Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет
имени Н. Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н. Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

ОТЧЕТ

по лабораторной работе № 5

по курсу «Анализ алгоритмов»

на тему: «Организация асинхронного взаимодействия потоков вычисления на примере конвейерных вычислений»

Студент ИУ7-56Б (Группа) _____ (Подпись, дата) М. Ю. Вольняга (И. О. Фамилия)

Преподаватель _____ (Подпись, дата) Л. Л. Волкова (И. О. Фамилия)

2024 г.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Разработчики архитектуры компьютеров издавна прибегали к методам проектирования, известным под общим названием «совмещение операций», при котором аппаратура компьютера в любой момент времени выполняет одновременно более одной базовой операции [1].

Этот метод включает в себя, в частности, такое понятие, как конвейеризация. Конвейеры широко применяются программистами для решения трудоемких задач, которые можно разделить на этапы, а также в большинстве современных быстродействующих процессоров [1].

В качестве операций, выполняющихся на конвейере в данной работе, взяты следующие:

1) считывание из файлов *TF-IDF*;
2) кластеризация;
3) запись результата в файл.

Целью данной лабораторной работы является описание параллельных конвейерных вычислений на основе нативных потоков для иерархической кластеризации документов.

В рамках выполнения работы необходимо решить следующие задачи:

1) описать организацию конвейерной обработки данных;
2) описать алгоритмы иерархической кластеризации, которые будут использоваться в данной лабораторной работе;
3) спроектировать программное обеспечение, позволяющее выполнять конвейерную и последовательную обработку данных;
4) разработать программное обеспечение, позволяющее выполнять конвейерную и последовательную обработку данных;
5) провести сравнительный анализ времени работы конвейерной и последовательной версии.

1 Аналитический раздел

В данном разделе будут рассмотрены конвейерная обработка данных, алгоритмы классификации полнотекстовых документов, алгоритмы классификации без учителя, иерархические алгоритмы, алгоритм дивизимной иерархической кластеризации, алгоритм дивизимной иерархической кластеризации и алгоритм k-средних.

1.1 Конвейерная обработка данных

Конвейер — организация вычислений, при которой увеличивается количество выполняемых инструкций за единицу времени за счет использования принципов параллельности.

Конвейеризация в компьютерной обработке данных основана на разбиении выполнения функций на более мелкие этапы, называемые ступенями, и выделении отдельной аппаратуры для каждой из них. Это позволяет организовать передачу данных от одного этапа к следующему, что увеличивает производительность за счет одновременного выполнения нескольких команд на различных ступенях конвейера [1].

Хотя конвейеризация не уменьшает время выполнения отдельной команды, она повышает пропускную способность процессора, что приводит к более быстрому выполнению программы по сравнению с простой, не конвейерной схемой.

1.2 Алгоритмы классификации полнотекстовых документов

Классификация текстов — ключевая задача в компьютерной лингвистике, охватывающая алгоритмы с учителем и без учителя, и имеющая важное значение для обеспечения информационной безопасности. Алгоритмы с учителем используют предварительно размеченные данные для обучения, в то время как алгоритмы без учителя, такие как кластеризация, организуют данные на основе внутренних закономерностей [2].

В данной лабораторной работе исследуется применение алгоритмов классификации без учителя для определения групп документов с помощью метода иерархической кластеризации с дивизимным подходом. Целью является выяв-

ление кластеров документов, таким образом, чтобы документы внутри одного кластера были максимально схожи по смыслу, а документы из различных кластеров — значительно отличались. Особенность данного подхода заключается в отсутствии необходимости в предварительной разметке данных и определении количества кластеров, что открывает широкие возможности для анализа неструктурированных данных [2]. Кластеризация — это разбиение элементов некоторого множества на группы по принципу схожести. Эти группы принято называть кластерами [3].

1.3 Алгоритмы классификации без учителя

Алгоритмы классификации без учителя разбивают набор документов на группы, где одна группа содержит родственные документы, а разные группы содержат разные документы. Без обучающего подмножества и известных категорий, алгоритм кластеризации автоматически определяет количество и состав кластеров, используя расстояния между документами [2].

Кластеризация текстов основана на идее, что похожие документы подходят к одним и тем же запросам, а разные документы подходят к разным запросам [2].

Исследование проводится над набором документов вида:

$$D = \{d_j \mid j = 1, \ldots, |D|\}, \tag{1.1}$$

содержащей разнообразные тематические классы. Цель алгоритмов классификации без учителя — автоматически классифицировать документы на кластеры вида:

$$C = \{c_j \mid j = 1, \ldots, |C|\}, \tag{1.2}$$

так чтобы каждый кластер представлял собой группу тематически схожих документов. Задача кластеризации сводится к определению оптимального множества кластеров C, удовлетворяющего заданным критериям качества [2].

1.4 Иерархические алгоритмы

Иерархические алгоритмы создают структурированное множество кластеров — иерархию, которое может оказаться весьма информативным для некоторых приложений [2].

В рамках иерархической кластеризации существуют агломеративные (восходящие) и дивизимные (нисходящие) подходы. Агломеративные методы последовательно объединяют мелкие кластеры в более крупные, в то время как дивизимные методы начинают с одного большого кластера и делят его на более мелкие. Дивизимные методы часто требуют дополнительных алгоритмов для определения способа разделения и могут быть более эффективными при ограничении процесса до верхних уровней иерархии без полного разделения на индивидуальные документы. Сложность дивизимного алгоритма зависит от выбранного дополнительного алгоритма плоской кластеризации, если выбран алгоритм k-средних, то $O(|D|)$ [2].

1.5 Представление данных в задаче

В данной лабораторной работе рассматривается использование иерархической кластеризации для анализа текстовых документов. Основа алгоритма — представление документов в виде векторов терминов с весами, вычисленными по методу *TF-IDF*.

Входные данные

Для каждого документа в наборе вида (1.1) формируется вектор:

$$\mathbf{d}_i = (d_{i1}, \ldots, d_{iT}), \tag{1.3}$$

где T — количество уникальных терминов во всех документах, а d_{ij} — вес j-го термина в i-м документе.

Вычисление весов терминов

Веса терминов рассчитываются по формулам:

$$d_{ij} = \frac{w_{ij}}{\|\mathbf{w}_i\|}, \tag{1.4}$$

$$w_{ij} = tf_{ij} \times \log\left(\frac{|D|}{df_j}\right). \tag{1.5}$$

В формулах (1.4) – (1.5) используются следующие обозначения: tf_{ij} — частота j-го термина в i-м документе, $|D|$ — общее количество документов в наборе, df_j — документная частота термина j, и $\|\mathbf{w}_i\|$ — евклидова норма вектора весов i-го документа.

Выходные данные Алгоритм генерирует структуру кластеров, представленную иерархическим деревом, и метки кластеров вида (1.2), характеризующие группы родственных документов.

1.6 Алгоритм дивизимной иерархической кластеризации

1) Вход: множество проиндексированных документов D вида (1.1).
2) Вычислить веса терминов для каждого документа с использованием метода *TF-IDF*, по формуле (1.4).
3) Изначально все элементы принадлежат одному кластеру C_i, $i = 0$ (1.2).
4) Разделить кластер на два подкластера C_{i+1} и C_{i+2}.
5) Применить алгоритм *k-средних* к подкластерам C_{i+1} и C_{i+2}.
6) Увеличить i и повторить шаги 4 и 5 до достижения желаемой структуры кластеризации.
7) Возвратить итоговую структуру кластеров.

1.7 Алгоритм k-средних

При заранее известном числе кластеров k, алгоритм k-средних начинает с некоторого начального разбиения документов и уточняет его, оптимизируя целевую функцию – среднеквадратичную ошибку кластеризации как среднеквадратичное расстояние между документами и центрами их кластеров:

$$e(D, C) = \sum_{j=1}^{k} \sum_{i:d_i \in C_j} \|d_i - \mu_j\|^2, \tag{1.6}$$

где μ_j — центр, или центроид, кластера C_j, $|C| = k$, вычисляющийся по формуле

$$\mu_j = \frac{1}{|C_j|} \sum_{i:d_i \in C_j} d_i, \tag{1.7}$$

где $|C_j|$ — количество документов в C_j. Идеальным кластером алгоритм k - средних считает сферу с центроидом в центре сферы.

Алгоритм k-средних состоит из следующих шагов [2].

1) *Вход:* множество проиндексированных документов D, количество кластеров k.

2) Назначить начальные центры для кластеров $\{\mu_j\},\ j = 1,\ ...,\ k$ случайным образом.

3) Установить каждому кластеру C_j пустой набор, $j = 1,\ ...,\ k$.

4) Для каждого документа $d_i \in D$ выполнить:

   – найти ближайший центр кластера $j^* := \arg\min_j \|\mu_j - d_i\|,\ j = 1,\ ...,\ k$;

   – добавить документ d_i в соответствующий кластер $C_{j^*} := C_{j^*} \cup \{d_i\}$.

5) Для каждого кластера C_j обновить центр как среднее его элементов:

$$\mu_j := \frac{1}{|C_j|} \sum_{i:d_i \in C_j} d_i.$$

6) Если условие остановки не достигнуто, вернуться к шагу 4.

7) *Выход:* множество центров кластеров $\{\mu_j\}$ и множество самих кластеров C.

Вывод

В данном разделе были рассмотрены конвейерная обработка данных, алгоритмы классификации полнотекстовых документов, алгоритмы классификации без учителя, иерархические алгоритмы, алгоритм дивизимной иерархической кластеризации, алгоритм дивизимной иерархической кластеризации и алгоритм k-средних.

2 Конструкторский раздел

В данном разделе будут представлены требования к программному обеспечению, описание используемых типов данных и схемы реализуемых алгоритмов.

2.1 Требования к программному обеспечению

К программе предъявлен ряд требований:

– должен присутствовать интерфейс для выбора действий;

– считывание данных должно производиться из файла;

– результат должен записываться в файл;

– должен присутствовать замер реального времени для реализаций алгоритмов;

– результат замера должен выводится в виде таблицы.

2.2 Описание используемых типов данных

При реализации алгоритмов будут использованы следующие структуры и типы данных:

– массив символов для хранения терма;

– вещественное число для хранения *TF-IDF* терма;

– мьютекс — примитив синхронизации.

2.3 Разработка алгоритмов

На рисунке 2.1 приведена схема дивизимной иерархической кластеризации. На рисунке 2.2 приведена схема алгоритма k-средних. На рисунке 2.3 приведена схема алгоритма запуска конвейера. На рисунке 2.4 приведена схема обслуживающего устройства, которое создает начальные запросы. На рисунке 2.5 приведена схема обслуживающего устройства, которое считывает *TF-IDF* из файла. На рисунке 2.6 приведена схема обслуживающего устройства, которое кластеризует документы. На рисунке 2.7 приведена схема обслуживающего устройства, которое записывает результат в файл.

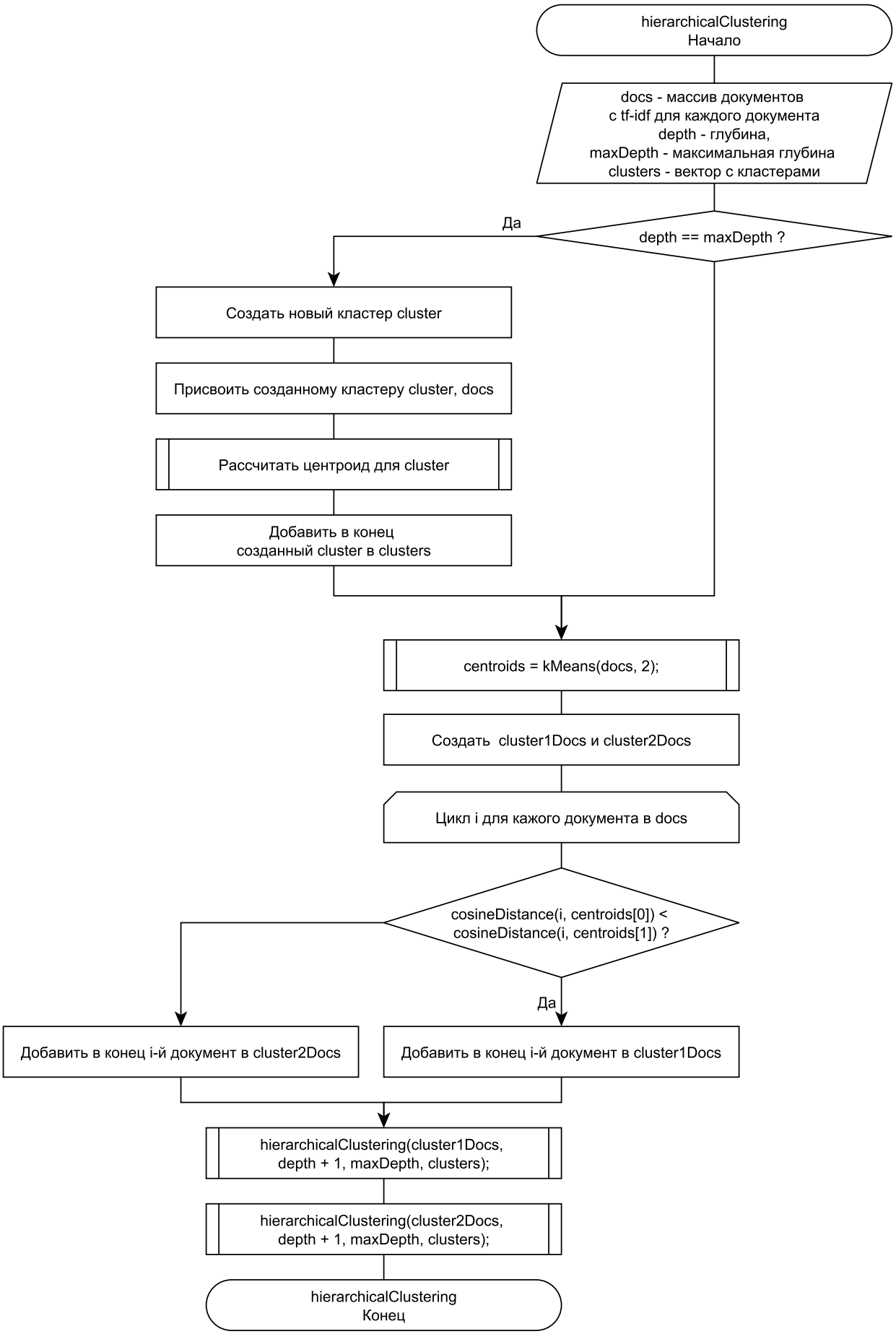

Рисунок 2.1 – Схема дивизимной иерархической кластеризации

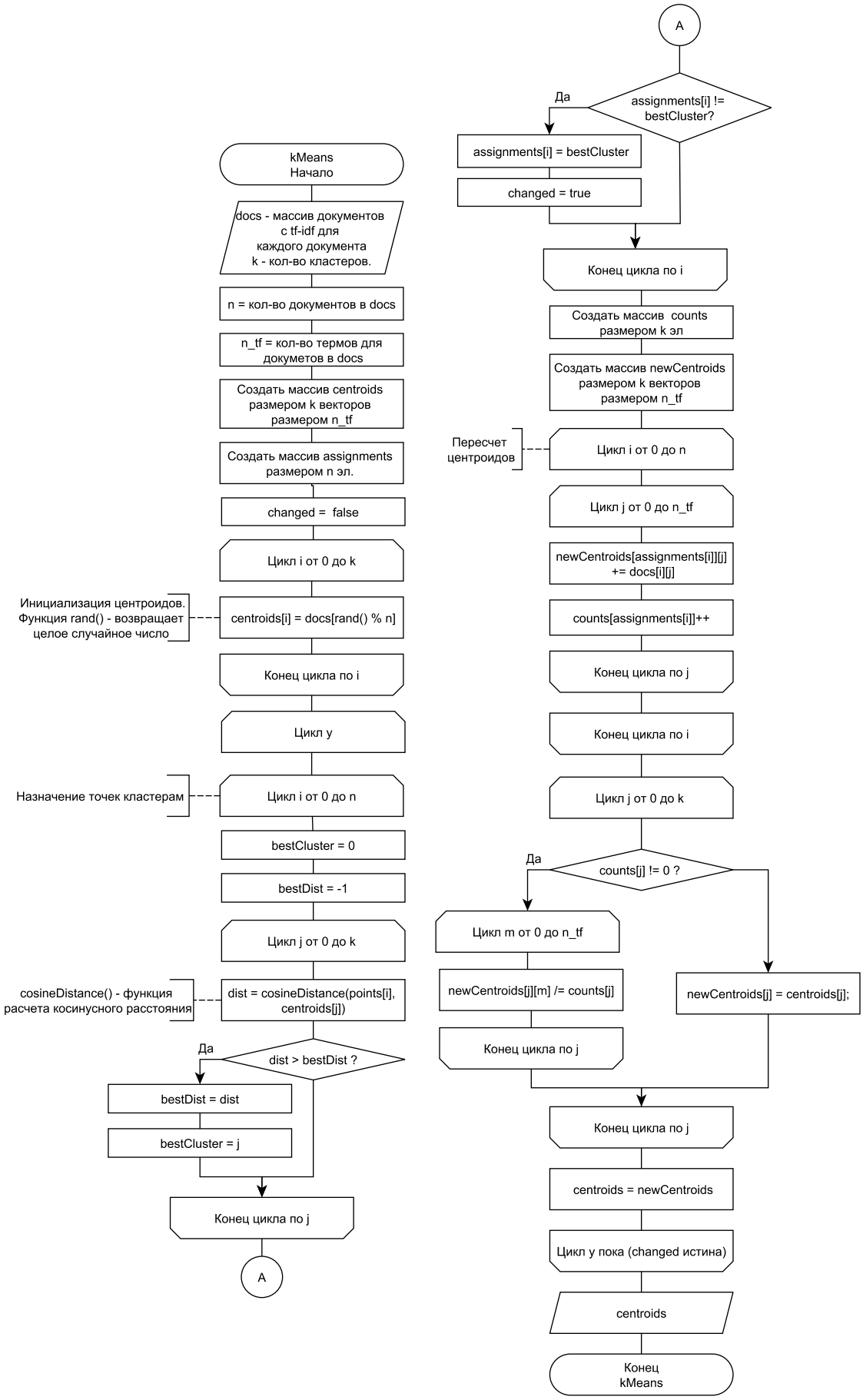

Рисунок 2.2 – Схема алгоритма k-средних

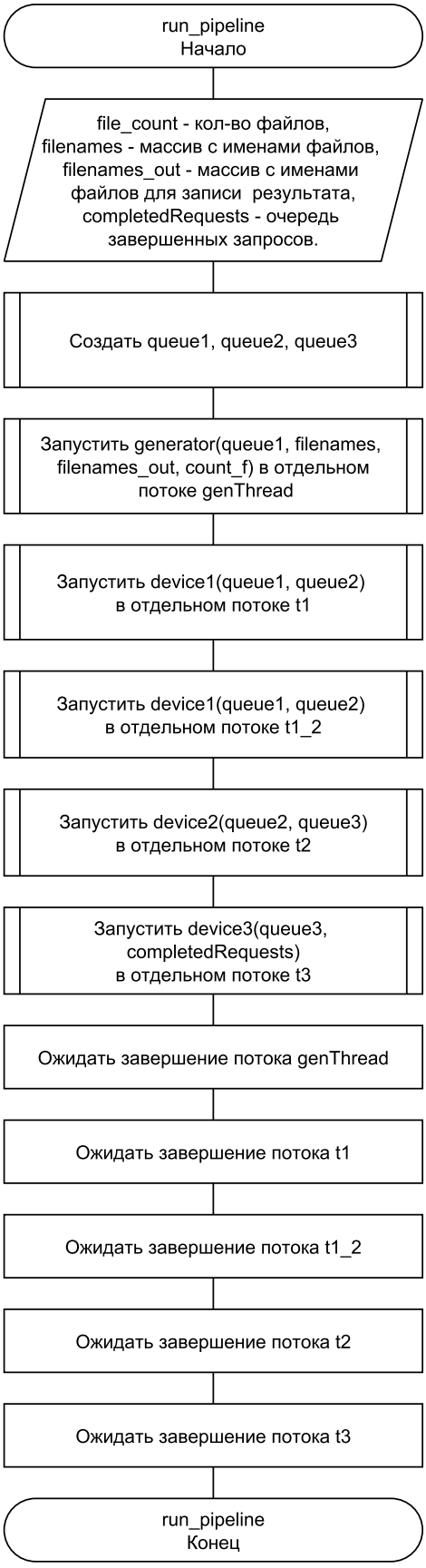

Рисунок 2.3 – Схема алгоритма запуска конвейера

Рисунок 2.4 – Схема алгоритма обслуживающего устройства, которое создает начальные запросы

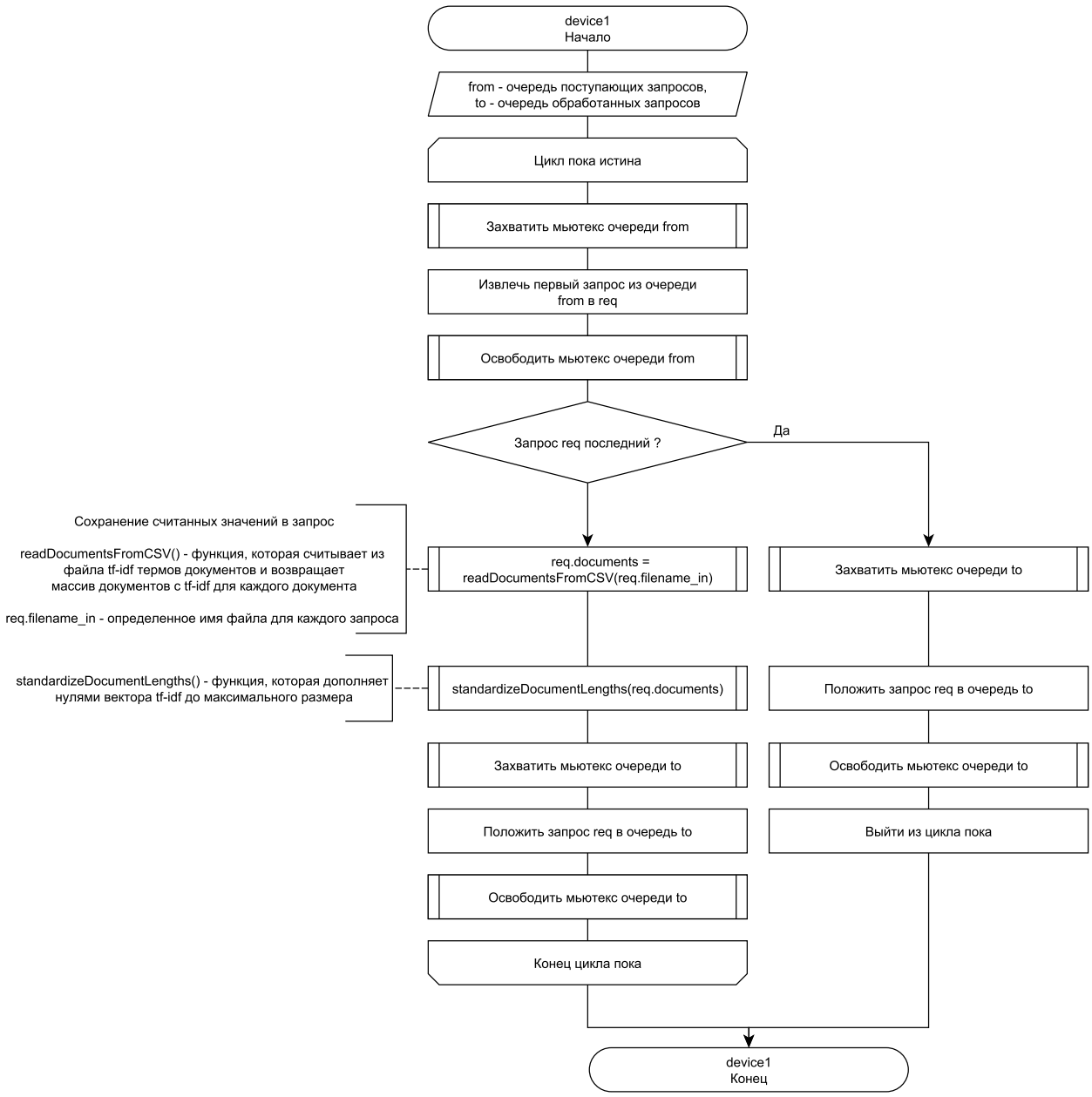

Рисунок 2.5 – Схема алгоритма обслуживающего устройства, которое считывает *TF-IDF* из файла

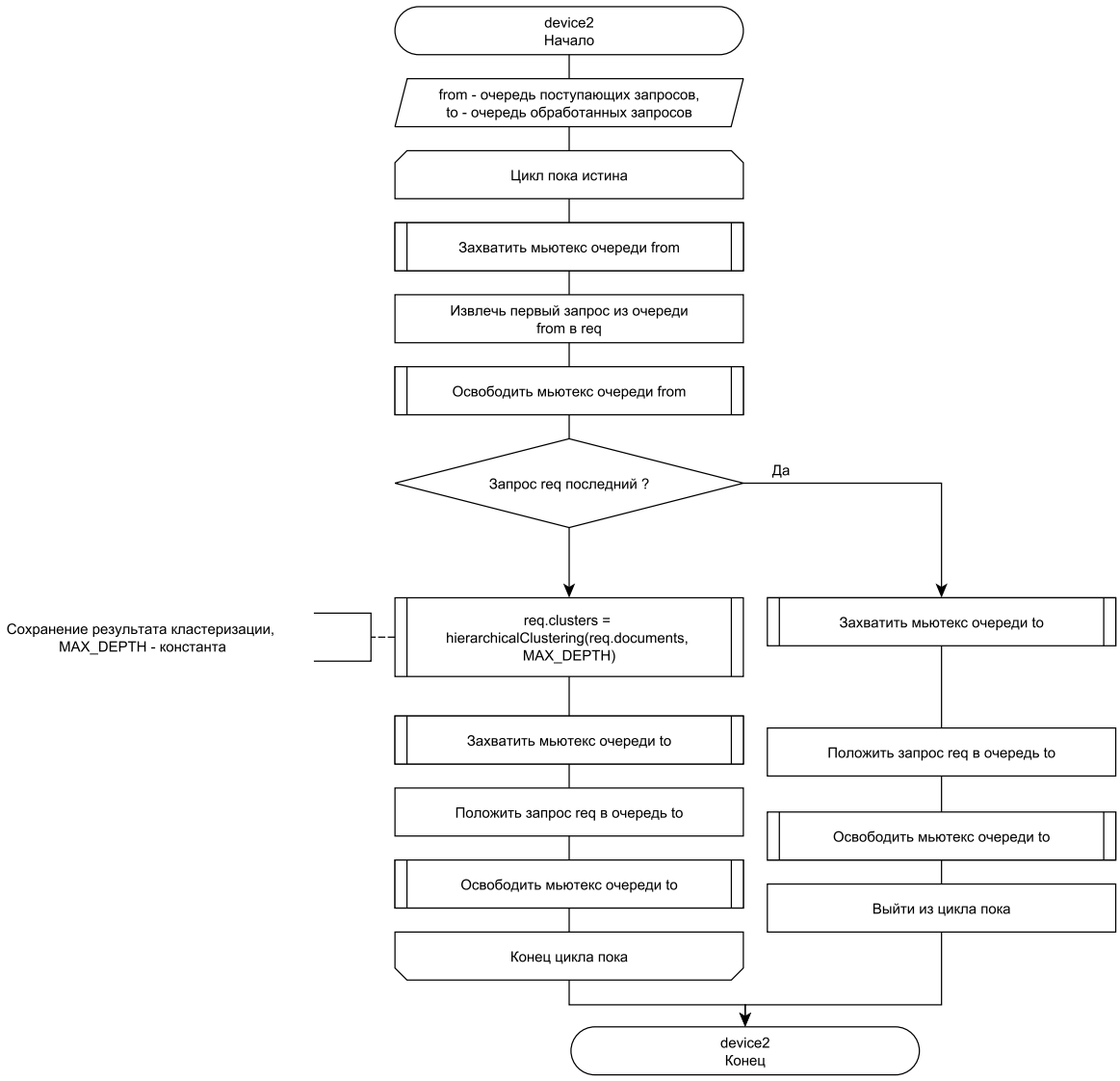

Рисунок 2.6 – Схема алгоритма обслуживающего устройства, которое кластеризует документы

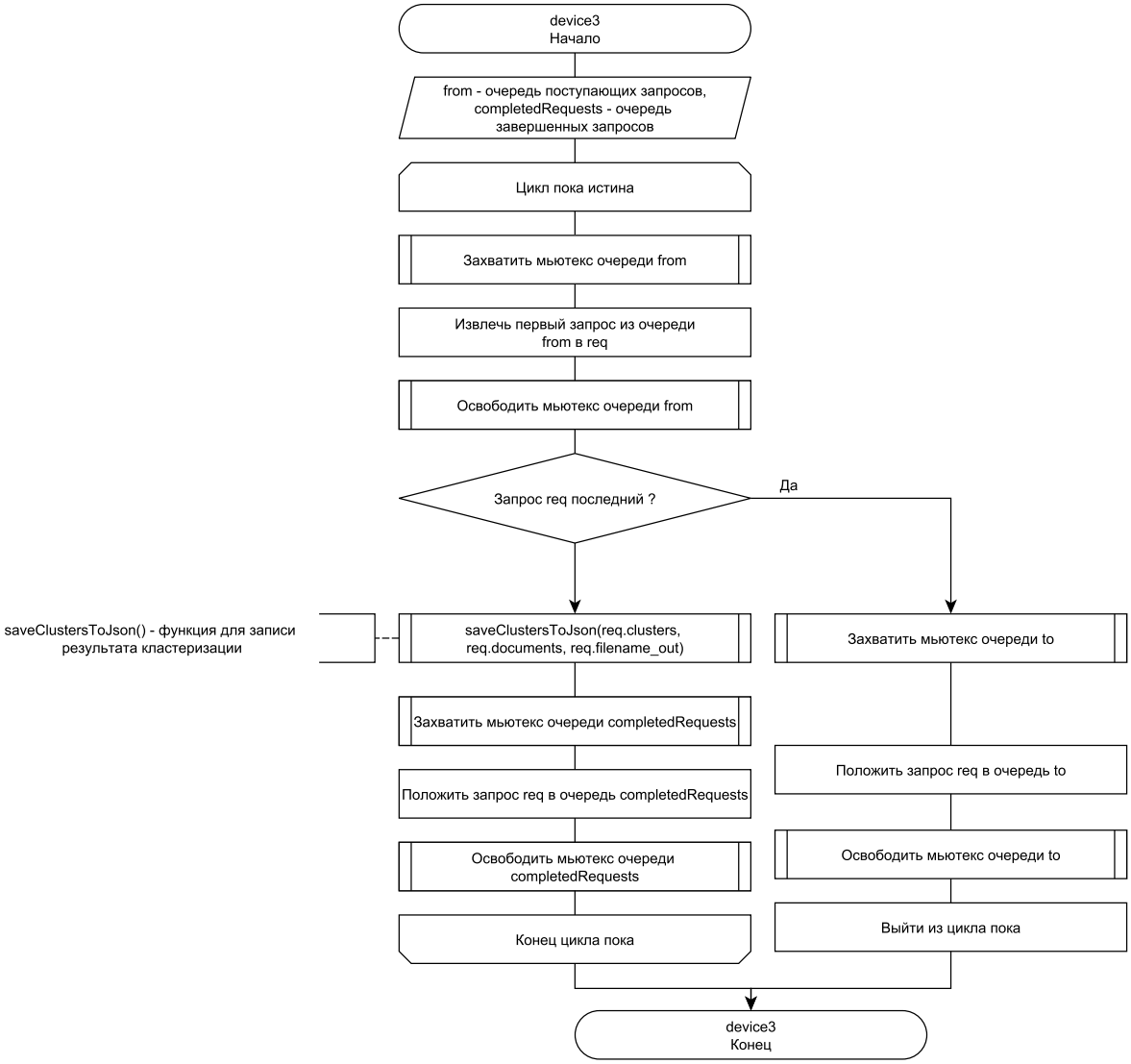

Рисунок 2.7 – Схема алгоритма обслуживающего устройства, которое записывает результат в файл

Вывод

В данном разделе были представлены требования к программному обеспечению, описание используемых типов данных и схемы реализуемых алгоритмов.

3 Технологический раздел

В данном разделе будут представлены средства реализации, сведения о модулях программы и листинги кода реализации алгоритмов.

3.1 Средства реализации

В качестве языка программирования для разработки программного обеспечения был выбран язык *C++* [4].

Данный выбор обусловлен следующим:

- язык позволяет реализовать все алгоритмы, выбранные в результате проектирования;
- язык поддерживает все типы и структуры данных, которые были выбраны в результате проектирования;
- язык позволяет работать с нативными потоками [5].

Время выполнения реализаций было замерено с помощью функции *clock* [6]. Для хранения термов использовалась структура данных *string* [7], в качестве массивов использовалась структура данных *vector* [8]. В качестве примитива синхронизации использовался *mutex* [9].

Для создания потоков и работы с ними был использован класс *thread* из стандартной библиотеки выбранного языка [5]. В листинге 3.1, приведен пример работы с описанным классом, каждый объект класса представляет собой поток операционной системы, что позволяет нескольким функциям выполняться параллельно [5].

Листинг 3.1 – Пример работы с классом thread

```
#include <iostream>
#include <thread>

void foo(int a){
    std::cout << a << '\n';
}

int main(){
    std::thread thread(foo, 10);
    thread.join();

    return 0;
}
```

3.2 Сведения о модулях программы

Данная программа разбита на следующие модули:

– *main.cpp* — файл, который содержит точку входа в программу;

– *Cluster.cpp* и *Cluster.h* — модуль, который реализует класс *Cluster*;

– *kMeans.cpp* и *kMeans.h* — модуль, содержащий реализацию функции *kMeans*;

– *Klastering.cpp* и *Klastering.h* — модуль, содержащий реализации функций дивизимной иерархической кластеризации;

– *ThreadSafeQueue.cpp* и *ThreadSafeQueue.h* — модуль, содержащий реализации функций обслуживающих устройств;

– *io.cpp* и *io.h* — модуль, содержащий реализации функций для работы входными и выходными файлами;

– *Document.cpp* и *Document.h* — модуль, который реализует класс *Document*;

– *Term.cpp* и *Term.h* — модуль, который реализует класс *Term*;

3.3 Реализация алгоритмов

В листинге 3.2 приведена реализация дивизимной иерархической кластеризации. В листинге А.1 приведена реализация алгоритма k-средних. В листинге 3.3 приведена реализация алгоритма, запуска конвейера. В листингах 3.4 – 3.7 приведены реализации алгоритмов обслуживающих устройств.

Листинг 3.2 – Реализация дивизимной иерархической кластеризации

```
void hierarchicalClustering(const
   std::vector<std::vector<double>> &docs,
                    int depth,
                    int maxDepth,
                    std::vector<Cluster> &clusters) {
    if (depth == maxDepth) {
        Cluster cluster;
        cluster.docs = docs;
        cluster.calculateCentroid();
        clusters.push_back(cluster);
        return;
    }

    // Применяем k-средних для разделения на 2 кластера
    std::vector<std::vector<double>> centroids = kMeans(docs, 2);
    std::vector<std::vector<double>> cluster1Docs, cluster2Docs;

    for (const auto &doc: docs) {
        if (cosineDistance(doc, centroids[0]) <
           cosineDistance(doc, centroids[1])) {
            cluster1Docs.push_back(doc);
        }
        else {
            cluster2Docs.push_back(doc);
        }
    }

    // Рекурсивно разделяем каждый подкластер
    hierarchicalClustering(cluster1Docs, depth + 1, maxDepth,
       clusters);
    hierarchicalClustering(cluster2Docs, depth + 1, maxDepth,
       clusters);
}
```

Листинг 3.3 – Реализация алгоритма запуска конвейера

```
void run_pipeline(const std::vector<std::string> &filenames,
    const std::vector<std::string> &filenames_out, int count_f,
                    AtomicQueue<Request> &completedRequests) {
        AtomicQueue<Request> queue1, queue2, queue3;

    std::thread genThread(generator, std::ref(queue1),
       std::ref(filenames), std::ref(filenames_out), count_f);
    std::thread t1(device1, std::ref(queue1), std::ref(queue2));
    std::thread t1_2(device1, std::ref(queue1),
       std::ref(queue2));
    std::thread t2(device2, std::ref(queue2), std::ref(queue3));
    std::thread t3(device3, std::ref(queue3),
       std::ref(completedRequests));
    genThread.join();
    t1.join();
    t1_2.join();
    t2.join();
    t3.join();
}
```

Листинг 3.4 – Реализация алгоритма обслуживающего устройства, которое создает начальные запросы

```
void generator(AtomicQueue<Request> &q, const
   std::vector<std::string> &filenames,
                const std::vector<std::string> &filenames_out,
                   int count_f) {
    for (size_t i = 0; i < count_f; ++i) {
        q.push({{}, {}, filenames[i], filenames_out[i]});
    }
    q.push({{}, {}, "", "", true});
}
```

Листинг 3.5 – Реализация алгоритма обслуживающего устройства, которое считывает *TF-IDF* из файла

```
void device1(AtomicQueue<Request> &from, AtomicQueue<Request>
   &to) {
    while (true) {
        Request req = from.pop();
        if (req.is_last) {
            to.push(req);
            break;
        }

        req.time_start_1 = get_time();
        req.documents = readDocumentsFromCSV(req.filename_in);
        standardizeDocumentLengths(req.documents);
        req.time_end_1 = get_time();

        to.push(req);
    }
}
```

Листинг 3.6 – Реализация алгоритма обслуживающего устройства, которое кластеризует документы

```
void device2(AtomicQueue<Request> &from, AtomicQueue<Request>
   &to) {
    while (true) {
        Request req = from.pop();
        if (req.is_last) {
            to.push(req);
            break;
        }

        req.time_start_2 = get_time();
        req.clusters = hierarchicalClustering(req.documents,
           MAX_DEPTH);
        req.time_end_2 = get_time();

        to.push(req);
    }
}
```

Листинг 3.7 – Реализация алгоритма обслуживающего устройства, которое записывает результат в файл

```
void device3(AtomicQueue<Request> &from, AtomicQueue<Request>
   &completedRequests) {
    while (true) {
        Request req = from.pop();
        if (req.is_last) {
            break;
        }

        req.time_start_3 = get_time();
        saveClustersToJson(req.clusters, req.documents,
           req.filename_out);
        req.time_end_3 = get_time();
        completedRequests.push(req);
    }
}
```

Вывод

В данном разделе были представлены средства реализации, сведения о модулях программы и листинги кода реализации алгоритмов.

4 Исследовательская часть

В данном разделе будут приведены демонстрация работы программы, технические характеристики устройства, сравнительный анализ времени выполнения реализуемых алгоритмов.

4.1 Демонстрация работы программы

На рисунке 4.1 представлена демонстрация работы разработанного программного обеспечения, а именно показан результат собранной статистики для каждой заявки. На рисунке 4.1, N — означает номер заявки, Time 1, Time 2, Time 3 — означают время выполнения в миллисекундах для обслуживающих устройств, которое считывает *TF-IDF* из файла, которое кластеризует документы и которое записывает результат в файл, соответственно. Пример входных файлов представлен на рисунке 4.2. Пример результата кластеризации в виде json файла, представлен на рисунке 4.3.

```
                Menu
1. Run the sequential version of document clustering.
2. Run the conveyor version of document clustering.
3. Perform timing measurements of implemented algorithms.
4. Display the tf-idf of documents.
0. Exit.

Select an option (0-4):2

| N     | Time 1 (ms)     | Time 2 (ms)     | Time 3 (ms)     |Total Time (ms) |
| 1     | 47.16           | 2.63            | 10.37           |60.16           |
| 2     | 47.25           | 1.19            | 10.38           |58.82           |
| 3     | 44.16           | 1.66            | 10.50           |56.32           |
| 4     | 47.42           | 1.69            | 11.22           |60.33           |
| 5     | 38.02           | 1.64            | 10.68           |50.35           |
| 6     | 38.22           | 2.30            | 10.43           |50.95           |
| 7     | 52.29           | 2.45            | 10.54           |65.27           |
| 8     | 49.88           | 1.45            | 10.52           |61.86           |
| 9     | 38.03           | 1.76            | 10.41           |50.20           |
| 10    | 43.58           | 2.29            | 10.79           |56.66           |
```

Рисунок 4.1 – Демонстрация работы программы

```
Document,Term,TF-IDF
news0,мобилизация,0.323565390388597
news0,нелогичный,0.0916643406262363
news0,заявить,0.054831234176486984
news0,депутат,0.06731898150728603
news0,верховный,0.07324778740136165
news0,рада,0.08089134759714936
news0,евгений,0.07324778740136165
news0,шевченко,0.09166434062623631
news0,интервью,0.06731898150728603
news0,украинский,0.1167581850158465
news0,политолог,0.09166434062623631
news0,вадим,0.09166434062623631
news0,закон,0.05837909250792325
news0,загнать,0.09166434062623631
news0,наш,0.027953534093788215
news0,этот,0.025484005145127456
news0,собака,0.09166434062623631
news0,стать,0.027953534093788215
news0,мочь,0.017998127726737652
```

Рисунок 4.2 – Пример входного файла


```
{
    "Кластер 1": [
        "article6",
        "article3",
        "news19",
        "news15",
        "news14",
        "news7",
        "news20",
        "news2",
        "news16",
        "news18",
        "news12",
        "news8",
        "news17",
        "news9",
        "news11",
        "news1",
        "article17"
    ],
    "Кластер 2": [
        "book17",
        "news6",
        "book0",
        "book3",
        "book1",
```


Рисунок 4.3 – Пример результата кластеризации

4.2 Технические характеристики

Технические характеристики устройства, на котором выполнялись замеры по времени, представлены далее.

– Процессор: Ryzen 5 4600H, 6 процессорных ядер архитектуры Zen 2 и 12 потоков, работающих на базовой частоте в 3.0 ГГц (до 4.0 ГГц в Turbo режиме), 12 логических ядер [10]

– Оперативная память: 16 ГБайт.

– Операционная система: Windows 10 Pro 64-разрядная система [11].

При замерах времени ноутбук был включен в сеть электропитания и был нагружен только системными приложениями.

4.3 Время выполнения реализаций алгоритмов

Результаты замеров времени выполнения реализации алгоритма иерархической кластеризации документов в зависимости от числа заявок приведены в таблице 4.1. Каждый замер проводился 100 раз, после чего рассчитывалось их среднее арифметическое значение.

На рисунке 4.4 изображен график зависимостей времени выполнения реализаций от числа заявок.

Таблица 4.1 – Зависимость времени выполнения (в мс) от количества заявок

Кол-во заявок	Послед. реализация (мс)	Конвейер. реализация (мс)
1	44	35
2	88	69
3	133	79
4	177	108
5	222	123
6	264	149
7	309	164
8	354	193
9	402	208
10	443	229
11	484	249
12	530	268
13	573	292
14	690	326
15	661	368
16	713	390
17	763	384
18	798	406
19	846	443
20	898	478
21	930	491
22	970	489
23	1024	503
24	1061	577

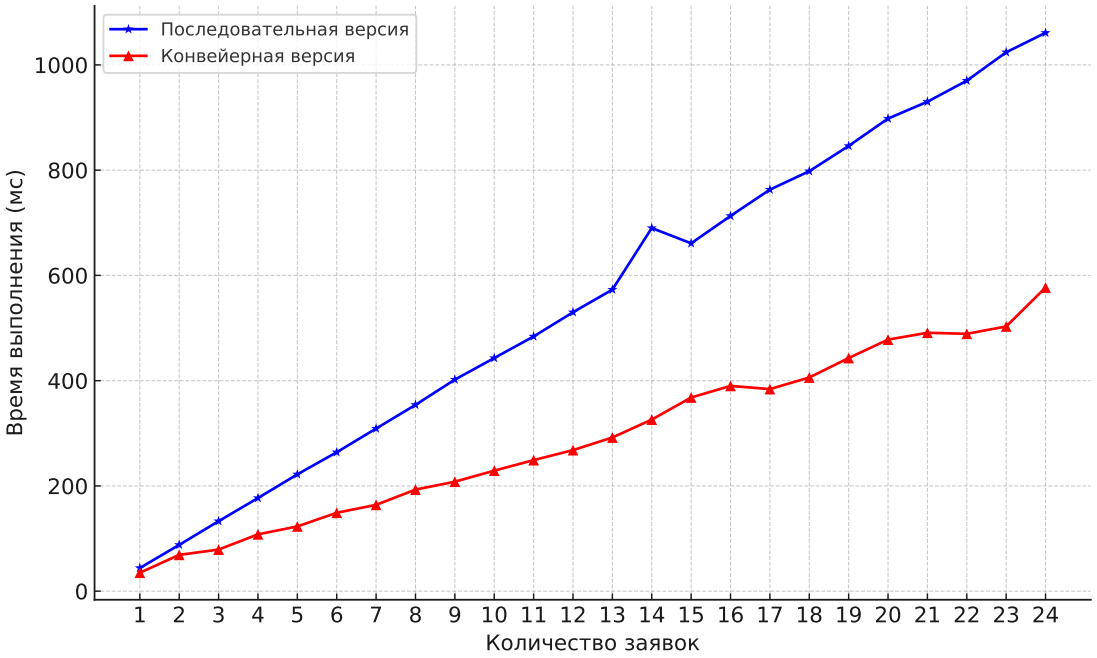

Рисунок 4.4 – График зависимости времени выполнения реализации от числа заявок

Вывод

В результате исследования реализуемых алгоритмов по времени выполнения можно сделать следующий вывод. Конвейерная реализация алгоритма более эффективна по времени выполнению при увеличении количества заявок, при 24 заявках конвейерная версия обработки на 45.62% эффективнее по сравнению с последовательной реализацией алгоритма (см. таблицу 4.1). Такой результат объясняется тем, что в конвейерной реализации потоки могут выполнять различные этапы работы параллельно, что позволяет сократить время обработки последовательности заявок.

ЗАКЛЮЧЕНИЕ

Цель лабораторной работы достигнута описаны параллельные конвейерные вычислений на основе нативных потоков для иерархической кластеризации документов.

В ходе выполнения лабораторной работы были решены все задачи:

1) описана организация конвейерной обработки данных;

2) описаны алгоритмы иерархической кластеризации.

3) спроектировано программное обеспечение, позволяющее выполнять конвейерную и последовательную обработку данных;

4) разработано программное обеспечение, позволяющее выполнять конвейерную и последовательную обработку данных;

5) проведен сравнительный анализ времени работы конвейерной и последовательной версии.

В результате исследования реализуемых алгоритмов по времени выполнения можно сделать следующий вывод. Конвейерная реализация алгоритма более эффективна по времени выполнению при увеличении количества заявок, при 24 заявках конвейерная версия обработки на 45.62% эффективнее по сравнению с последовательной реализацией алгоритма (см. таблицу 4.1). Такой результат объясняется тем, что в конвейерной реализации потоки могут выполнять различные этапы работы параллельно, что позволяет сократить время обработки последовательности заявок.

ПРИЛОЖЕНИЕ А

Листинг А.1 – Реализация алгоритма k-средних

```
std::vector<std::vector<double>> kMeans(const
   std::vector<std::vector<double>> &docs, int k) {
    int n = docs.size();
    std::vector<std::vector<double>> centroids(k,
       std::vector<double>(docs[0].size()));
    std::vector<int> assignments(n, 0);
    // Инициализация центроидов
    for (int i = 0; i < k; ++i) {
        centroids[i] = docs[rand() % n];
    }
    bool changed;
    do {
        changed = false;
        // Назначение точек кластерам
        for (int i = 0; i < n; ++i) {
            double bestDist = -1.0;
            int bestCluster = 0;
            for (int j = 0; j < k; ++j) {
                double dist = cosineDistance(docs[i],
                   centroids[j]);
                if (dist > bestDist) {
                    bestDist = dist;
                    bestCluster = j;
                }
            }
            if (assignments[i] != bestCluster) {
                assignments[i] = bestCluster;
                changed = true;
            }
        }
        // Обновление центроидов
        std::vector<int> counts(k, 0);
        std::vector<std::vector<double>> newCentroids(k,
           std::vector<double>(docs[0].size(), 0.0));
        for (int i = 0; i < n; ++i) {
            for (size_t j = 0; j < docs[i].size(); ++j) {
                newCentroids[assignments[i]][j] += docs[i][j];
```

```
34              }
35              counts[assignments[i]]++;
36          }
37          for (int j = 0; j < k; ++j) {
38              if (counts[j] != 0) {
39                  for (size_t m = 0; m < newCentroids[j].size();
                        ++m) {
40                      newCentroids[j][m] /= counts[j];
41                  }
42              }
43              else {
44                  newCentroids[j] = centroids[j];
45              }
46          }
47          centroids = newCentroids;
48      } while (changed);
49
50      return centroids;
51  }
```

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Принципы конвейерной технологии [Электронный ресурс]. — Режим доступа: https://www.sites.google.com/site/shoradimon/18-principy-konvejernoj-tehnologii (дата обращения: 24.12.2023).
2. *Большакова Е. И., Клышински Э. С., Ландэ Д. В., Носков А. А., Пескова О. В., Ягунова Е. В.* Автоматическая обработка текстов на естественном языке и компьютерная лингвистика: учеб. пособие. — МИЭМ, 2011. — С. 1—272.
3. *Котелина Н. О., Матвийчук Б. Р.* Кластеризация изображения методом K-средних // Вестник Сыктывкарского университета. — 2019. — Т. Выпуск 3 (32). — С. 102—106.
4. C++ reference [Электронный ресурс]. — Режим доступа: https://en.cppreference.com/w/ (дата обращения: 20.12.2023).
5. Concurrency support library [Электронный ресурс]. — Режим доступа: https://en.cppreference.com/w/cpp/thread (дата обращения: 20.12.2023).
6. std::clock [Электронный ресурс]. — Режим доступа: https://en.cppreference.com/w/cpp/chrono/c/clock (дата обращения: 20.12.2023).
7. Strings library [Электронный ресурс]. — Режим доступа: https://en.cppreference.com/w/cpp/string (дата обращения: 20.12.2023).
8. std::vector [Электронный ресурс]. — Режим доступа: https://en.cppreference.com/w/cpp/string (дата обращения: 20.12.2023).
9. std::mutex [Электронный ресурс]. — Режим доступа: https://en.cppreference.com/w/cpp/thread/mutex (дата обращения: 20.12.2023).
10. Ryzen 4600H [Электронный ресурс]. — Режим доступа: https://www.amd.com/en/products/apu/amd-ryzen-5-4600h (дата обращения: 20.09.2023).
11. Windows 10 Pro 2h21 64-bit [Электронный ресурс]. — Режим доступа: https://www.microsoft.com/ru-ru/software-download/windows10 (дата обращения: 25.09.2022).