|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | | Министерство науки и высшего образования РФ | | | | | | | | |  | |
|  | | | | |  | | |  | | | | |
|  | | | | ФГБОУ ВО «Пермский государственный национальный исследовательский университет» | | | | |  | | | |
|  | | | | |  | | |  | | | | |
|  | | | **Технологии разработки распределенных приложений**  Лабораторная работа №4  «Разработка распределенного приложения для локальной сети» | | | | | | |  | | |
|  | | | | |  | | |  | | | | |
|  | Работу выполнили  Студенты гр. ПМИ-4-18:  Колесников А.С  Пухов Н.А. | | | | |  | Проверил  профессор, доктор физико-математических наук  Постаногов И.С  «\_\_\_» \_\_\_\_\_\_\_\_\_\_ 2021 г. | | | | |  | |
|  | | | | |  | | |  | | | | |
|  | | | | | Пермь 2021 | | |  | | | | |

СОДЕРЖАНИЕ

[1 Постановка задачи 3](#_Toc90877317)

[2 Архитектура системы 3](#_Toc90877318)

[3 Архитектура модулей системы 3](#_Toc90877319)

[3.1 Архитектура сервера 3](#_Toc90877320)

[3.2 Архитектура диспетчера 3](#_Toc90877321)

[3.3 Архитектура клиента 3](#_Toc90877322)

[4 Методы коммуникации компонентов системы 5](#_Toc90877323)

[5 Способ передачи данных 5](#_Toc90877324)

[6 Структура передаваемых данных 5](#_Toc90877325)

[7 Отказоустойчивость системы 7](#_Toc90877326)

[8 Исходный код приложения 7](#_Toc90877327)

[8.1 Клиент 7](#_Toc90877328)

[8.2 Сервер-хранилище 14](#_Toc90877329)

[8.3 Сервер-диспетчер 14](#_Toc90877330)

[8.4 Сервер-документа 14](#_Toc90877331)

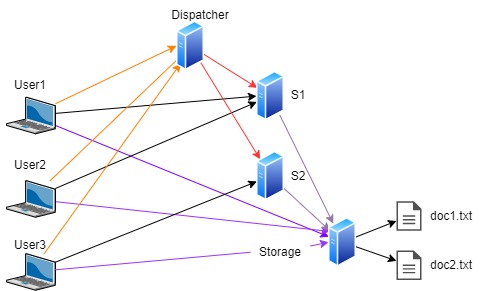
1. Постановка задачи

Задача заключается в создании распределенного приложения по совместному редактированию документов. Пользователи могут одновременно вносить свои изменения в один или несколько документов и моментально видеть изменения документа, которые производятся другими пользователями. При этом приложением должны разрешаться конфликтные ситуации, когда несколько пользователей пытаются изменить один и тот же фрагмент текста.

1. Архитектура системы

Архитектура приложения является сервис-ориентированной и состоит из одного диспетчера, одного файлового хранилища и одного или нескольких серверов для работы с одним документом. Приложением может пользоваться один или несколько клиентов.

1. Диспетчер (Dispatcher). В функции диспетчера входит предоставление клиентам информации о том, какой сервер работает с тем или иным документом.
2. Файловое хранилище (Storage). Файловое хранилище содержит документы и предоставляет команды для получения списка доступных документов, получения/сохранения содержимого одного документа.
3. Рабочий сервер (S1, S2). В обязанности рабочего сервера входит предоставление услуг клиентам по работе с одним документом: получение содержимого документа актуальной версии, прием и отправка сделанных изменений.



Рисунок

1 – Архитектура системы

Выбор архитектуры приложения продиктован необходимостью иметь состояние на рабочих серверах, так как совместное редактирование документов предполагает наличие на серверах объектов документов в оперативной памяти, которые должны содержать информацию о том, какие изменения относительно какой версии документа сделаны.

1. Архитектура модулей системы

Серверная часть была реализована на языке Python, клиентская часть на языке C#.

* 1. Архитектура диспетчера

Диспетчер предоставляет клиентам информации о том, какой сервер работает с тем или иным документом. Это позволяет сделать балансировку нагрузку между несколькими рабочими серверами, тем самым давая возможность горизонтального масштабирования. Предоставляемая диспетчером информация состоит из IP-адреса рабочего сервера. Диспетчер многопоточен.

* 1. Архитектура хранилища файлов

Файловое хранилище содержит документы и предоставляет команды для получения списка доступных документов, получения/сохранения содержимого одного документа. Отдельный сервер хранилища необходим для централизованного доступа к разделяемым ресурсам, чтобы поддерживать их консистентность. Многопоточность присутствует.

* 1. Архитектура рабочего сервера

В обязанности рабочего сервера входит предоставление услуг клиентам по работе с одним документом: получение содержимого документа актуальной версии, прием и отправка сделанных изменений. Многопоточность присутствует на уровне редактирования нескольких документов.

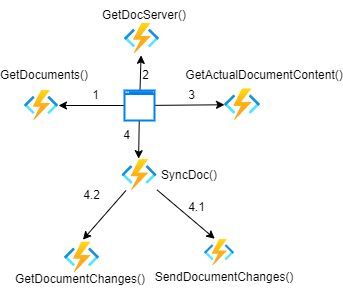
* 1. Архитектура клиента

Клиент был реализован на объектно-ориентированном языке C# на технологии WPF.

Алгоритм работы клиента:

1. Происходит запуск графического приложения, которое подключается к серверу-хранилищу и получает список всех доступных для редактирования документов;
2. После выбора пользователем нужного документа, отправляет запрос на сервер-диспетчер для получения адреса сервера, на котором хранится выбранный документ;
3. Клиент подкачается к серверу с документом и получает актуальную версию документа;
4. Происходит синхронизация версии на клиенте с версией на сервере:
   1. Вычисляются различия в версиях клиента и сервера, которые после отправляются серверу;
   2. Клиент запрашивает изменения с сервера и применяет их к своей версии документа.

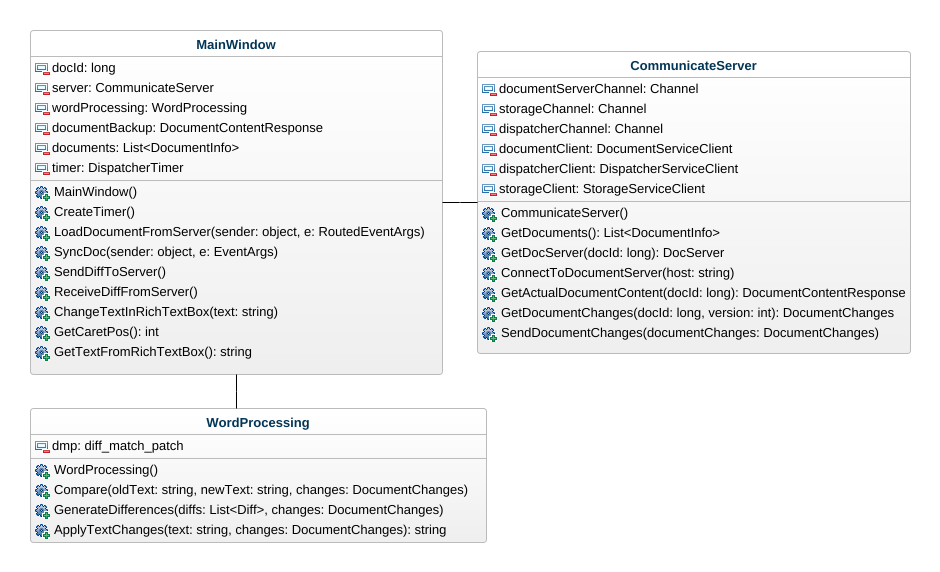
Данный алгоритм изображен на рисунке 2.



Рисунок

2 – Архитектура клиента

Также была составлена диаграмма классов (рис. 3), где класс MainWindow класс главного окна графического приложения, который содержит все обработчики событий. Класс CommunicateServer отвечает за коммуникацию с серверами. Класс WordProcessing выделяет и применяет изменения к тексту.



Рисунок

3 – Диаграмма классов клиента

1. Методы коммуникации компонентов системы

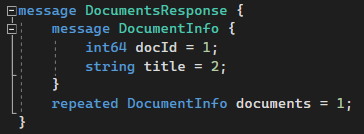
Для коммуникации клиента с сервером была выбрана система удаленного вызова процедур gRPC, т.к. она предоставляет все необходимые инструменты, которые позволяют надежно и эффективно выстраивать коммуникацию между несколькими серверами в контексте сервис-ориентированной архитектуры.

1. Способ передачи данных

Поскольку используется технология gRPC, которая является одной из реализаций удаленного вызова процедур, то способ передачи данных - нерезидентная синхронная связь с синхронизацией по ответу, так как ответ приходит клиенту только тогда, когда сообщение обработано на сервере, при этом сообщение нигде не сохраняется в отличие от очереди сообщений.

1. Структура передаваемых данных

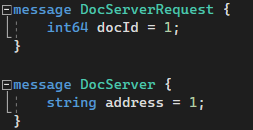
Так как методом коммуникации была выбрана система gRPC, то структуры передаваемых данных описывались в proto-файлах. Для получения информации о существующих файлах отправлялся запрос на сервер-хранилище, который возвращал список документов, где каждый документ имеет id документа и название (рис. 4).



Рисунок

4 – Список документов, возвращаемый сервером-хранилищем

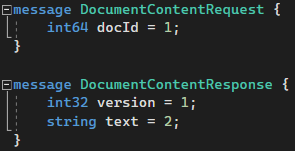
Для получения сервера документа, на сервер-диспетчер отправлялся запрос с нужным id, на который приходил ответом адрес сервера (рис. 5).



Рисунок

5 – Запрос и ответ сервера-диспетчера

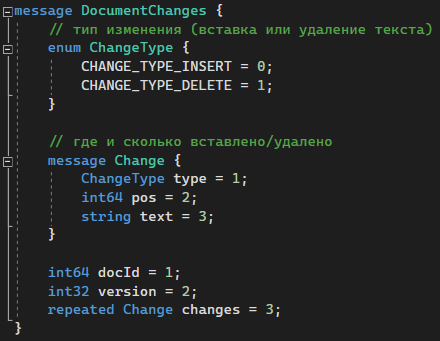
Для получения содержимого документа создавался запрос с id-документа, ответом на который было содержимое документа и его версия (рис. 6).



Рисунок

6 – Получение содержимого документа

Для отправки изменений на сервер создается запрос, в котором требуется указать id-документа, версию и список изменений, содержащий тип изменения, позицию и изменяемый текст. Данная структура представлена на рисунке 7.



Рисунок

7 – Структура изменений документа

1. Отказоустойчивость системы

При отказе диспетчера система не сможет предоставить клиентам возможность редактировать другие документы, однако позволит клиентам изменять текущие открытые документы. Если диспетчер будет восстановлен на другом узле сети, то система возобновит свою полную работоспособность. Для этого будет необходимо изменить вручную соответствующую А-запись доменного имени диспетчера в настройках DNS-сервера.

При отказе хранилища файлов система не сможет сохранить документы на диске. При восстановлении хранилища файлов на другом узле сети система снова позволит сохранить документы на диске, при этом содержимое текущих открытых документов не пропадает, так как оно хранится в актуальном состоянии на рабочих серверах. Сам сервер хранилища файлов уведомит диспетчера о своем появлении с указанием ему своего IP-адреса.

При отказе одного из нескольких рабочих серверов система будет успешно полностью функционировать. Может только пропасть актуальная версия содержимого открытых документов, если они не были своевременно сохранены пользователями. При этом для открытых документов клиент запросит у диспетчера другой из работающих серверов, если текущий сервер перестанет отвечать на запросы. При восстановлении рабочего сервера на другом узле сети, сам сервер уведомит диспетчера о своем появлении с указанием ему своего IP-адреса, таким образом пополнив пул доступных рабочих серверов.

При потере соединения с одним из серверов, клиент выдаст предупреждение о недоступности соответствующего сервера.

1. Исходный код приложения
   1. Клиент

MainWindow.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Data;

using System.Windows.Documents;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Imaging;

using System.Windows.Navigation;

using System.Windows.Shapes;

using System.Windows.Threading;

using Doc;

using Grpc.Core;

namespace Client

{

/// <summary>

/// Interaction logic for MainWindow.xaml

/// </summary>

public partial class MainWindow : Window

{

private long docId;

private CommunicateServer server;

private WordProcessing wordProcessing;

private DocumentContentResponse documentBackup;

private List<DocumentsResponse.Types.DocumentInfo> documents;

private DispatcherTimer timer;

public MainWindow()

{

InitializeComponent();

wordProcessing = new WordProcessing();

CreateTimer();

try

{

server = new CommunicateServer();

documents = server.GetDocuments();

foreach (var doc in documents)

{

documentsList.Items.Add(doc.Title);

}

}

catch(Exception ex)

{

MessageBox.Show(this, ex.Message, "Ошибка", MessageBoxButton.OK, MessageBoxImage.Error);

}

}

private void CreateTimer()

{

timer = new DispatcherTimer();

timer.Interval = TimeSpan.FromMilliseconds(1000);

timer.Tick += SyncDoc;

}

private void LoadDocumentFromServer(object sender, RoutedEventArgs e)

{

if (timer.IsEnabled)

timer.Stop();

try

{

var doc = documents.Where(z => z.Title == documentsList.SelectedItem.ToString()).First();

docId = doc.DocId;

var docServer = server.GetDocServer(docId);

server.ConnectToDocumentServer(docServer.Address);

var actualDoc = server.GetActualDocumentContent(docId);

documentBackup = new DocumentContentResponse()

{

Text = actualDoc.Text,

Version = actualDoc.Version

};

ChangeTextInRichTextBox(actualDoc.Text);

timer.Start();

}

catch (Exception ex)

{

MessageBox.Show(this, ex.Message, "Ошибка", MessageBoxButton.OK, MessageBoxImage.Error);

}

}

private void SyncDoc(object sender, EventArgs e)

{

try

{

SendDiffToServer();

ReceiveDiffFromServer();

}

catch (Exception ex)

{

MessageBox.Show(this, ex.Message, "Ошибка", MessageBoxButton.OK, MessageBoxImage.Error);

}

}

private void SendDiffToServer()

{

var newText = GetTextFromRichTextBox();

var oldText = documentBackup.Text;

var dataForSend = new DocumentChanges()

{

DocId = docId,

Version = documentBackup.Version

};

wordProcessing.Compare(oldText, newText, dataForSend);

if (dataForSend.Changes.Count != 0)

server.SendDocumentChanges(dataForSend);

}

private void ReceiveDiffFromServer()

{

DocumentChanges docChanges;

try

{

docChanges = server.GetDocumentChanges(docId, documentBackup.Version);

}

catch(GetDocumentChangesException)

{

LoadDocumentFromServer(this, new RoutedEventArgs());

docChanges = server.GetDocumentChanges(docId, documentBackup.Version);

}

if (docChanges.Changes.Count != 0)

{

var text = documentBackup.Text;

var newText = wordProcessing.ApplyTextChanges(text, docChanges);

docId = docChanges.DocId;

documentBackup.Version = docChanges.Version;

documentBackup.Text = newText;

ChangeTextInRichTextBox(newText);

}

}

private void ChangeTextInRichTextBox(string text)

{

var docChanges = new DocumentChanges();

wordProcessing.Compare(documentBackup.Text, text, docChanges);

int prevCursorPos = GetCaretPos();

int offset = prevCursorPos;

for (int i = 0; i < docChanges.Changes.Count; i++)

{

var change = docChanges.Changes[i];

if (change.Pos >= prevCursorPos)

continue;

if (change.Type == DocumentChanges.Types.ChangeType.Insert)

{

offset += change.Text.Length;

}

else if (change.Type == DocumentChanges.Types.ChangeType.Delete)

{

offset -= change.Text.Length;

}

}

docBox.Document.Blocks.Clear();

docBox.Document.Blocks.Add(new Paragraph(new Run(text)));

docBox.CaretPosition = docBox.Document.ContentStart;

int nl\_count = 0;

for (int i = 0; i < text.Length; ++i)

{

if (i >= offset)

break;

if (text[i] == '\n')

nl\_count++;

}

for (int i = 0; i < Math.Abs(offset) - nl\_count && i < text.Length - nl\_count; ++i)

docBox.CaretPosition = docBox.CaretPosition.GetNextInsertionPosition(offset > 0 ? LogicalDirection.Forward : LogicalDirection.Backward);

}

private int GetCaretPos()

{

TextPointer start = docBox.Document.ContentStart;

TextPointer caret = docBox.CaretPosition;

TextRange range = new TextRange(start, caret);

return range.Text.Length;

}

private string GetTextFromRichTextBox()

{

var text = new TextRange(docBox.Document.ContentStart, docBox.Document.ContentEnd).Text;

return text.Substring(0, text.Length - 2);

}

}

}

CommunicateServer.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Net;

using System.Text;

using System.Threading.Tasks;

using Doc;

using Grpc.Core;

namespace Client

{

class CommunicateServer

{

private Channel documentServerChannel;

private Channel storageChannel;

private Channel dispatcherChannel;

private DocumentService.DocumentServiceClient documentClient;

private DispatcherService.DispatcherServiceClient dispatcherClient;

private StorageService.StorageServiceClient storageClient;

public CommunicateServer()

{

var ips = Dns.GetHostAddresses("trrp1.mooo.com");

var ip = "localhost";

ip = ips[0].ToString();

try

{

storageChannel = new Channel(ip, 50052, ChannelCredentials.Insecure);

storageClient = new StorageService.StorageServiceClient(storageChannel);

dispatcherChannel = new Channel(ip, 50051, ChannelCredentials.Insecure);

dispatcherClient = new DispatcherService.DispatcherServiceClient(dispatcherChannel);

}

catch (RpcException e)

{

throw new Exception("Не удалось связаться с серверми диспечера и документов");

}

}

public List<DocumentsResponse.Types.DocumentInfo> GetDocuments()

{

List<DocumentsResponse.Types.DocumentInfo> documents;

try

{

var request = new DocumentsRequest();

documents = storageClient.GetDocuments(request).Documents.ToList();

}

catch (RpcException e)

{

throw new Exception("Сервер документов недоступен. Поторите попытку позже.");

}

return documents;

}

public DocServer GetDocServer(long docId)

{

DocServer serverResponse;

try

{

var serverRequest = new DocServerRequest() { DocId = docId };

serverResponse = dispatcherClient.GetDocServer(serverRequest);

}

catch (RpcException e)

{

throw new Exception("Ошибка получения адреса сервера документа. Проблемы с доступом к диспетчеру.");

}

return serverResponse;

}

public void ConnectToDocumentServer(string host)

{

try

{

documentServerChannel = new Channel(host, ChannelCredentials.Insecure);

documentClient = new DocumentService.DocumentServiceClient(documentServerChannel);

}

catch(RpcException e)

{

throw new Exception("Сервер документа недоступен");

}

}

public DocumentContentResponse GetActualDocumentContent(long docId)

{

DocumentContentResponse document;

try

{

var request = new DocumentContentRequest() { DocId = docId };

document = documentClient.GetActualDocumentContent(request);

}

catch (RpcException e)

{

throw new Exception("Ошибка получения актуальной версии документа");

}

return document;

}

public DocumentChanges GetDocumentChanges(long docId, int version)

{

DocumentChanges documentChanges;

try

{

var request = new DocumentChangesRequest() {

DocId = docId,

Version = version

};

documentChanges = documentClient.GetDocumentChanges(request);

}

catch (RpcException e)

{

if (StatusCode.NotFound == e.StatusCode)

{

throw new GetDocumentChangesException();

}

else if (StatusCode.Unavailable == e.StatusCode)

throw new Exception("Сервер документа недоступен. Проверьте соединение с сетью.");

else throw new Exception("Ошибка при получении изменений документа.");

}

return documentChanges;

}

public void SendDocumentChanges(DocumentChanges documentChanges)

{

try

{

documentClient.SendDocumentChanges(documentChanges);

}

catch (RpcException e)

{

throw new Exception("Ошибка отправки изменений. Проверьте соединение с сетью.");

}

}

}

}

WordProcessing.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using DiffMatchPatch;

using Doc;

namespace Client

{

class WordProcessing

{

private diff\_match\_patch dmp;

public WordProcessing()

{

dmp = new diff\_match\_patch();

}

public void Compare(string oldText, string newText, DocumentChanges changes)

{

var diff = dmp.diff\_main(oldText, newText);

dmp.diff\_cleanupSemantic(diff);

GenerateDifferences(diff, changes);

}

private void GenerateDifferences(List<Diff> diffs, DocumentChanges changes)

{

long position = 0;

foreach (var diff in diffs)

{

switch(diff.operation)

{

case Operation.INSERT:

changes.Changes.Add(new DocumentChanges.Types.Change()

{

Type = DocumentChanges.Types.ChangeType.Insert,

Pos = position,

Text = diff.text

});

position += diff.text.Length;

break;

case Operation.DELETE:

changes.Changes.Add(new DocumentChanges.Types.Change()

{

Type = DocumentChanges.Types.ChangeType.Delete,

Pos = position,

Text = diff.text

});

break;

case Operation.EQUAL:

position += diff.text.Length;

break;

}

}

}

public string ApplyTextChanges(string text, DocumentChanges changes)

{

var sb = new StringBuilder(text);

foreach (var change in changes.Changes)

{

switch (change.Type)

{

case DocumentChanges.Types.ChangeType.Insert:

sb.Insert((int)change.Pos, change.Text);

break;

case DocumentChanges.Types.ChangeType.Delete:

sb.Remove((int)change.Pos, change.Text.Length);

break;

}

}

return sb.ToString();

}

}

}

* 1. Сервер-хранилище
  2. Сервер-диспетчер
  3. Сервер-документа

поп