$ по модулям СОК.

Таблица 1. Остатки степеней $10^i, i = 0..4$ по модулям СОК.

	$p_1 = 4$	$p_2 = 5$	$p_3 = 7$	$p_4 = 9$	$p_5 = 11$	$p_6 = 13$
10^0	1	1	1	1	1	1
10^1	2	0	3	1	10	10
10^2	0	0	2	1	1	9
10^3	0	0	6	1	10	12

Составим по модулям СОК таблицу остатков коэффициентов A^i (табл. 2), где $A = 0..9,\ i = 0..4$:

$$A = 1446 = 1 \cdot 10^3 + 4 \cdot 10^2 + 4 \cdot 10^1 + 6 \cdot 10^0$$

тогда

$$A \bmod 4 = (1 \cdot 0 + 0 \cdot 0 + 0 \cdot 2 + 2 \cdot 1) \bmod 4 = 2$$

$$A \bmod 5 = (1 \cdot 0 + 4 \cdot 0 + 4 \cdot 0 + 1 \cdot 1) \bmod 5 = 1$$
$$A \bmod 7 = (1 \cdot 6 + 4 \cdot 2 + 4 \cdot 3 + 6 \cdot 1) \bmod 7 = 4$$
$$A \bmod 9 = (1 \cdot 1 + 4 \cdot 1 + 4 \cdot 1 + 6 \cdot 1) \bmod 9 = 6$$
$$A \bmod 11 = (1 \cdot 10 + 4 \cdot 1 + 4 \cdot 10 + 6 \cdot 1) \bmod 11 = 5$$
$$A \bmod 13 = (1 \cdot 12 + 4 \cdot 9 + 4 \cdot 10 + 6 \cdot 1) \bmod 13 = 3$$

то есть

$$A = (2, 1, 4, 6, 5, 3).$$

Таблица 2. Остатки *коэффициентов* A^i по модулям СОК.

	$p_1 = 4$	$p_2 = 5$	$p_3 = 7$	$p_4 = 9$	$p_5 = 11$	$p_6 = 13$
0	0	0	0	0	0	0
1	1	1	1	1	1	1
2	2	2	2	2	2	2
3	3	3	3	3	3	3
4	0	4	4	4	4	4
5	1	0	5	5	5	5
6	2	1	6	6	6	6
7	3	2	0	7	7	7
8	0	3	1	8	8	8
9	1	4	2	0	9	9

Конец примера.

На рисунке 3 сведены вместе введенные выше определения и обозначения (рис. 3).

Введенные определения

$A = A_0 N^0 + \ldots + A_m N^m$ – изначальное число где

N – основание позиционной системы счисления

A_i – коэффициент перед N^i

$N^i = \left\{N_1^{(i)}, \ldots, N_n^{(i)}\right\}$ – представленное в СОК основание степени i

$A_i = \left\{A_1^{(i)}, \ldots, A_n^{(i)}\right\}$ – представленный в СОК i-тый коэффициент

Рис. 3. Введенные определения в 1.3.

1.4. Восстановление числа A методом ортогональных базисов

Для восстановления изначального числа A используется метод ортогональных базисов [1], в его основе лежит КТО (китайская теорема об остатках).

Пусть $\{p_1, p_2, \ldots, p_k\}$ – система оснований, $P_k = p_1 \cdot p_2 \cdot \ldots \cdot p_k$ – объем диапазона системы. Тогда перевод из набора остатков $(a_1, a_2, \ldots, a_k)$ в число A осуществляется по формуле:

$$A = \left|\sum_{i=1}^{k} a_i \cdot B_i^{(k)}\right|_{P_k},$$

где

$$B_i^{(k)} = \frac{p_i}{P_k}\left(\frac{p_i}{P_k} mod\ p_i\right).$$

Так как система оснований заранее задана, то ортогональные базисы $B_i^{(k)}$ могут быть заранее вычислены, причем единожды.

На рисунке 4 сведены вместе введенные выше определения и обозначения (рис. 4).

Введенные определения

$B^{(k)} = \left\{B_1^{(k)}, \ldots, B_k^{(k)}\right\}$ – ортогональные базисы для системы оснований $\{p_1, p_2, \ldots, p_k\}$ (рабочей)

$B = \left\{B_1, \ldots, B_n\right\}$ – ортогональные базисы для полной системы оснований $\{p_1, p_2, \ldots, p_n\}$ (где $B_i = \frac{P}{p_i}\left(\frac{p_i}{P} mod\ p_i\right)$)

Рис. 4. Введенные определения в 1.4.

1.5. Локализация и исправление искажений

Для исправления ошибок используется метод исправления ошибок, основанный на методе проекций с максимальным правдоподобием [1].

Пусть дана ИСОК $\{p_1, p_2, \ldots, p_n\}$ с рабочими основаниями $\{p_1, p_2, \ldots, p_k\}$. $P_k = p_1 \cdot\ p_2 \cdot \ldots \cdot p_k$ – рабочий диапазон СОК, $P = p_1 \cdot\ p_2 \cdot \ldots \cdot p_k \cdot\ p_{k+1} \cdot \ldots \cdot p_n$ – полный диапазон ИСОК. Число A представлено в ИСОК остатками $(a_1, a_2, \ldots a_k, a_{k+1}, \ldots, a_n)$. Максимальное количество ошибок, которые способна исправить ИСОК не превышает половины количества контрольных значений $q = \left\lfloor \frac{n-k}{2} \right\rfloor$.

Для обнаружения ошибки найдем позиционное представление искаженного числа $A^* = (a_1^*, a_2^*, \ldots, a_n^*)$: система оснований $\{p_1, p_2, \ldots, p_n\}$ имеет ортогональные базисы $B = \{B_1, \ldots, B_n\}$ где $B_i = \frac{P}{p_i}\left(\frac{p_i}{P} mod\ p_i\right)$. При попытке восстановления $A^* = \left|\sum_{i=1}^{k} a_i^* \cdot B_i\right|_P$ определяется его искажение тем, что восстановленное A^* оказывается больше рабочего диапазона P_k.

Для локализации необходимо вычислить различные проекции и сравнить их расстояние Хемминга с A^*.

Для проекции A^{Δ} используется СОК = $\{p_{g(1)}, p_{g(2)}, \ldots, p_{g(k)}\}$ - группа из различных k оснований ($P_{\Delta} = p_{g(1)} \cdot \ldots \cdot p_{g(k)}$– диапазон этой СОК),

содержащихся в $\{p_1, p_2, \ldots, p_n\}$. Далее находится ортогональный базис $B^{\Delta} = \{B_1^{\Delta}, \ldots, B_k^{\Delta}\}$ и делается попытка восстановления:

$$A^{\Delta} = \left|\sum_{i=1}^{k} a_{g(i)}^{*} \cdot B_i\right|_{P_{\Delta}}.$$

Если $A^{\Delta} < P_k$, то тогда A^{Δ} преобразуется в набор остатков $(a_1^{\Delta}, a_2^{\Delta}, \ldots, a_n^{\Delta})$. Расстояние Хемминга измеряется количеством отличий между наборами остатков $(a_1^{\Delta}, a_2^{\Delta}, \ldots, a_n^{\Delta})$ и $(a_1^{*}, a_2^{*}, \ldots, a_n^{*})$, если оно не превышает q тогда A^{Δ} объявляется исправленным числом, а все отличающиеся a_i^{*} по основаниям p_i – локализованной ошибкой.

На рисунке 5 сведены вместе введенные выше определения и обозначения (рис. 5).

Введенные определения

$q = \left\lfloor \frac{n-k}{2} \right\rfloor$ - максимальное количество ошибок, которое может исправить метод

$A^{*} = (a_1^{*}, a_2^{*}, \ldots, a_n^{*})$ – искаженное число

$A^{\Delta} = (a_1^{\Delta}, a_2^{\Delta}, \ldots, a_n^{\Delta})$– проекция, восстановленная по набору оснований

$P^{\Delta} = \{p_{g(1)}, p_{g(2)}, \ldots, p_{g(k)}\}$ и их ортогональному базису $B^{\Delta} = \left\{B_1^{\Delta}, \ldots, B_k^{\Delta}\right\}$

$(P_{\Delta} = p_{g(1)} \cdot \ldots \cdot p_{g(k)}$ – её диапазон)

$g(x)$: $\aleph[1; k] \rightarrow \aleph[1; n]$ – функция выбора оснований для A^{Δ}.

$d\left((a_1^{\Delta}, a_2^{\Delta}, \ldots, a_n^{\Delta}), (a_1^{*}, a_2^{*}, \ldots, a_n^{*})\right)$ – расстояние Хемминга

Рис. 5. Введенные определения в 1.5.

2. Программная реализация

Реализация выполнена на языке C++ в компиляторе Clion, разбита на определенные функции, каждая выполняет свою задачу. Цель реализации была воссоздать метод ИСОК используя стандартные библиотеки компилятора.

Все алгоритмы, описанные в 1й части, разбиты на задачи и выделены в функции.

Для работы с длинными числами была использована открытая библиотека LongInt [3]. Алгоритмы преобразования числа адаптированы под эту библиотеку.

Для имитации разнесенных ячеек памяти используется класс vector<LongInt>, для имитации “неисправностей” этих ячеек памяти и проверки исправности алгоритма восстановления числа был создан псевдослучайный генератор ошибок.

Также в каждой функции для отладки и удобного просмотра работы алгоритмов закомментированы в нужных местах выводы.

Для удобного просмотра работы алгоритмов они вызываются друг за другом и выводят результаты своей работы в командую строку.

3. Экспериментальные исследования метода

Разберем работу программы на двух примерах работы на примере с небольшими числами, которые можно легко проверить «на бумаге» (фрагмент работы алгоритма с заданными данными был разобран в примере в 1.3.) (пусть $b = 4;\ k = 2;\ n = 6;\ A = 1446;$) и с относительно большими числами (число $A > 64$ бит) (пусть $b = 80;\quad k = 8;\quad n = 16;$ $A = 123456789000987654321;$)

Генератор ключей выдаст следующий результат (листинги 1 и 2):

Листинг 1. Фрагмент вывода с результатами генерации набора оснований $(b = 4; k = 2; n = 6; A = 1446;)$

```
// Создание набора оснований P //

P1 = [4, 5, 7, 9, 11, 13]
P2 = [4, 5, 7, 9, 11, 13]
P = [4, 5, 7, 9, 11, 13]
```

Листинг 2. Фрагмент вывода с результатами генерации набора оснований $(b = 80; k = 8; n = 16; A = 12345678900098765432 1;)$

```
// Создание набора оснований P //

P1 = [1021, 1023, 1024, 1025, 1027, 1031, 1033, 1037, 1039,
1043, 1049, 1051, 1061, 1063, 1069, 1073]
P2 = [512, 2045, 2047, 2049, 2051, 2053, 2057, 2059, 2063,
2069, 2071, 2077, 2081, 2083, 2087, 2089]
P = [1021, 1023, 1024, 1025, 1027, 1031, 1033, 1037, 1039,
1043, 1049, 1051, 1061, 1063, 1069, 1073]
```

Здесь мы видим, что при небольших вводных параметрах 1й и 2й набор оснований совпал, а при больших вводных уже полностью отличается.

Произведем преобразование числа A, результаты на листинге 3 и 4 подписаны как Clear Result (чистое A, которое еще не было “испорчено” генератором ошибок):

Листинг 3. Фрагмент вывода с результатами преобразования числа $(b = 4; k = 2; n = 6; A = 1446;)$

```
// Кодировка A //

A = 1446
Clear Result = [2, 1, 6, 6, 6, 6]
```

Так как число A выходит за изначально заданный предел (4 бита), берутся только первые его 4 бита ($1446\ mod\ 16 = 6$).

Листинг 4. Фрагмент вывода с результатами преобразования числа $(b = 80; k = 8; n = 16; A = 12345678900098765432 1;)$

```
// Кодировка A //

A = 12345678900098765721
Clear Result = [699, 720, 921, 696, 279, 631, 73, 875, 265,
591, 595, 741, 591, 1050, 803, 633]
```

Далее необходимо симулировать ошибки в хранении разбитого на остатки числа A, для этого используется генератор ошибок, который выводит сколько ошибок и где именно он их сделал (листинги 5, 6).

Листинг 5. Фрагмент вывода с результатами работы генератора ошибок $(b = 4; k = 2; n = 6; A = 1446;)$

```
// Генерация ошибок в кодировке //

-----------------------------------------------
Errors: 2
Error_Is_Here = [0, 0, 1, 1, 0, 0]
-----------------------------------------------
Corrupt Result = [2, 1, 4, 4, 6, 6]
```

Листинг 6. Фрагмент вывода с результатами работы генератора ошибок $(b = 80; k = 8; n = 16; A = 12345678900098765432 1;)$

```
// Генерация ошибок в кодировке //

-----------------------------------------------
Errors: 1
Error_Is_Here = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0,
0]
-----------------------------------------------
Corrupt Result = [699, 720, 921, 696, 279, 631, 73, 875, 265,
591, 595, 741, 591, 1045, 803, 633]
```

Локализация искажений будет производится с помощью функции выбора оснований для проекций, предложенной в материалах [1], она обладает высокой скоростью вследствие ограниченного перебора набора остатков. Работает она следующий образом: если мы имеем (k, n) избыточную СОК и $k < \lfloor (n-k)/2 \rfloor$ тогда проекции берутся следующим образом:

$$\text{СОК}_1 = a_1, \dots, a_k, \cancel{a_{k+1}, \dots, a_{2k}, \dots, a_{\lfloor \frac{n}{k} \rfloor k-k+1}, a_{\lfloor \frac{n}{k} \rfloor k-k+2}, \dots, a_{\lfloor \frac{n}{k} \rfloor k}, \dots, a_n},$$

$$\text{СОК}_2 = \cancel{a_1, \dots, a_k}, a_{k+1}, \dots, a_{2k}, \cancel{\dots, a_{\lfloor \frac{n}{k} \rfloor k-k+1}, a_{\lfloor \frac{n}{k} \rfloor k-k+2}, \dots, a_{\lfloor \frac{n}{k} \rfloor k}, \dots, a_n},$$

…

$$\text{СОК}_{\lfloor \frac{n}{k} \rfloor} = \cancel{a_1, \dots, a_k, a_{k+1}, \dots, a_{2k}, \dots, a_{\lfloor \frac{n}{k} \rfloor k-k+1}}, a_{\lfloor \frac{n}{k} \rfloor k-k+2}, \dots, a_{\lfloor \frac{n}{k} \rfloor k}, \cancel{\dots, a_n},$$

Такая выборка имеет и свои недостатки:

- Не использование оснований $a_{\lfloor \frac{n}{k} \rfloor k+1}, \dots a_n$, это решается тем, что изначально берется n кратное k.
- При больших n и k фактическая вероятность восстановить число при заданных методом количестве возможных ошибок q уменьшается, так как в каждой выборке с большой вероятностью окажется ошибка.

Для примера посчитаем вероятность нахождения таким методом хотя бы одной «целой» проекции при разном количество ошибок, пусть $k = 4$; $n = 24$; подсчет вероятности будет вестись экспериментально с помощью приложения Maple, результат на графике 1.

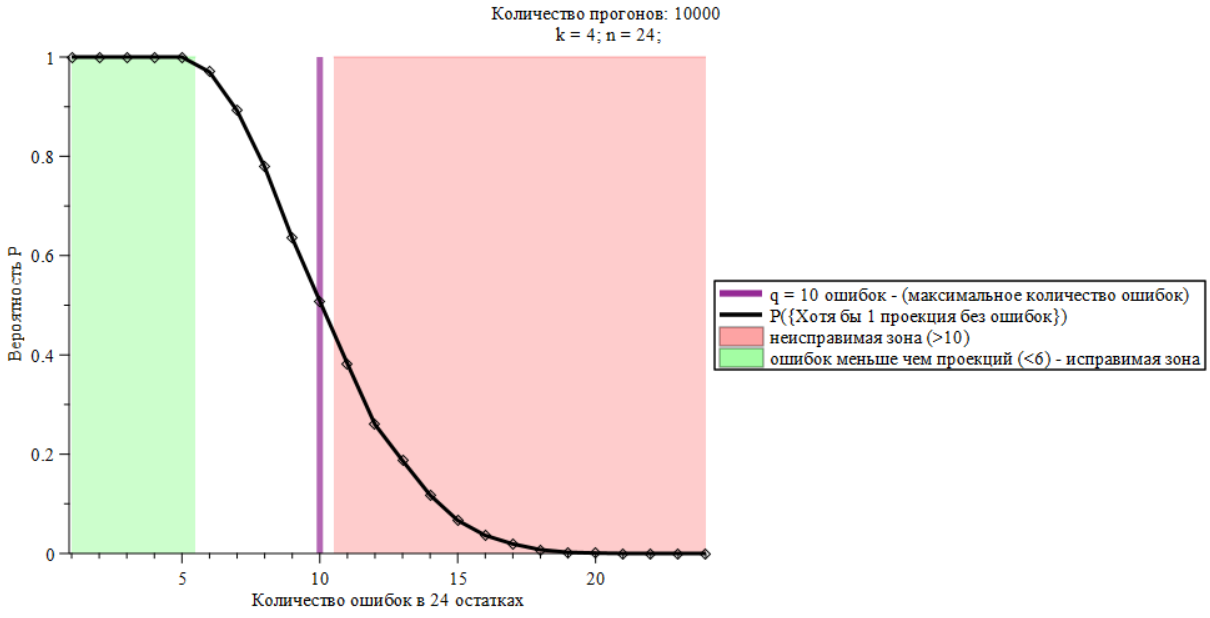


График 1. Вероятность нахождения хотя бы одной проекции без ошибок

Из результатов можно сделать вывод что такой метод эффективен со 100% вероятностью только до тех пор, пока ошибок появляется меньше, чем число проекций.

Для ускорения работы программы я буду использовать такой перебор, но при этом подразумевается, что восстановление ошибок можно улучшить более подробным перебором набора остатков. Также в своем примере с большим числом A вместо 4 ошибок я буду находить только 1 ошибку для того, чтобы гарантированно возникала ситуация, в которой программа сможет локализовать ошибку.

Для восстановления числа нам потребуются ортогональные системы, результат их генерации показан на листинге 7:

Листинг 7. Фрагмент вывода с результатами подсчета ортогональной системы ($b = 4; k = 2; n = 6; A = 1446;$)

```
// Создание ортогональной системы для любого набора ключей //

B         = [45045, 36036, 25740, 140140, 16380, 97020]
B[ 1.. 2] = [5, 16, 0, 0, 0, 0]
B[ 3.. 4] = [0, 0, 36, 28, 0, 0]
B[ 5.. 6] = [0, 0, 0, 0, 78, 66]
```

Для большого A ортогональная система принимает неразборчивый вид, для примера:

$$B_1 = 771783758616969303035487461090858245515249408000.$$

Для восстановления A для наглядности используются «чистое» и «искаженное» рядом, результаты показаны на листингах 8 и 9:

Листинг 8. Фрагмент вывода с результатами восстановления числа A ($b = 4; k = 2; n = 6; A = 1446;$)

```
// Декодировка A //
Clear A           = 6
Corrupt A         = 28606
```

Листинг 9. Фрагмент вывода с результатами восстановления числа A $(b = 80; k = 8; n = 16; A = 12345678900098765721;)$

```
// Декодировка A //
Clear A           = 12345678900098765721
Corrupt A         =
157117309053793313252629055909593994151432798812l
```

Восстановление A и локализация ошибок является самой последней частью, алгоритм работы основывается на результатах работы алгоритмов, описанных выше. Результаты показаны на листингах 10 и 11:

Листинг 10. Фрагмент вывода с результатами исправления ошибок в A $(b = 4; k = 2; n = 6; A = 1446;)$

```
Keys: P[0..6] with corrupt A
=================================================
P = [4, 5] A = 6 = d(CorruptA, CurrentA) = (2 <= 2) = 1 ,
(q=2)
P = [7, 9] A = 4 = d(CorruptA, CurrentA) = (4 <= 2) = 0 ,
(q=2)
P = [11, 13] A = 6 = d(CorruptA, CurrentA) = (2 <= 2) = 1 ,
(q=2)
-------------------------------------------------
A исправлена успешно
 A = 6 = [2, 1, 6, 6, 6, 6]
Ошибка в 3м фрагменте
Ошибка в 4м фрагменте
d(CorruptA, CurrentA) = (2 <= 2) = 1 , (q=2)
```

Листинг 11. Фрагмент вывода с результатами исправления ошибок в A $(b = 80; k = 8; n = 16; A = 12345678900098765721;)$

```
// Восстановление A и вывод ошибок //
Keys: P[0..16] with corrupt A
=================================================
P = [1021, 1023, 1024, 1025, 1027, 1031, 1033, 1037] A =
12345678900098765721 = d(CorruptA, CurrentA) = (1 <= 4) = 1 ,
(q=4)
P = [1039, 1043, 1049, 1051, 1061, 1063, 1069, 1073] A =
14860224657507460519979631 = d(CorruptA, CurrentA) = (16 <= 4)
= 0 , (q=4)
-------------------------------------------------
A исправлена успешно
 A = 12345678900098765721 = [699, 720, 921, 696, 279, 631, 73,
875, 265, 591, 595, 741, 591, 1050, 803, 633]
Ошибка в 14м фрагменте
d(CorruptA, CurrentA) = (1 <= 4) = 1 , (q=4)
```

Заключение

В курсовой работе создана имитационная модель распределенной отказоустойчивой системы хранения данных на основе системы избыточных остаточных классов. Модель реализована в виде программного средства, построенного с применением языка программирования C++. В ходе проведения экспериментов были обнаружены особенности выбора функции выборки проекций, использование меньшей выборки позволяет удешевить процесс исправления ошибок, но влияет на вероятность восстановления числа при большом количестве ошибок.

К дальнейшим направлениям работы относится улучшение программы с точки зрения программирования, а именно: добавление генерации всех таких константных значений, как таблицы остатков по основаниям или ортогональные базисы для различных наборов оснований; перевод большинство алгоритмов в многопоточную работу; хранение заранее вычисленных констант в отдельных файлах; перевод программы с хранения абстрактного числа на кодирование и декодирование файлов; общая оптимизация программной реализации алгоритмов.

Также к дальнейшим направлениям работы относится сравнение данного метода разделения данных с другими, оценка преимуществ метода над другими, выявление недостатков и/или ограничений.

Литература

1. Разработка методов и алгоритмов построения отказоустойчивых распределенных систем хранения данных на основе модулярной арифметики,: дис. Назаров А.С. канд. техн. наук, Ставрополь, 2019.
2. Пилиди В.С. Математические основы защиты информации. Ростов-на-Дону, изд-во ЮФУ, 2014
3. Работа с очень длинными числами на C++ – URL: https://habr.com/ru/post/578718/