МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное бюджетное образовательное учреждение
высшего образования
«Тольяттинский государственный университет»

Институт математики, физики и информационных технологий
(наименование института полностью)

Кафедра «Прикладная математика и информатика»
(наименование кафедры)
02.03.03 Математическое обеспечение и администрирование информационных систем
(код и наименование направления подготовки, специальности)
Технология программирования
(направленность (профиль)/специализация)

БАКАЛАВРСКАЯ РАБОТА

на тему **Разработка распределённой вычислительной системы для обработки видеопотока**

Студент	Г.Р. Ахмеров	
	(И.О. Фамилия)	(личная подпись)
Руководитель	А.Б. Кузьмичев	
	(И.О. Фамилия)	(личная подпись)
Консультанты	К.А. Селиверстова	
	(И.О. Фамилия)	(личная подпись)

Допустить к защите

Заведующий кафедрой к.т.н, доцент, А.В. Очеповский _____
(ученая степень, звание, И.О. Фамилия) (личная подпись)

«____»_____20____г.

Тольятти 2019

АННОТАЦИЯ

Тема: «Разработка распределённой вычислительной системы для обработки видеопотока».

Цель работы: увеличение эффективности обработки видеопотока.

Объектом исследования являются распределенные вычислительные системы для обработки данных.

Предметом исследования являются модели и методы построения распределенных вычислительных системы для обработки данных.

В данной ВКР исследуются технологии создания распределённых вычислительных систем и производится разработка программной реализации распределённой вычислительной системы для обработки видеопотока.

Структура выпускной квалификационной работе представлена введением, тремя разделами, заключением, списком литературы.

Во введении описывается актуальность проводимого исследования, заключающаяся в том, что обработка видеопотока используется в популярных и развивающихся сегодня областях, таких как системы управления беспилотными автомобилями. В первом разделе проводится анализ существующих технологий построения распределённых систем и их сравнение.

Во втором разделе описывается проектирование распределённой вычислительной системы для обработки видеопотока и прогнозирование её производительности.

В третьем разделе представлена реализация распределённой вычислительной системы для обработки видеопотока и анализ её эффективности.

Был сделан вывод о том, что применение распределённой архитектуры действительно увеличивает эффективность обработки видеопотока, но степень ускорения зависит от сложности выполняемых преобразований, с увеличением эффективности при увеличении сложности преобразований.

Данная бакалаврская работа состоит из пояснительной записки на 44 стр., включая 15 рисунков, 6 таблиц, списка из 31 источника, в том числе 5 на иностранном языке и 1 приложения.

ABSTRACT

The topic of the given graduation work is «Development of a distributed computing system for video processing».

The aim of the work is to improve the efficiency of video processing.

The object of the graduation work is distributed computing systems for data processing.

The subject of the graduation work is the models and techniques of creating distributed computing systems for data processing.

The work touches upon models of distributed computing systems. We analyze the tasks of image and video processing and develop a software implementation of a distributed computing system for video processing.

The introduction describes the relevance of the study, namely the fact that the video and image processing techniques are used in such popular areas as automatic driving systems. In the first section we analyze and compare the existing technologies for creating distributed systems.

In the second section we describe the design of the distributed computing system for video processing and model its efficiency.

In the third section we implement the distributed computing system for video processing and analyze its efficiency.

All three parts look toward improving the effectiveness of video processing.

Overall, the results suggest that the usage of a distributed architecture actually improves the effectivenes of video processing, but the speedup is determined by the complexity of used transformations (the system was more efficient when more complex transformations were used).

This graduation work consist of an explonatory note on 44 pages, including 15 figures, 6 tables, the list of 31 references including 5 foreign sources and 1 appendix.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ.....5
1 ОБЗОР СОВРЕМЕННОГО СОСТОЯНИЯ ТЕХНОЛОГИЙ ПОСТРОЕНИЯ РАСПРЕДЕЛЁННЫХ ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМ.....7
1.1 Обзор технологий построения распределённых вычислительных систем.....7
1.2 Обзор моделей коммуникации в распределённых вычислительных системах.....8
1.3 Анализ модели акторов.....11
1.4 Анализ модели взаимодействующих последовательных процессов.....15
1.5 Сравнительный анализ и выделение недостатков моделей распределенных вычислительных систем.....17
1.6 Выводы и результаты по разделу 1.....19
2 ПРОЕКТИРОВАНИЕ РАСПРЕДЕЛЕННОЙ ВЫЧИСЛИТЕЛЬНОЙ СИСТЕМЫ ДЛЯ ОБРАБОТКИ ВИДЕОПОТОКА.....20
2.1 Прогнозирование производительности распределённой вычислительной системы для обработки видеопотока.....20
2.2 Проектирование системы обработки видеопотока в распределённой вычислительной системе.....23
2.2.1 Проектирование подсистемы балансировки нагрузки.....24
2.2.2 Проектирование подсистемы упорядоченного слияния.....25
2.3 Выводы и результаты по разделу 2.....25
3 РЕАЛИЗАЦИЯ РАСПРЕДЕЛЁННОЙ ВЫЧИСЛИТЕЛЬНОЙ СИСТЕМЫ ДЛЯ ОБРАБОТКИ ВИДЕОПОТОКА.....26
3.1 Выбор библиотеки для обработки видеопотока.....26
3.2 Выбор языка программирования для реализации распределённой вычислительной системы.....29
3.3 Разработка алгоритма обработки видеопотока в распределённой вычислительной системе.....31
3.4 Разработка алгоритма балансировки нагрузки.....32
3.5 Разработка алгоритма упорядоченного слияния.....35
3.6 Анализ эффективности разработанной распределённой вычислительной системы обработки видеопотока.....36
3.7 Выводы и результаты по разделу 3.....39
ЗАКЛЮЧЕНИЕ.....40
СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ.....41
ПРИЛОЖЕНИЕ А.....44

ВВЕДЕНИЕ

В данной работе рассматривается применение распределённой архитектуры для обработки видеопотока, с целью увеличения эффективности.

Под *видеопотоком* понимается последовательность отдельных кадров или изображений, которые расположены в определённом порядке. В ряде случаев, эти кадры можно обрабатывать независимо друг от друга. То есть, задачу можно свести к обработке изображений с сохранением порядка результатов обработки.

Алгоритмы *обработки изображений* получают на вход изображение — фотографию или кадр видео, а на выходе дают новое изображение или набор характеристик, связанных со входным изображением. Обычно, системы обработки изображений рассматривают изображения как двумерные массивы, и применяют к ним заранее заданные методы обработки.

В последнее время растёт необходимость в обработке видеопотоков. Беспилотные автомобили и летательные аппараты анализируют видеопотоки с камер для получения сведений об окружающей обстановке и ориентации в пространстве.

В системах автоматического управления транспортными средствами, повышены требования ко времени отклика и к надёжности системы: слишком высокая задержка может привести к аварии. В то же время, является высоким и количество обрабатываемой информации.

Появляется необходимость в создании систем с крайне высокой производительностью. Для этого применяют параллельные и распределенные архитектуры.

Одной из моделей, применяемых для описания параллельных и распределенных систем, является модель акторов. Акторы — это универсальные вычислительные примитивы. Они могут выполнять вычисления, и обмениваться сообщениями и создавать новых акторов. Каждый актор может существовать на отдельном компьютере, и наоборот, несколько акторов могут

выполняться на одном и том же. Таким образом, система акторов может быть распределена по компьютерной сети.

Целью данной работы работы является повышение эффективности обработки видеопотока. Для достижения поставленной цели необходимо решить следующие задачи:

1. Выбрать технологию построения распределённой вычислительной системы.
2. Спроектировать распределённую вычислительную систему для обработки видеопотока.
3. Реализовать распределённую вычислительную систему для обработки видеопотока.
4. Провести анализ эффективности реализованной распределённой вычислительной системы для обработки видеопотока.

1 ОБЗОР СОВРЕМЕННОГО СОСТОЯНИЯ ТЕХНОЛОГИЙ ПОСТРОЕНИЯ РАСПРЕДЕЛЁННЫХ ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМ

1.1 Обзор технологий построения распределённых вычислительных систем

В настоящее время многие системы, обрабатывающие значительные объёмы данных, являются распределёнными. *Распределенной вычислительной системой (РВС)* называется набор соединённых каналами связи независимых компьютеров, которые с точки зрения пользователя некоторого программного обеспечения выглядят единым целым [8]. Ключевыми моментами в этом определении являются: автономность узлов РВС и представление системы пользователем, как единого целого.

При описании РВС применяются следующие понятия:

– *Ресурсом* называется любая программная или аппаратная сущность, представленная или используемая в распределенной сети. Например, компьютер, устройство хранения, файл, коммуникационный канал, сервис и т.п.

– *Узел* — любое аппаратное устройство в распределенной вычислительной системе.

– *Сервер* — это поставщик информации в РВС (например, веб-сервер).

– *Клиент* — это потребитель информации в РВС (например, веб-браузер).

– *Пир* — это узел, сочетающий в себе клиентскую часть с серверной (т.е. является одновременно и поставщиком, и потребителем информации) [14].

Так как система должна обеспечивать пользователю видимость работы с единой вычислительной системой, но её узлы являются автономными элементами, система должна обладать следующими характеристиками:

– *возможность совместного использования ресурсов* — как аппаратных, так и программных;

– *параллельность* — несколько процессов РВС могут выполняться одновременно, при необходимости взаимодействуя между собой;

– *отказоустойчивость* — система должна продолжать работать при отказе одного или нескольких узлов;

– *масштабируемость* — способность системы справляться с увеличением рабочей нагрузки при добавлении ресурсов;

– *прозрачность для пользователя* — в то время как пользователю предоставляется возможность работать со всеми ресурсами системы, информация об их физическом местоположении от пользователя скрыта.

Вычислительные среды, состоящие из множества узлов на базе различных аппаратных и программных платформ, называют *гетерогенными* [14].

Для того, чтобы такое гетерогенное оборудование РВС работало как единое целое, стек ПО принято разбивать на два слоя. Верхний слой заключает в себе распределённые приложения, решающие определённые прикладные задачи средствами РВС. Второй слой служит базой для функциональных возможностей первого. Он называется *промежуточным программным обеспечением (ППО)* [14].

1.2 Обзор моделей коммуникации в распределённых вычислительных системах

Одним из ключевых моментов при построении РВС является выбор модели взаимодействия её узлов. Наиболее часто используемыми моделями являются обмен сообщениями (message passing), удалённый вызов процедур (remote procedure call — RPC) и распределённая разделяемая память (distributed shared memory — DSM). Рассмотрим каждую из моделей подробнее.

Обмен сообщениями возлагает ответственность за создание сообщения и указание конкретного получателя на разработчика.

При реализации системы обмена сообщениями, принимаются следующие решения:

– операции отправки и приёма сообщения могут быть *синхронными*, когда процесс останавливается и следующие операции не начинают выполняться до завершения отправки или приёма, или *асинхронными*, когда операция завершается сразу;

– приём сообщений может быть *буферизированным*, то есть с применением очереди, сохраняющей поступающие сообщения до тех пор, пока принимающий процесс не начнёт их обрабатывать;

– отправка может быть *ненадёжной*, когда отправитель не требует подтверждения о доставке сообщения и не отсылает сообщение повторно, чтобы гарантировать доставку. *Надёжная* отправка гарантирует доставку сообщения при завершении операции отправки. Обработка подтверждения доставки и повторная отправка выполняются автоматически.

Модель RPC повышает уровень абстракции, но основывается на обмене сообщениями. Эта модель позволяет использовать семантику вызова локальных процедур — при вызове процедуры, ей передаются аргументы, а вызывающий процесс блокируется до завершения выполнения процедуры. По завершении, процедура может вернуть результат. По сути, при этом происходит блокирующая отправка сообщения получателю, за которой следует приём обратного сообщения с результатом.

Распределённая разделяемая память — это память, распределённая по сети автономных компьютеров, но которую при этом можно использовать как централизованную. Доступ к памяти осуществляется через виртуальные адреса, благодаря чему процессы могут обмениваться информацией, читая и изменяя данные.

Распределённая разделяемая память должна обладать следующими свойствами:

– *согласованность* — для увеличения эффективности, данные в DSM могут быть продублированы на нескольких компьютерах. Но это требует поддержания согласованности данных;

– *синхронизация* — разделяемые данные должны защищаться с помощью замков, семафоров или мониторов.

Сравним вышеописанные модели коммуникаций в контексте РВС для обработки видеопотока. При обработке видеопотока, необходимо обработать каждый его кадр один раз. Так как наибольшая эффективность будет достигаться тогда, когда будет обрабатываться больше кадров видео одновременно, необходимо обрабатывать каждый кадр на отдельном узле.

Рассмотрим применение каждой модели коммуникаций для решения этой задачи. В случае с моделью обмена сообщениями, для обработки каждого кадра достаточно отправить два сообщения: одно с изначальным кадром, другое — с обработанным.

При использовании модели RPC, количество сообщений также равняется двум: одно для вызова процедуры и ещё одно для отправки результата обратно.

Когда применяется DSM, кадр для обработки сначала необходимо разместить в распределённой памяти, после чего оповестить через память процесс-обработчик. Этот процесс, затем, должен обработать видео и оповестить о завершении обработки вызывающий процесс через память. Таким образом, при использовании DSM, получающий процесс начнёт выполнять обработку только после того, как сам проверит содержимое распределённой памяти, что увеличивает задержку.

В случае с моделью обмена сообщениями и моделью RPC, обработка сообщения или запроса будет происходить сразу, как только придёт сообщение или запрос. Однако, использование RPC означает синхронную отправку сообщений, в то время как более низкоуровневая модель обмена сообщениями даёт больше контроля, позволяя отправлять сообщения асинхронно.

Исходя из этих особенностей моделей, можно сделать вывод о том, что самой эффективной из них является модель обмена сообщениями. Поэтому, для разработки распределённой вычислительной системы для обработки видеопотока будет применяться именно эта модель.

1.3 Анализ модели акторов

Модель акторов — это математическая модель параллельных вычислений. Актор является фундаментальным вычислительным примитивом, который заключает в себе три способности:

– обработка информации;

– хранение информации;

– общение.

Акторы — это основные и единственные сущности в данной модели, из чего следует, что всё является актором. Не может существовать системы из одного актора, поскольку актор должен общаться с другими акторами. Акторы взаимодействуют только посредством отправки друг другу сообщений, которые также являются акторами. На рисунке 1 представлен пример системы из четырёх акторов.

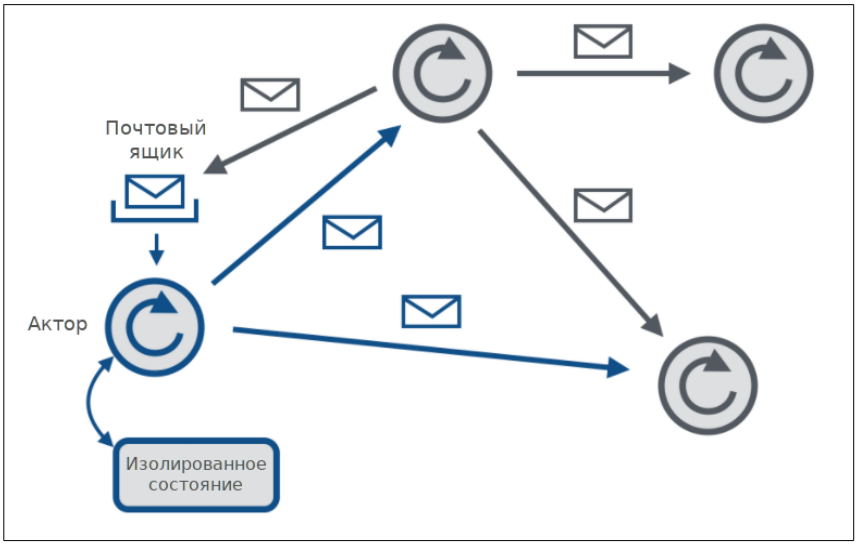

Рисунок 1 – Система из четырёх акторов

Существует некоторое сходство между моделью акторов и объектно-ориентированным программированием — в последнем, все является объектом, так же используется отправка сообщений, и существует локальное изменяемое состояние. Важным отличием модели акторов от ООП является полное отсутствие возможности изменять состояние других акторов напрямую. Также, актор не может послать другому актору изменяемую ссылку на данные. Актор может лишь посылать сообщения. Акторы — независимые сущности, и модель не привязывает их к определённому процессу или потоку. Они обладают способностью общаться, находясь в разных процессах или компьютерах. Кратко, можно сказать, что ООП описывает состояние и поведение, в то время как модель акторов описывает вычисления и их точную, законченную, формальную модель.

Актор, которому отправляются сообщения, называется получателем или целью. Так, единственным видом событий в этой модели является приём сообщения целью. События — дискретные шаги в продолжительной истории вычислений; они являются фундаментальными единицами взаимодействиями в теории модели акторов. Каждое событие E состоит из приема актора-сообщения, называемого $\text{message}(E)$, актором-целью (актором-получателем), называемого $\text{target}(E)$.

В момент получения сообщения, актор-получатель проявляет некоторое поведение, отправляя сообщения другим акторам. Одним из проявлений поведения актора является создание им других акторов. Действительно, почти все из акторов-сообщений создаются перед отправкой актору-получателю. Для каждого актора A, должно существовать уникальное событие $\text{Birth}(A)$, во время которого A появляется впервые. Более точно, можно сказать, что $\text{Birth}(A)$ имеет такое свойство, что если A участвует в другом событии E, тогда $\text{Birth}(X) \rightarrow E$, где «$\rightarrow$» — бинарное отношение между событиями, означающее частичную упорядоченность событий.

Каждый актор обладает адресом. В разных реализациях, адресом может являться прямой физический адрес (такой как MAC-адрес), email, адрес в памяти, некоторый идентификатор, и тому подобное. Несколько акторов могут иметь один адрес, и один актор может иметь несколько адресов. Отношение между акторами и адресами задано в виде связи «многие-ко-многим». Адрес не является уникальным идентификатором актора. Акторы не имеют конкретной сущности (identity), только адрес. Поэтому, если взглянуть на нашу теоретическую модель акторов «сверху», мы увидим лишь адреса. Невозможно сказать, сколько за одним адресом стоит акторов — один или больше, поскольку за этим адресом может скрываться актор-прокси, пересылающий сообщения другим акторам. Всё что мы можем сделать с адресом — отправить на него сообщение. Адреса, в модели акторов, представляют собой возможности. Соотнесение адресов с акторами не является частью теоретической модели акторов, хотя оно и входит в реализации.

Актор может послать сообщение самому себе (поддержка рекурсии). Это сообщение будет принято и обработано позже. Также, у актора может существовать почтовый ящик, который тоже является актором, и представляет собой пункт назначения сообщений и их хранилище. Акторы не обязаны иметь почтовые ящики, поскольку в данном случае каждый почтовый ящик тоже будет должен обладать почтовым ящиком, что приведёт к бесконечной рекурсии.

В модели акторов, существует две аксиомы — организационная и операционная.

Операционная аксиома. В ответ на сообщение, актор может совершить лишь действия из следующего списка:

– создать конечное количество новых акторов;

– отправить конечное количество сообщений другим акторам, адреса которых ему уже известны;

– подготовить своё локальное хранилище для приёма следующего сообщения (определить, что нужно делать с этим сообщением).

Организационная аксиома. Локальное хранилище актора может включать лишь адреса, которые:

– были предоставлены при создании актора;

– были приняты в сообщениях;

– принадлежат акторам, которые были созданы данным актором (первый параграф операционной аксиомы).

Теоретически, актор обрабатывает входящие сообщения друг за другом, по одному за раз. Но физические реализации часто каким-либо образом оптимизируют процесс обработки сообщений. Модель акторов не постулирует практически никаких гарантий, связанных с отправкой и обработкой сообщений. Различные физические реализации вольны добавлять свои интересные возможности. Так, Erlang позволяет сопоставлять сообщения с образцом (pattern matching).

Модель акторов также поддерживает делегирование, поскольку акторы могут отправлять адреса других акторов в сообщении.

Как было уже упомянуто выше, актор может отправлять сообщения самому себе. Модель акторов позволяет избежать при этом блокировки (deadlock), используя *future*. Future — это особый вид актора-сообщения. Он может хранить все ещё вычисляемый в данный момент результат. Другими словами, он хранит «будущий» результат вычисления (возможно, довольно долгого). Актор может послать future другим акторам, и самому себе.

Модель акторов не даёт никаких гарантий по поводу порядка доставки сообщений. Также, сообщения могут быть отброшены или утеряны. Такое поведение называется «негарантированной доставкой» («best-effort delivery»). Сообщения могут быть сохранены в хранилище (второй параграф определения актора — хранилище), и могут быть отправлены повторно. Сообщения могут быть доставлены не больше одного раза (один или ноль раз). Большое количество одновременно поступающих сообщений может послужить причиной отказа в обслуживании для актора. Другими словами, актор будет

неспособен обработать входящий поток сообщений. Чтобы смягчить данную проблему, можно использовать актора-почтовый ящик, который принимает сообщения и хранит их до тех пор, пока основной актор не сможет их обработать. Доставка сообщений в почтовый ящик получателя может занять сколь угодно долгое время. В физических реализациях модели акторов, операции добавления и извлечения сообщений из почтового ящика являются атомарными, что исключает появление состояния гонки.

Между акторами, не существует никаких каналов или других промежуточных сущностей. Отправка сообщений происходит напрямую другим акторам. Конечно, можно реализовать канал в виде актора, обрабатывающего сообщения «put» и «get», но в этом нет никакой необходимости.

В модели акторов, вычисления являются распределенными в пространстве, где вычислительные устройства, называемые акторами, асинхронно общаются, используя адреса друг друга. При этом, вычисление целиком не находится в каком-либо вполне определённом состоянии. Локальное состояние актора определено лишь в момент получения им сообщения, а в другое время оно может быть неопределённым.

Модель акторов, по сути, является довольно абстрактной теоретической моделью. Поэтому, любая реализация, основанная на вышеупомянутых аксиомах, является верной.

1.4 Анализ модели взаимодействующих последовательных процессов

Модель взаимодействующих последовательных процессов (communicating sequential processes — CSP) очень похожа на модель акторов. Решение также строится из набора автономных сущностей, каждая из которых обладает своим собственным личным состоянием и взаимодействует с другими сущностями только посредством сообщений. Сущности эти в модели CSP называются «процессы».

Процессы в модели CSP являются легковесными, без какого либо распараллеливания своей работы. При необходимости распараллелить решение какой-либо задачи, достаточно запустить несколько CSP-процессов, каждый из которых выполняется последовательно.

Взаимодействуют CSP-процессы друг с другом через синхронные сообщения, но сообщения отсылаются не в почтовые ящики, как в модели акторов, а в каналы. Каналы можно рассматривать как очереди сообщений, как правило, фиксированного размера.

В отличии от модели акторов, где для каждого актора автоматически создаётся почтовый ящик, каналы в CSP должны создаваться явно. И если появляется необходимость в том, чтобы два процесса взаимодействовали друг с другом, для этого будет нужно сначала создать канал, затем передать его первому процессу, который будет писать в канал, после чего передать второму процессу, который будет из канала читать.

При этом у каналов есть, как минимум, две операции, которые нужно вызывать явным образом. Во-первых, это операция write (send) для записи сообщения в канал.

Во-вторых, это операция read (receive) для чтения сообщения из канала. И необходимость явным образом вызывать read/receive отличает CSP от модели акторов, так как в случае с акторами операция read/receive может быть вообще скрыта от актора. То есть, реализация модели акторов может извлекать сообщения из очереди актора и вызывать обработчик для извлечённого сообщения.

CSP-процесс же должен сам выбрать момент для вызова read/receive, затем он должен определить, что за сообщение он получил и выполнить обработку извлечённого сообщения.

Внутри приложения CSP-процессы могут быть реализованы по-разному:

– CSP-процесс может быть реализован отдельным потоком ОС. Получается дорогое решение, но зато с вытесняющей многозадачностью;

– CSP-процесс может быть реализован сопрограммой (stackful coroutine, fiber, green thread). Это намного дешевле, но многозадачность только кооперативная.

1.5 Сравнительный анализ и выделение недостатков моделей распределенных вычислительных систем

Рассмотрим каждую модель более подробно.

Многопоточная модель основана на идее нескольких потоков с общей памятью, которые могут выполняться параллельно. Эта модель имеет множество проблем: взаимные блокировки, плохая масштабируемость, разделяемое состояние. Эти проблемы связаны с синхронизацией и их можно решить с помощью таких инструментов как семафоры и мьютексы. Но разделяемое состояние, когда одна и та же область памяти изменяется несколькими потоками, ведёт ко множеству новых проблем и ошибок.

Другими словами, многопоточная модель слишком ненадёжна и неэффективна. Поэтому, в дальнейшем сравнении будем рассматривать только модель акторов и CSP.

Обе эти модели основаны на идее обмена сообщениями: отправитель посылает сообщение, а получатель его принимает. В обоих моделях, получатель является синхронным: когда он готов принять новое сообщение, он делает вызов, блокирующий до появления сообщения.

Модель CSP полностью синхронна. Процесс, отправляющий данные в канал, блокируется до тех пор, пока читающий процесс не получит все данные. Преимущество такого основанного на блокировках механизма заключается в том, что канал может содержать не более одного сообщения, а это не только упрощает реализацию модели, но и избавляет от лишних проблем.

В модели акторов же, отправка сообщений происходит асинхронно. Отправитель не блокируется, вне зависимости от того, готов ли получатель принять новое сообщение или нет. Вместо блокировки, сообщение будет помещено

в очередь, которую обычно называют «почтовым ящиком». Это удобно, но может вызвать дополнительные вопросы: например, насколько большим должен быть почтовый ящик, или же стоит сделать его размер неограниченным?

В модели CSP, процессы взаимодействуют с помощью каналов. Процессы могут создавать каналы и обращаться с ними как с любыми другими объектами — например, можно передать канал в качестве аргумента в функцию. Акторы же имеют адреса, ровно по одному на каждого.

В CSP, сообщения приходят в том же порядке, в каком они были отправлены. В модели акторов, сообщения могут быть доставлены в любом порядке. К тому же, некоторые сообщения могут быть не доставлены вовсе.

Вышеописанное сравнение кратко представлено в виде таблицы 1.

Таблица 1 – Сравнение модели акторов и CSP

Характеристика	Модель акторов	CSP
Отправка сообщений	асинхронно	синхронно
Приём сообщений	синхронно	синхронно
Способ взаимодействия	адреса	каналы

Основным недостатком модели CSP является синхронная отправка сообщений, поскольку отправляющий процесс будет простаивать до тех пор, пока принимающий процесс не начнёт приём сообщения.

Основным недостатком модели акторов является то, что некоторые сообщения могут быть не доставлены. В таком случае, сообщение будет необходимо отправить повторно. Это может привести простою обоих сторон.

Тем не менее, отправка сообщений происходит чаще, чем потеря сообщений. Поэтому, наибольшую эффективность следует ожидать от модели акторов, которая и будет использоваться для реализации распределённой вычислительной системы для обработки видеопотока.

1.6 Выводы и результаты по разделу 1

В первом подразделе были рассмотрены современные технологии построения распределённых вычислительных систем.

Во втором подразделе были рассмотрены и сравнены модели коммуникации, применяемые распределённых вычислительных системах. Была выбрана модель обмена сообщениями.

Во третьем и четвёртом подразделах проведён анализ двух моделей параллелизма, основывающихся на обмене сообщениями — модели акторов и модели взаимодействующих последовательных процессов.

В пятом подразделе проведён сравнительный анализ и выделение недостатков этих моделей. Был сделан вывод о том, что модель акторов более эффективна. Поэтому, для реализации распределённой вычислительной системы для обработки видеопотока будет использоваться модель акторов.

2 ПРОЕКТИРОВАНИЕ РАСПРЕДЕЛЕННОЙ ВЫЧИСЛИТЕЛЬНОЙ СИСТЕМЫ ДЛЯ ОБРАБОТКИ ВИДЕОПОТОКА

2.1 Прогнозирование производительности распределённой вычислительной системы для обработки видеопотока

Производительность, один из главных показателей эффективности вычислительной системы, определяет её вычислительную мощность через количество той или иной вычислительной «работы», выполняемой системой в единицу времени [27].

При оценке производительности, важно выбрать определённую меру производительности. Поскольку описываемая в данной работе система имеет дело с кадрами видео, в качестве меры производительности было выбрано количество обрабатываемых системой кадров в секунду — FPS.

Существует множество математических моделей, описывающих зависимость производительности от количества используемых ресурсов, в частности — количества процессоров [27][30].

В идеальном случае, при увеличении количества вычислителей в n раз, производительность повышается также в n раз [27]. Это — линейная модель. Такого ускорения можно было бы ожидать, если бы решаемая задача не содержала частей, которые необходимо вычислять последовательно. Однако, решаемые на практике задачи обычно имеют хотя бы небольшую последовательную часть.

Для оценки производительности с учётом части задачи, которую нельзя распараллелить, применяется закон Амдала [30]. Ускорение описывается следующей формулой:

$$S(n) = \frac{1}{\alpha + \frac{p}{n}}, \text{ где} \tag{1}$$

$S(n)$ — теоретическое ускорение выполнения задачи;

α — последовательная часть задачи;

p — параллельная часть задачи ($p = 1 - \alpha$);

n — количество процессоров.

Ускорение при различных p показано на рисунке 2.

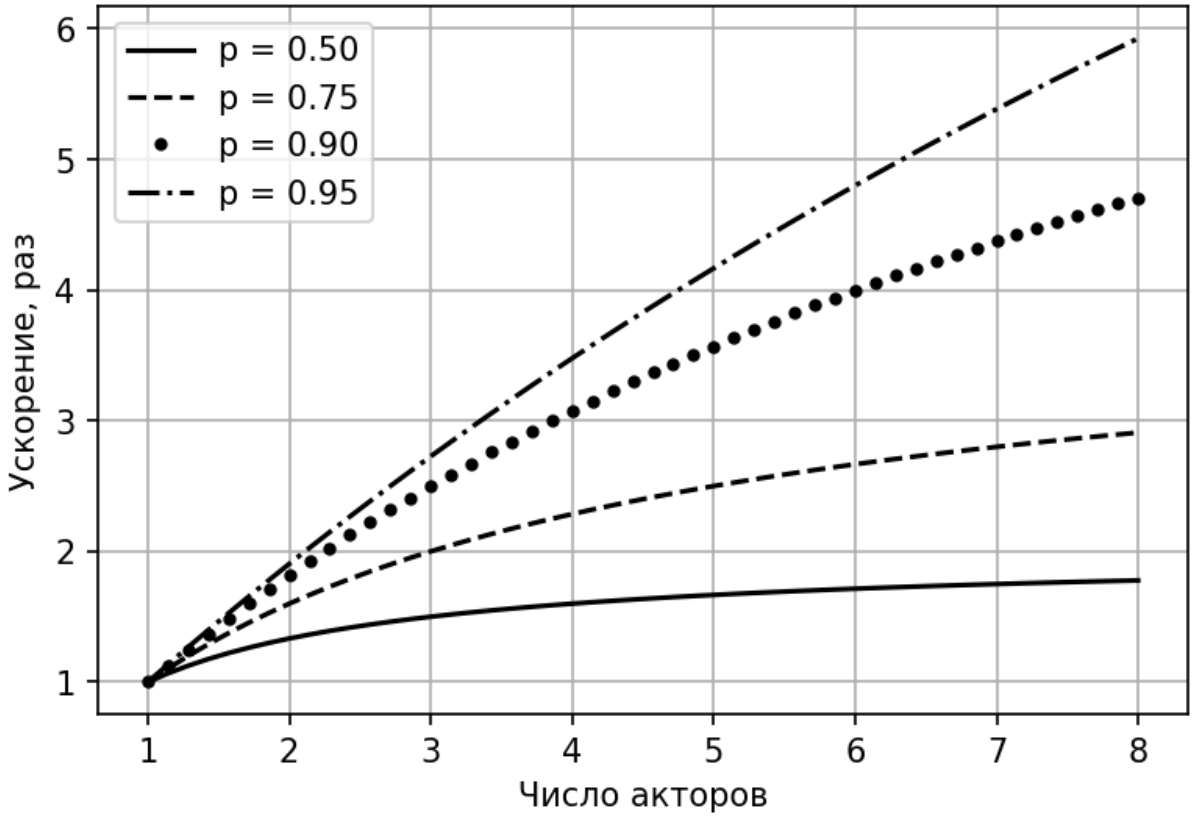

Рисунок 2 – Ускорение по закону Амдала

Закон Амдала говорит о том, что ускорение ограничено временем выполнения последовательной части задачи [30]. Другими словами,

$$\lim_{n \to \infty} S(n) = \frac{1}{\alpha}. \quad (2)$$

Тем не менее, закон Амдала не учитывает ещё одного важного момента: реальные параллельные и распределенные вычислительные системы также тратят время либо на согласование при использовании разделяемых данных, либо на пересылку сообщений, если данные не разделяются.

Универсальный закон масштабируемости (УЗМ, англ. universal scalability law, USL) позволяет оценить производительность с учётом издержек на

общение между процессами [27]. Ускорение описывается следующей формулой:

$$S(n) = \frac{n}{1 + \alpha(n-1) + \beta n(n-1)}, \text{ где} \quad (3)$$

$S(n)$ — теоретическое ускорение выполнения задачи;

α — последовательная часть задачи;

β — задержка из-за общения между процессами;

n — количество процессоров.

Ускорение по универсальному закону масштабирования показано на рисунке 3.

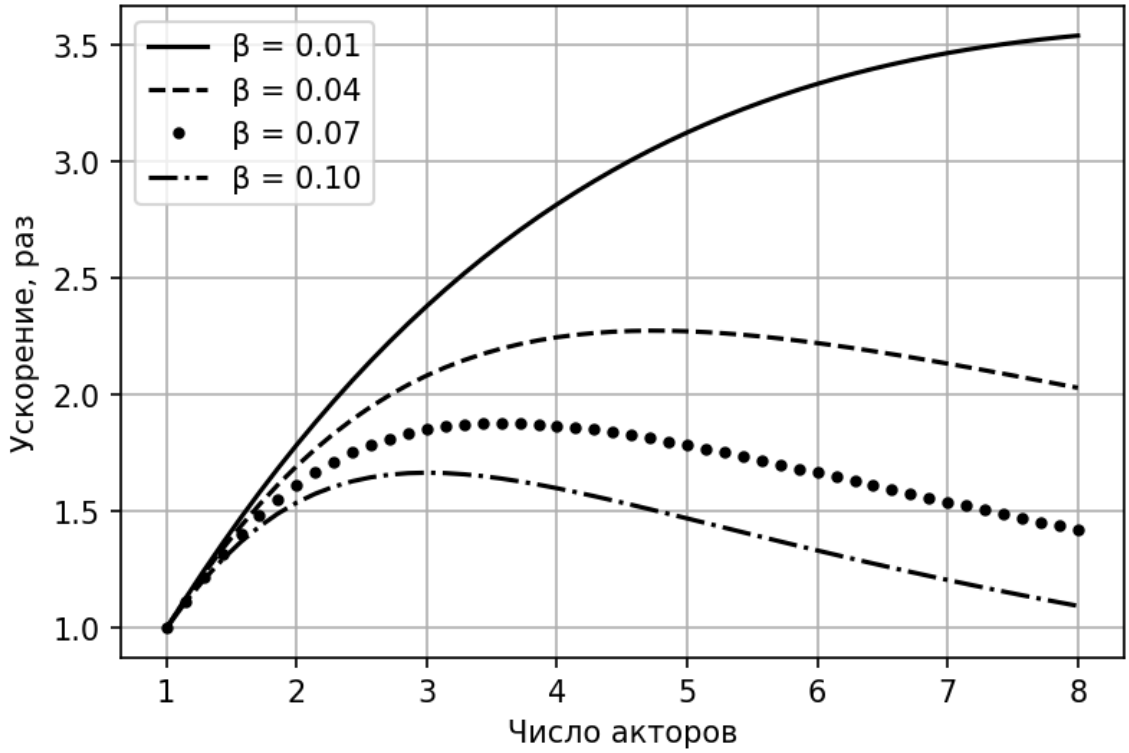

Рисунок 3 – Ускорение по УЗМ ($\alpha = 0.1$)

Сравнение вышеописанных моделей ускорения представлено на рисунке 4.

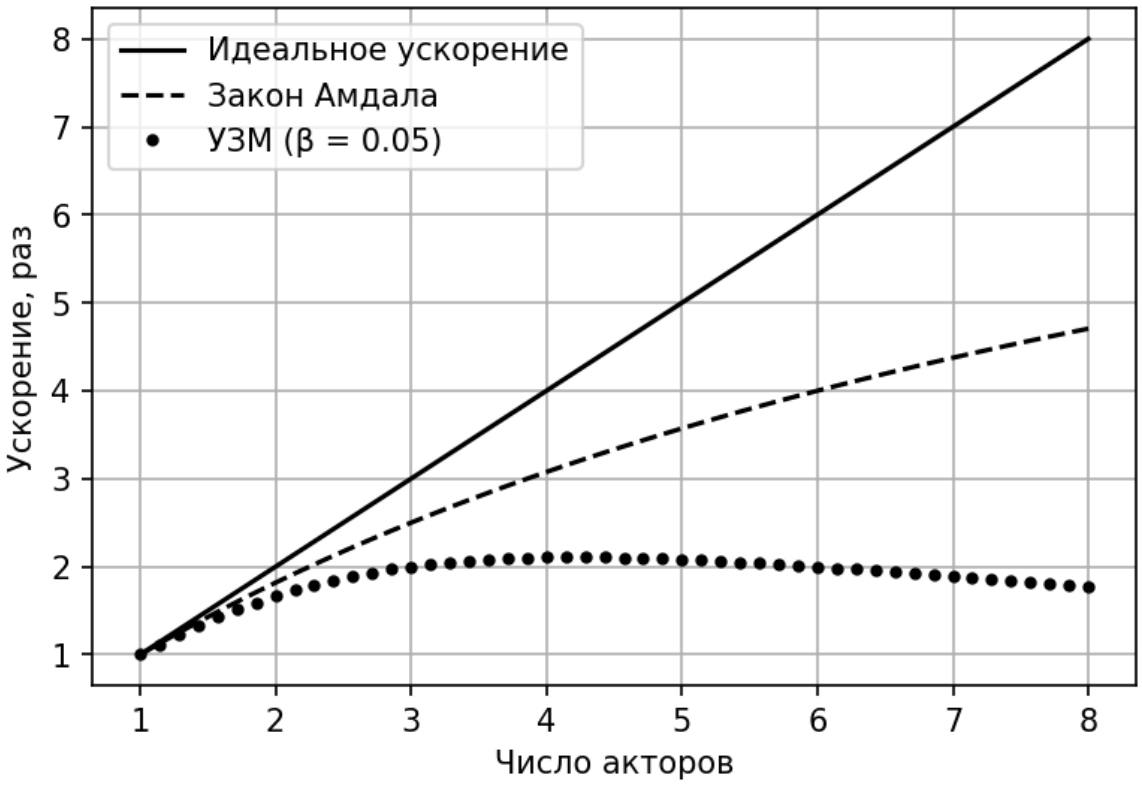

Рисунок 4 – Сравнение моделей ускорения ($\alpha = 0.1$)

При анализе эффективности РВС мы проверим, какому закону будет подчиняться её производительность.

2.2 Проектирование системы обработки видеопотока в распределённой вычислительной системе

Обработка видеопотока работает на уровне кадров. Каждый кадр обрабатывается отдельно от остальных. Благодаря этой независимости, несколько кадров могут обрабатываться одновременно.

Для каждого кадра из потока, система выбирает актора-обработчика с помощью подсистемы балансировки нагрузки. Этот блок обеспечивает равномерную загрузку системы и необходим для оптимизации её эффективности. Далее, кадр асинхронно отправляется выбранному актору-обработчику, который выполняет заранее заданные преобразования. Затем, обработчик возвращает обработанный кадр, который теперь попадает в

подсистему упорядоченного слияния. Дело в том, что обработчики могут работать с разной скоростью, что вызывает нарушение порядка среди обработанных кадров. Эта подсистема восстанавливает порядок кадров.

Далее, подсистемы балансировки нагрузки и упорядоченного слияния будут рассмотрены подробнее.

2.2.1 Проектирование подсистемы балансировки нагрузки

С целью достижения наивысшей производительности, распределенная система должна максимизировать объем одновременно обрабатываемых данных. Для этого, необходимо обеспечить максимальную загрузку каждого узла системы, то есть минимизировать простой узлов. Для решения этой проблемы применяются алгоритмы балансировки нагрузки, которые распределяют задания между узлами. Балансировкой нагрузки занимается отдельное устройство или программа, называемое *балансировщиком*.

Разрабатываемая программа будет распределять поток кадров акторам-обработчикам. На рисунке 5 представлен процесс распределения кадров четырём акторам.

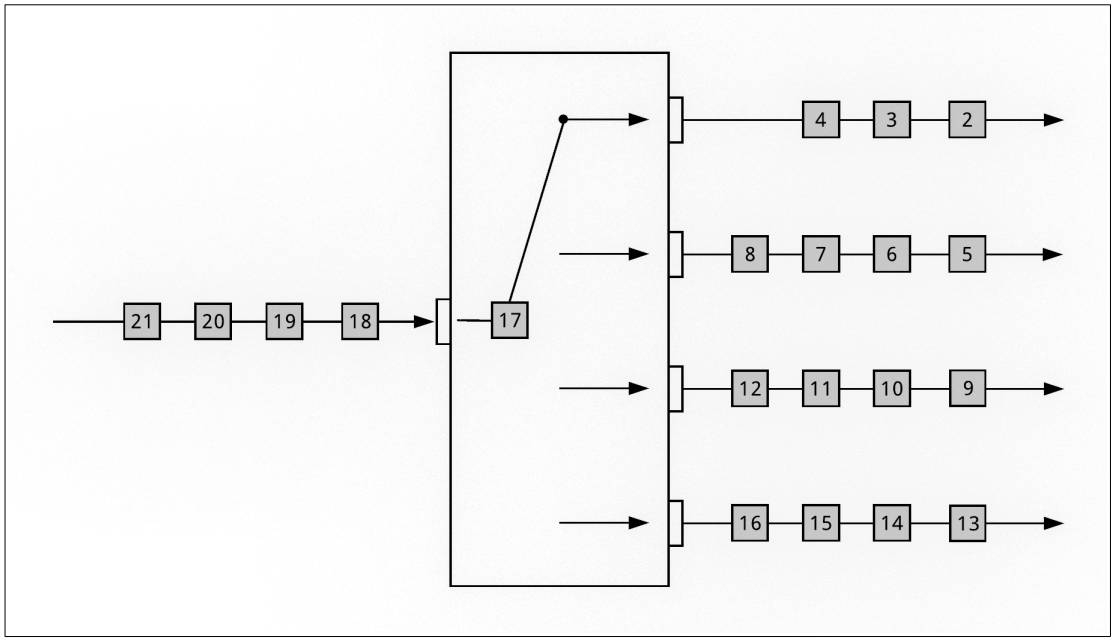

Рисунок 5 – Распределение потока кадров по потокам, обрабатываемым отдельными акторами

2.2.2 Проектирование подсистемы упорядоченного слияния

Так как несколько кадров обрабатываются одновременно и обработка любого кадра может быть закончена раньше остальных, порядок обработанных кадров нарушается. Поэтому, выходной поток также должен быть упорядочен.

Для восстановления порядка кадров, после обработки необходимо выполнять упорядоченное слияние. Оно может быть достигнуто с помощью добавления задержки до появления следующего по порядку кадра, в то время как остальные кадры должны помещаться во временный буфер. Пример упорядоченного слияния кадров из трёх потоков показан на рисунке 6.

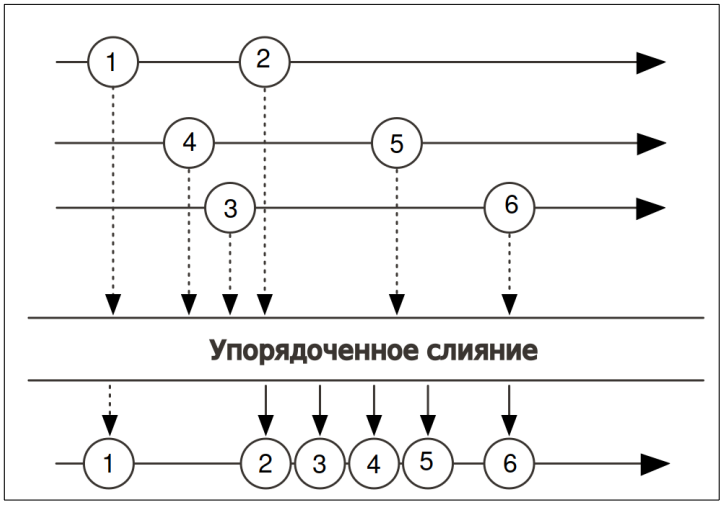

Рисунок 6 – Упорядоченное слияние

2.3 Выводы и результаты по разделу 2

В первом подразделе были рассмотрены и сравнены модели ускорения производительности: закон Амдала и универсальный закон масштабируемости.

Во втором подразделе описывается проектирование РВС для обработки видеопотока: взаимосвязь компонентов системы и её работа в целом. В первом пункте описывается балансировка нагрузки, необходимая для обеспечения эффективности системы. Во втором пункте рассмотрено упорядоченное слияние, необходимое для восстановления порядка обработанных кадров.

Полученные результаты необходимы для реализации РВС для обработки видеопотока, которая рассматривается в следующем разделе.

3 РЕАЛИЗАЦИЯ РАСПРЕДЕЛЁННОЙ ВЫЧИСЛИТЕЛЬНОЙ СИСТЕМЫ ДЛЯ ОБРАБОТКИ ВИДЕОПОТОКА

3.1 Выбор библиотеки для обработки видеопотока

В настоящий момент, существует несколько известных полнофункциональных библиотек для обработки изображений (как уже говорилось ранее, задача обработки видео во многих случаях сводится к задаче обработки изображений): OpenCV [19], JavaCV [20], AForge.NET [22].

При выборе библиотеки, стоит обратить особое внимание на используемое ей лицензионное соглашение, поскольку оно описывает, на каких условиях возможно использование библиотеки. В таблице 2 представлено сравнение лицензионных соглашений библиотек.

Таблица 2 – Сравнение лицензий библиотек для обработки изображений

Библиотека	Лицензия	Краткое содержание
OpenCV	BSD [19]	Библиотеку можно свободно использовать, при условии что реализуемое приложение будет содержать файл лицензии BSD.
JavaCV	GPLv2 [20]	Приложение должно иметь открытый исходный код и быть свободным, а также иметь лицензию GPL.
	Apache [20]	Библиотеку можно свободно использовать, при условии что реализуемое приложение будет содержать файл лицензии Apache.
AForge.NET	LGPLv3 [22]	Исходный код приложения может быть закрытым, если не изменять исходный код библиотеки.

Наличие понятной и подробной документации, примеров и книг существенно упрощает использование библиотеки. OpenCV, выпущенная в январе 1999 года, за время своего существования была использована во множестве приложений [19]. Поэтому, найти документацию или подходящий пример значительно проще. JavaCV — это обёртка над OpenCV для Java [20]. Примеров для неё значительно меньше. Тем не менее, библиотека очень похожа на OpenCV, что позволяет использовать примеры её примеры и документацию [21]. Фреймворк компьютерного зрения AForge.NET не связан с OpenCV, но

предоставляет большое множество примеров, демонстрирующих использование [22]. В таблице 3 представлено сравнение состояния документации библиотек.

Таблица 3 – Сравнение состояния документации библиотек

Библиотека	Книги	Сайт	Документация	Примеры
OpenCV	много [19]	многоязычный [19]	многоязычная [19]	много [19]
JavaCV	мало [20]	англоязычный [20]	англоязычная [21]	много [20]
AForge.NET	мало [22]	англоязычный [22]	англоязычная [22]	много [22]

Большинство функций OpenCV используют стиль языка C, но некоторые из них реализованы в виде класса C++. После версии 2.0, количество функций, реализованных в виде классов C++ увеличилось [19]. JavaCV — это обёртка над OpenCV для Java, предоставляющая большинство возможностей OpenCV [21]. AForge.NET — это библиотека, целиком реализованная на платформе .NET [22], благодаря чему она более удобна для пользователей этой платформы. Конечно, простота использования зависит и от способностей и опыта пользователя. Если ему привычнее C и C++, OpenCV может также оказаться удобнее.

Библиотека обработки изображений может делать множество различных преобразований. Рассмотрим две функции: простое преобразование — бинарный фильтр, и более сложное — распознавание лиц (в случае с AForge.NET, для распознавания лиц дополнительно применялась библиотека Accord.NET). Замеры производились десять раз с вычислением среднего времени выполнения. Результаты представлены в таблице 4.

Таблица 4 – Сравнение производительности библиотек обработки изображений

Библиотека	Бинаризация (мс)	Распознавание лиц (мс)
OpenCV	1.15	219.44
JavaCV	1.62	232.62
AForge.NET	3.31	389.79

Графически, сравнение производительности библиотек представлено на рисунке 7.

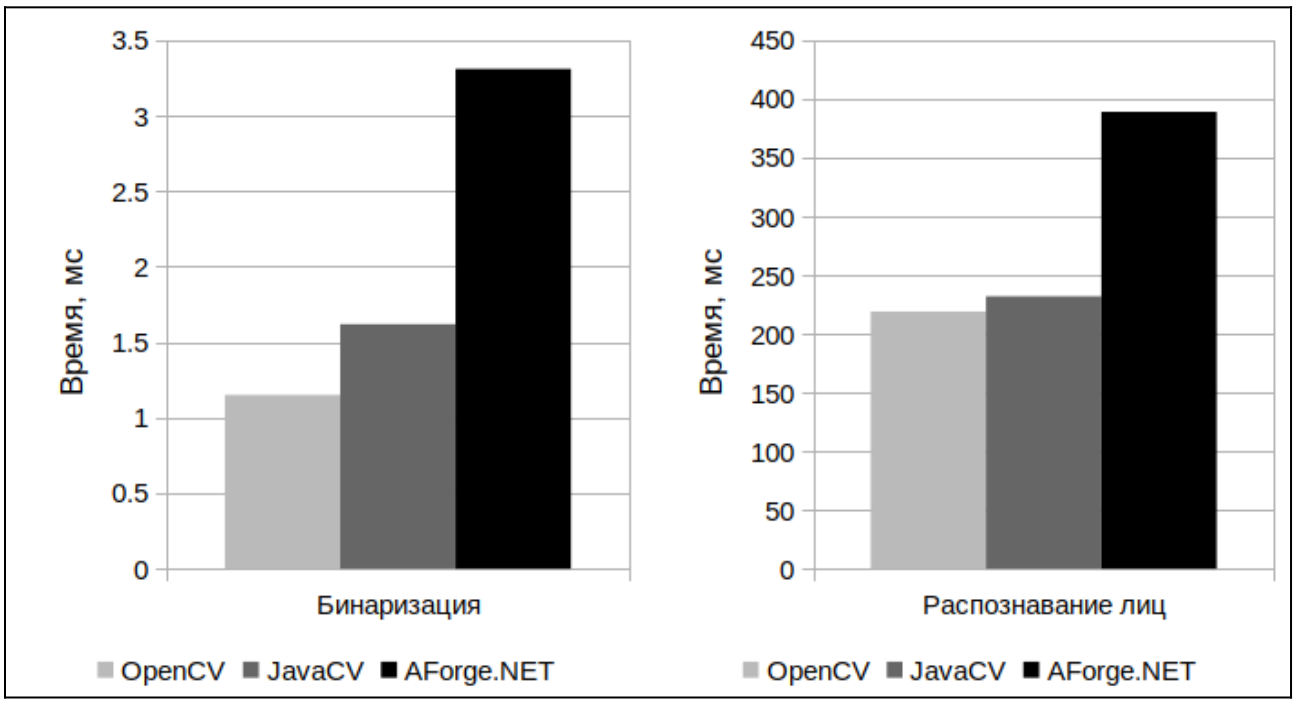

Рисунок 7 – Сравнение производительности библиотек обработки изображений

Замеры производились на ноутбуке с процессором AMD A8-6410. Очевидным выводом является, что OpenCV является самой быстрой. Производительность JavaCV оказалась чуть хуже, но разница минимальна. AForge.NET оказалась значительно медленнее.

Подведём итоги. Для этого, полученные результаты были приведены к пятибальной шкале и сведены в таблицу 5.

Таблица 5 – Сравнение характеристик библиотек обработки изображений

Характеристика	OpenCV	JavaCV	AForge.NET
Лицензия	5	5	4
Документация	5	3	4
Простота использования	3	5	5
Производительность	5	5	3
Итого	18	18	16

Исходя из результатов сравнения, трудно выделить одну библиотеку для обработки изображений. Проведём сравнение языков программирования и выберем между OpenCV и JavaCV исходя из полученных результатов.

3.2 Выбор языка программирования для реализации распределённой вычислительной системы

Перед реализацией распределённой вычислительной системы, необходимо выбрать язык программирования, с помощью которого она будет реализована. Библиотека OpenCV доступна для использования в следующих языках программирования [19]:

– C++ — статически типизированный мультипарадигмальный язык [31];

– Java — статически типизированный объектно-ориентированный язык [13];

– Scala — статически типизированный язык с упором на объектно-ориентированное и функциональное программирование [12].

При выборе языка программирования для реализации системы обработки изображений, необходимо руководствоваться следующими их характеристиками:

– **надёжность** языка уменьшает количество ошибок при написании кода;

– **быстродействие** — в случае языка, средний по важности показатель, поскольку основная работа будет выполняться с помощью высокопроизводительной библиотеки OpenCV;

– **гибкость** языка определяет, насколько легко выражать на нём действия, необходимые для решения задач;

– **скорость разработки**. На неё влияет время, затрачиваемое на сборку, запуск и отладку.

Оценим каждый из вышеописанных языков по выбранным характеристикам.

C++ является статически, но нестрого типизированным языком и поощряет мутабельность [31], что делает его средним по надёжности. Быстродействие очень высокое [15]. Для написания гибкого кода, приходится жертвовать либо его простотой, либо безопасностью, что говорит о средней

гибкости. Время сборки может увеличиваться очень быстро, особенно при активном использовании шаблонов, но готовые исполняемые файлы запускаются очень быстро [15]. Скорость разработки выше среднего [15].

Java является статически и строго типизированным языком, поощряющим мутабельность [13] — надёжность выше среднего. Быстродействие обычно ниже быстродействия C++, но в некоторых случаях сопоставимо с ним [15] — выше среднего. Гибкость языка средняя, поскольку страдает из-за его сильной нацеленности на объектно-ориентированное программирование и излишней простоты. Но эта простота позволяет значительно уменьшить время сборки, что даёт скорость разработки выше среднего [15].

Scala является статически и строго типизированным языком, избегающим по возможности мутабельности [12] — высокая надежность. Быстродействие выше среднего и сопоставимо с быстродействием Java, поскольку оба языка выполняются на JVM [15]. Язык обладает гибким синтаксисом и мощной системой типов, что делает его очень гибким [12]. Но это влечёт за собой увеличение времени сборки, что делает скорость разработки средней [15].

Вышеописанные оценки характеристик языков были приведены к пятибальной шкале и сведены в таблицу 6.

Таблица 6 – Сравнение языков программирования

Характеристика	C++	Java	Scala
Надёжность	3	4	5
Быстродействие	5	4	4
Гибкость	3	3	5
Скорость разработки	4	4	3
Итого	15	15	17

Исходя из результатов сравнения, в качестве языка для реализации был выбран язык программирования Scala. Также, в результате сравнения библиотек обработки изображений в предыдущем подразделе, были выделены OpenCV и JavaCV. Для Scala, выберем библиотеку JavaCV.

3.3 Разработка алгоритма обработки видеопотока в распределённой вычислительной системе

Во втором разделе, были выделены три подсистемы РВС для обработки видеопотока: балансировки нагрузки, обработки и упорядоченного слияния. На рисунке 8 представлена диаграмма активности распределённой вычислительной системы для обработки видеопотока, демонстрирующая взаимосвязь между этими подсистемами.

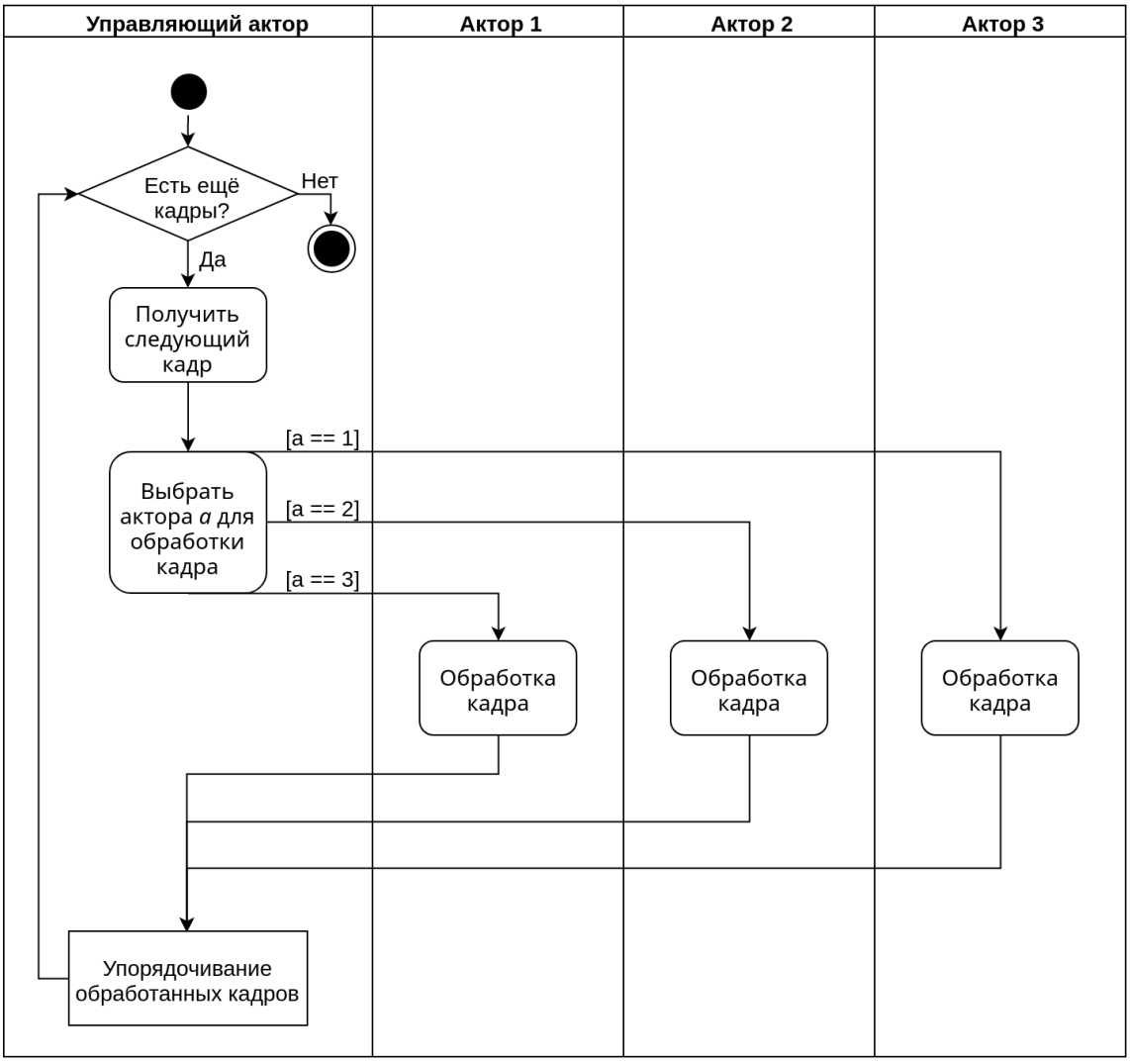

Рисунок 8 – Диаграмма активности распределённой вычислительной системы для обработки видеопотока (для трёх акторов-обработчиков)

Программная реализация подсистем рассматривается в следующих подразделах. Исходный код программной реализации находится в приложении А.

3.4 Разработка алгоритма балансировки нагрузки

Существуют следующие алгоритмы балансировки нагрузки [29]:

– круговое распределение (round robin);
– взвешенное распределение по кругу (weighted round robin);
– случайное распределение;
– распределение с учётом загруженности узлов.

Алгоритм кругового распределения — простой алгоритм балансировки. Первое поступившее задание балансировщик направляет первому узлу-обработчику, второе — второму, и т.д. Когда узлы кончаются, балансировщик начинает снова с первого узла. Таким образом, задания распределяются циклически и равномерно между всеми узлами [29].

Алгоритм распределения по кругу хорошо подходит для систем, состоящих из узлов с одинаковой производительностью. Однако, если в системе присутствуют узлы с различной производительностью, использование данного алгоритма может привести к перегрузке узлов с низкой относительной производительностью, или же к простою узлов с высокой [29].

Конечный автомат алгоритма распределения нагрузки по кругу для пяти узлов представлен на рисунке 9.

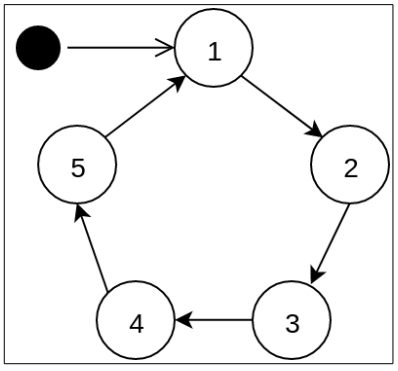

Рисунок 9 – Распределение нагрузки по кругу

Алгоритм взвешенного распределения по кругу призван решить вышеописанную проблему путём «взвешивания» узлов, то есть соотнесения каждого узла с определенным числом, называемым *весом* узла, которое

характеризует его относительную производительность. Веса учитываются при распределении заданий таким образом, чтобы количество заданий, распределенных узлу, было пропорционально весу узла [29].

Хотя данный алгоритм лучше подходит для случая, когда производительность узлов в системе различна [29], он требует задать веса заранее и не может подстраиваться под изменяющуюся во время выполнения конфигурацию системы. К тому же, заданные веса могут по той или иной причине не соответствовать (или соответствовать не в полной мере) действительной производительности узлов.

На рисунке 10 представлен конечный автомат взвешенного алгоритма распределения для четырёх узлов с весами, заданными в соотношении 2:4:1:3.

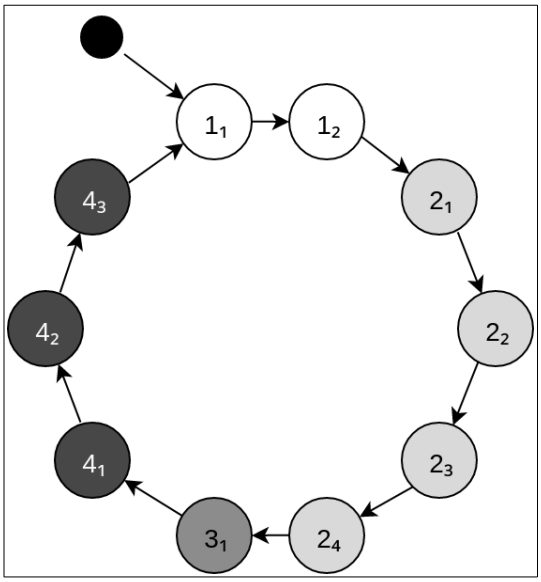

Рисунок 10 – Взвешенное распределение нагрузки по кругу. Различными цветами помечены группы состояний, соответствующие различным узлам

Алгоритм случайного распределения неплохо подходит при большом количестве узлов и заданий, при условии, что случайное распределение равномерно. Данный алгоритм обладает тем же минусом, что и распределение по кругу [29].

Алгоритмы с учётом загруженности узлов тем или иным образом получают информацию о загруженности узлов и применяют её для выбора

наиболее доступного [29]. В простейшем случае, каждый узел может посылать балансировщику сообщение об окончании выполнения задания, чтобы оповестить его о своей готовности к началу выполнению следующего.

Такие алгоритмы подходят для систем, в которых производительность отдельных узлов различна. Они также справляются с динамическим масштабированием вычислительной системы [29]. Однако, использование данного алгоритма повышает количество коммуникаций между узлами, что негативно сказывается на производительности.

Блок-схема алгоритма балансировки нагрузки с учётом загруженности узлов представлена на рисунке 11.

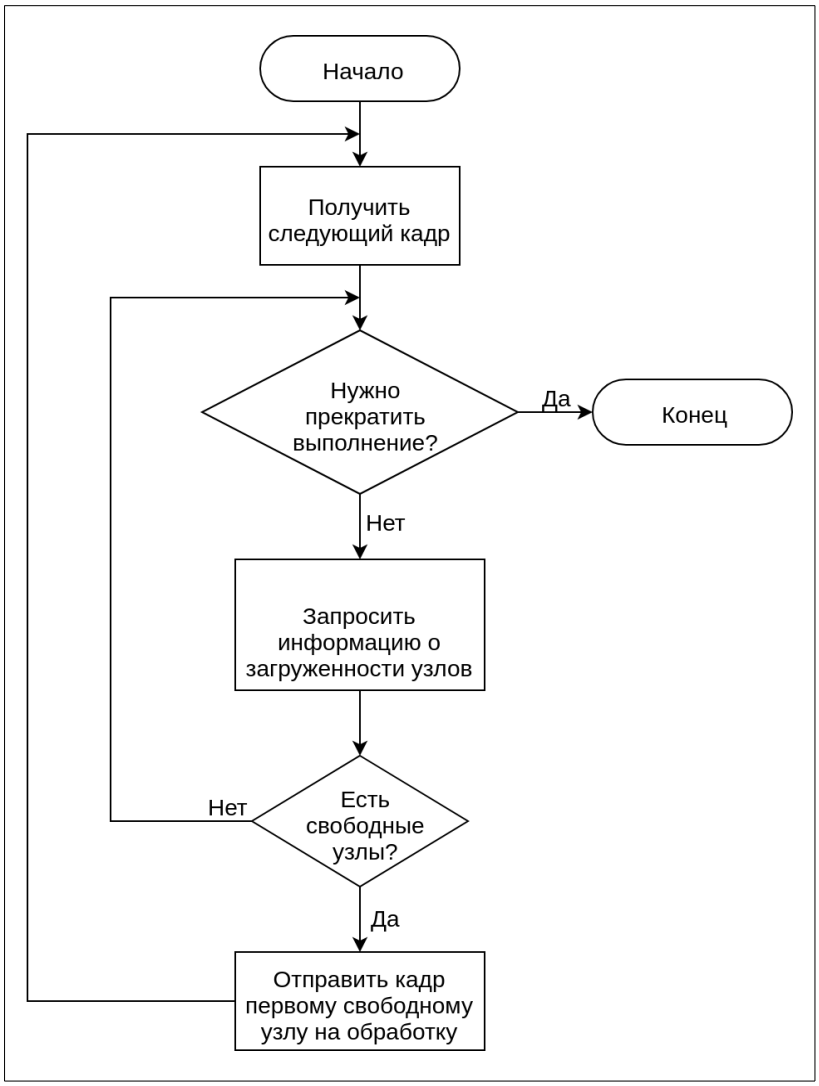

Рисунок 11 – Блок-схема алгоритма балансировки нагрузки с учётом загруженности узлов

В программной реализации использовались такие классы фреймворка Akka как RoundRobinPool и SmallestMailboxPool, реализующие соответственно круговой и динамический алгоритмы балансировки. Класс SmallestMailboxPool каждый раз выбирает актора с наименьшим количеством сообщений в очереди [16]. Исходный код программной реализации находится в приложении А.

3.5 Разработка алгоритма упорядоченного слияния

Алгоритм упорядоченного слияния восстанавливает порядок кадров после обработки. Каждому кадру заранее присваивается порядковый номер. Отсчёт начинается с нуля. При получении нового кадра, проверяется, является ли он следующим непосредственно за предыдущим, то есть больше ли его номер на единицу. Если это — следующий кадр, то он становится новым предыдущим, после чего аналогичным образом проверяются кадры во временном массиве, и перемещаются из него в поток упорядоченных кадров, если соответствуют условию. Если кадр не был следующим, то он помещается во временный массив.

Программная реализация использует для упорядочивания метод mapAsync класса Source библиотеки Akka Streams [16]. Этот метод обеспечивает параллельную обработку с упорядочиванием результатов. Обработка каждого кадра происходит во Future: метод запускает несколько Future и затем по порядку опрашивает их готовность. Как только результат обработки очередного кадра оказывается доступен, он помещается в поток обработанных кадров [16]. Исходный код программной реализации находится в приложении А.

Блок-схема алгоритма упорядоченного слияния представлена на рисунке 12.

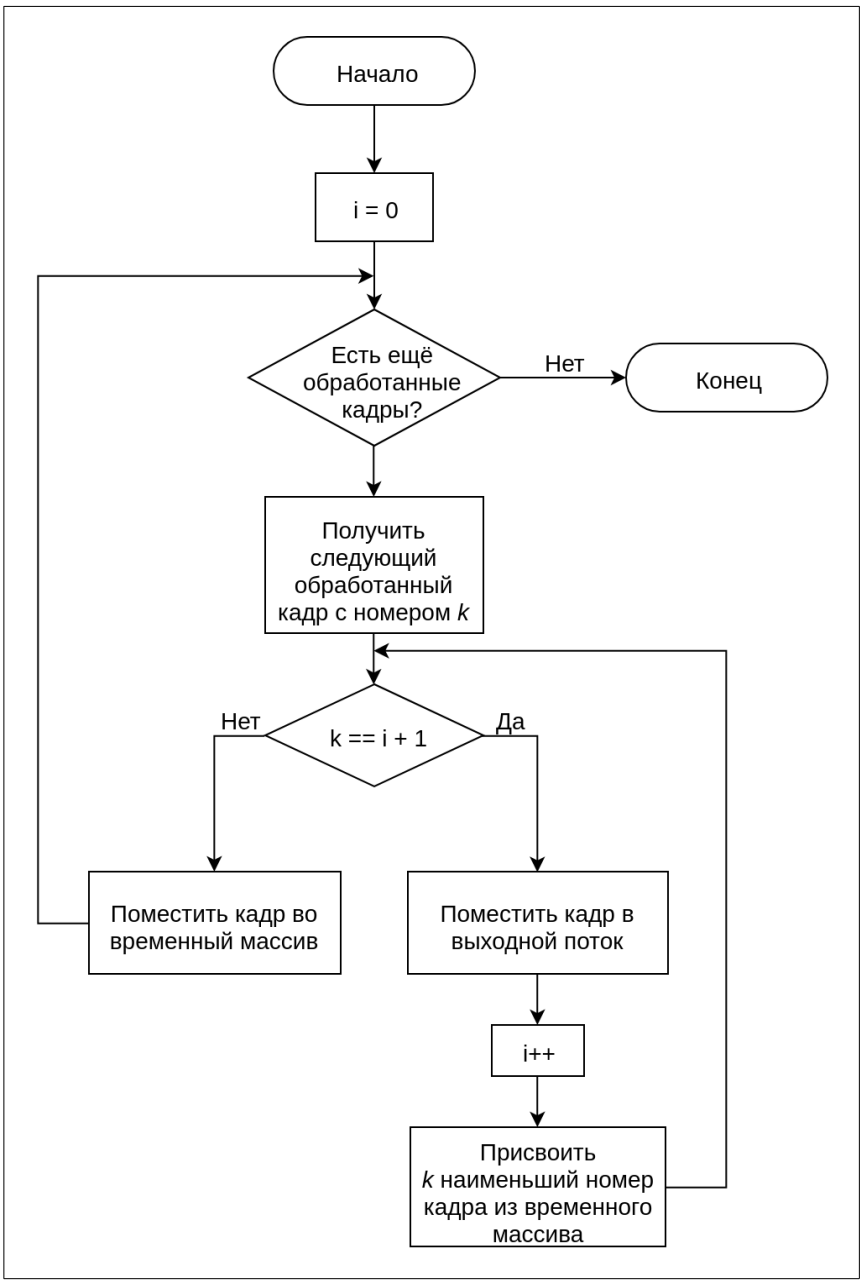

Рисунок 12 – Блок-схема алгоритма упорядоченного слияния

3.6 Анализ эффективности разработанной распределённой вычислительной системы обработки видеопотока

Замеры производились в распределённой системе из четырёх акторов-обработчиков, размещённых на отдельных компьютерах, оснащённых процессорами Intel Xeon E5-2630v3. Один из компьютеров также выполнял роль управляющего. На рисунке 13 представлена диаграмма развёртывания распределённой системы.

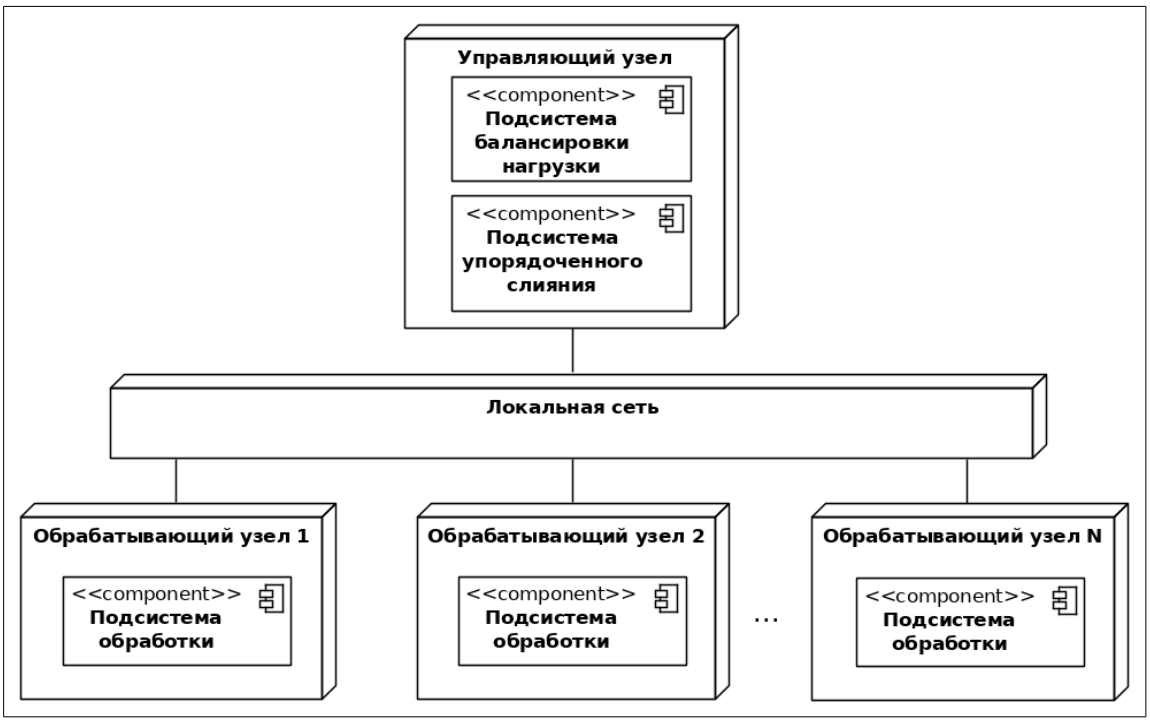

Рисунок 13 – Диаграмма развёртывания распределённой системы

Выполнялись два типа преобразований: распознавание лиц, что требует большого количества ресурсов, и простые преобразования — пороговый фильтр, инверсия цвета и поворот, что требует меньшего количества ресурсов. Результаты замеров производительности представлены на рисунке 14.

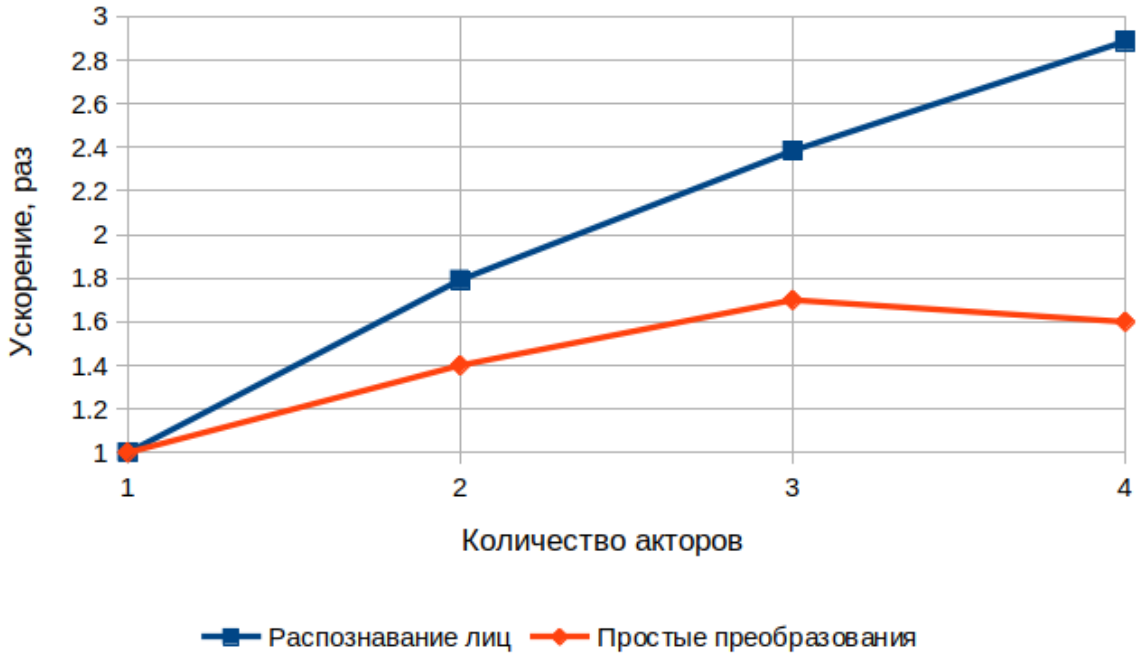

Рисунок 14 – Ускорение системы при различных видах преобразований

Как видно на графике, при более простых задачах производительность системы заметно хуже. А при четырёх узлах, производительность начинает падать. Это объясняется тем, что временные затраты на коммуникации являются слишком большими по сравнению со временем, затрачиваемым на вычисления.

Можно сделать вывод о том, что при достаточно больших задачах ускорение производительности системы соответствует закону Амдала, в то время как на мелких задачах ускорение производительности соответствует УЗМ.

Затем, были сравнены алгоритмы балансировки нагрузки: круговое и динамическое распределение. Результаты замеров представлены на рисунке 15.

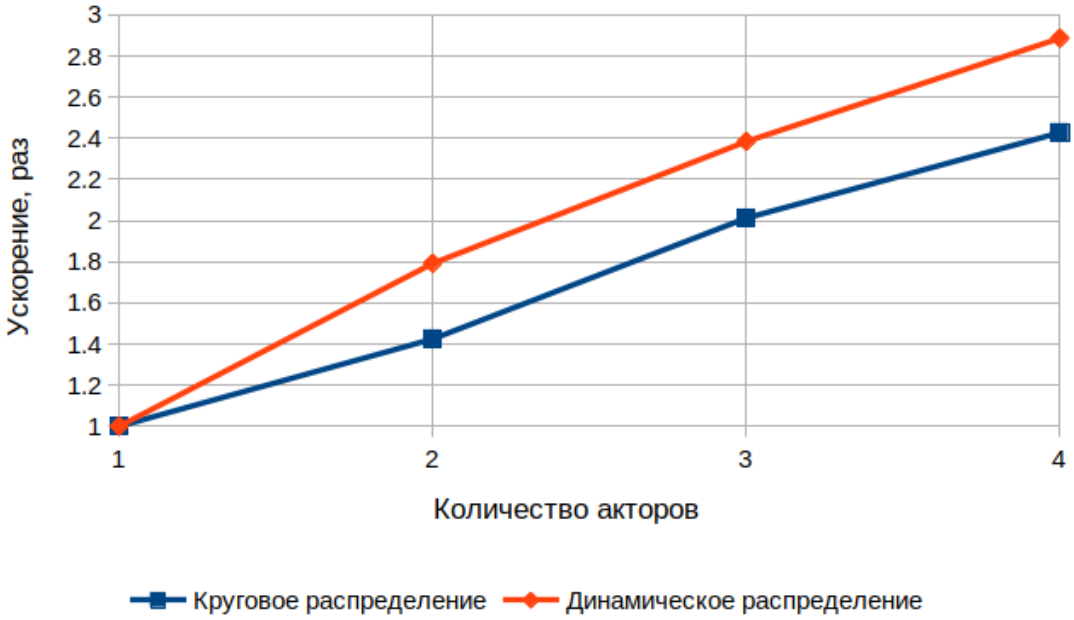

Рисунок 15 – Ускорение системы при использовании различных алгоритмов балансировки нагрузки

Как и ожидалось, динамическое распределение дало больший прирост производительности, нежели круговое. Это объясняется тем, что система была загружена более равномерно, а обработчики меньше простаивали.

Подводя итог, можно сказать, что производительность системы соответствует рассмотренным ранее законам ускорения производительности, а динамическая балансировка задач в самом деле оказалась эффективнее в системе с различной производительностью узлов.

3.7 Выводы и результаты по разделу 3

В первом подразделе были сравнены библиотеки для обработки изображений (видеопоток состоит из упорядоченных изображений). Были рассмотрены лицензии, состояние документации и производительность библиотек, в результате чего были выделены OpenCV и JavaCV.

Во втором подразделе были сравнены языки программирования C++, Java и Scala. Языки сравнивались по показателям надёжности, быстродействия, гибкости и скорости разработки. В результате сравнения был выделен язык Scala. Для этого языка, была выбрана библиотека JavaCV.

В третьем подразделе был разработан алгоритм обработки видеопотока в распределённой вычислительной системе. Была построена диаграмма активности системы.

В четвёртом подразделе были разработаны алгоритмы балансировки нагрузки: круговая, взвешенная круговая, случайная и с учётом загруженности узлов. Были построены конечные автоматы для первых двух алгоритмов. Для последнего была построена блок-схема. Был сделан вывод о том, что наиболее эффективной будет являться алгоритм с учётом загруженности узлов.

В пятом подразделе был разработан алгоритм упорядоченного слияния с применением временного буфер кадров. Была построена блок-схема алгоритма.

В шестом подразделе была проанализирована эффективность реализованной системы. Замеры производились в системе из четырёх акторов-обработчиков, каждый из которых находился на отдельном компьютере. Был сделан вывод о том, что система более эффективна при увеличении сложности преобразований. Также был сделан вывод о том, что алгоритм динамического распределения нагрузки в самом деле оказался эффективнее более простого алгоритма кругового распределения.

ЗАКЛЮЧЕНИЕ

В основной части работы были выполнены следующие задачи, необходимые для достижения поставленной цели:

1. Были рассмотрены и сравнены различные технологии для создания распределённых систем. Для построения системы была выбрана модель акторов.

2. Была спроектирована распределённая вычислительная система для обработки видеопотока, были выделены три подсистемы: балансировки нагрузки, обработки и упорядоченного слияния.

3. Спроектированная система была реализована на основе библиотеки Akka, которая реализует модель акторов, и библиотеки JavaCV (обёртка над OpenCV) для обработки изображений. Были рассмотрены алгоритмы балансировки нагрузки и реализован алгоритм упорядоченного слияния.

4. Был проведён анализ эффективности реализованной распределённой вычислительной системы. При более сложных преобразованиях, система показала больший прирост производительности, который соответствовал закону Амдала. При простых преобразованиях, ускорение производительности было меньше и соответствовало универсальному закону масштабируемости. Стоит также отметить, что в условиях различной производительности узлов, наиболее эффективным алгоритмом балансировки нагрузки оказался динамический.

Разработка в перспективе может применяться в системах управления беспилотными автомобилями или, с некоторыми доработками, в системах индексации баз данных изображений.

СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ

Нормативно-правовые акты

1. ГОСТ 2.105 – 95. Общие требования к текстовым документам [Текст]. – М.: Изд-во стандартов, 1996. – 29 с. – (Единая система конструкторской документации).

2. ГОСТ 7.1-2003. Библиографическая запись. Библиографическое описание документа.

3. ГОСТ 7.32-2001. Отчет о научно-исследовательской работе. Структура и правила оформления.

4. ГОСТ 7.82-2001. Библиографическая запись. Библиографическое описание электронных ресурсов.

5. ГОСТ 19.701 – 90. Схемы алгоритмов, программ, данных и систем. Условные обозначения и правила выполнения (ИСО 5807–85) [Текст]. Введен 1992–01–01. – М.: Изд-во стандартов, 1992. – 14 с. – (Единая система программной документации).

Научная и методическая литература

6. Гонсалес Р. Цифровая обработка изображений / Р. Гонсалес, Р. Вудс. – Москва : Техносфера, 2012. – 1104 с.

7. Алпатов А. Н. Цветков В. Я. Проблемы распределенных систем // Перспективы науки и образования. – 2014. – no. 6 (12).

8. Таненбаум Э. Распределённые системы. Принципы и парадигмы / Э. Таненбаум, М. Ван Стеен. – СПб.: Питер, 2003. – 877 с.

9. М.С. Косяков. Введение в распределенные вычисления. – СанктПетербург: НИУ ИТМО, 2014. – 155 с.

10. Высокопроизводительные вычисления для многоядерных многопроцессорных систем / Гергель В.П. – Издательство Нижегородского госуниверситета, 2011 – 421с.

11. Реализация сценария «следуй за мной» беспилотной системы управления автомобилем-роботом на основе данных лидара и видеодатчика /

Артёмкин В.В., Лукша С.С., Маликов А.Ю. // Вестник Рязанского государственного радиотехнического университета. – 2013. – №4-3(46).

12. Одерски, М. Scala. Профессиональное программирование / Одерски М., Спун Л., Веннерс Б. – 3-е изд. – СПб.: Изд-во Питер, 2018. – 688 с.

13. Джошуа Блох. Java. Эффективное программирование – Москва: Лори: Java «из первых рук», 2014. – 310 с.

14. Радченко Г.И. Распределенные вычислительные системы / Г.И. Радченко. – Челябинск : Фотохудожник, 2012. – 184 с.

Электронные ресурсы

15. Robert H. Loop Recognition in C++/Java/Go/Scala // Отчет результатов тестирования, опубликованный Google. [Электронный ресурс]. – Режим доступа: https://days2011.scala-lang.org/sites/days2011/files/ws3-1-Hundt.pdf (дата обращения: 20.05.2019).

16. Akka Documentation [Электронный ресурс]. – Режим доступа: https://doc.akka.io/. – (дата обращения: 20.05.2019).

17. Concurrency in Erlang & Scala: The Actor Model [Электронный ресурс]. – Режим доступа: https://rocketeer.be/articles/concurrency-in-erlang-scala/. – (дата обращения: 20.05.2019).

18. The Scala Programming Language [Электронный ресурс]. – Режим доступа: https://www.scala-lang.org. – (дата обращения: 20.05.2019).

19. OpenCV [Электронный ресурс]. – Режим доступа: https://opencv.org/. – (дата обращения: 20.05.2019).

20. JavaCV [Электронный ресурс]. – Режим доступа: https://github.com/bytedeco/javacv. – (дата обращения: 20.05.2019).

21. JavaCV 1.5 API [Электронный ресурс]. – Режим доступа: http://bytedeco.org/javacv/apidocs/. – (дата обращения: 20.05.2019).

22. AForge.NET [Электронный ресурс]. – Режим доступа: http://www.aforgenet.com/. – (дата обращения: 20.05.2019).

23. Трутнев Д.Р. Архитектуры информационных систем. Основы проектирования [Электронный ресурс]: Учебное пособие. – СПб.: НИУ ИТМО, 2012. – 66 с. – Режим доступа: http://window.edu.ru/resource/174/78174. – (дата обращения: 20.05.2019).

24. Parallel Programming Models [Электронный ресурс]: - Luis Moura e Silva and Rajkumar Buyya – Режим доступа: http://www.buyya.com/cluster/v2chap1.pdf. – (дата обращения: 20.05.2019).

25. Машинное зрение: понятия, задачи и области применения [Электронный ресурс]. – Электрон. дан. – Режим доступа: http://rusnauka.com/25_SSN_2009/Informatica/51050.doc.htm. – (дата обращения: 20.05.2019).

26. Системы компьютерного зрения: современные задачи и методы [Электронный ресурс]. – Электрон. дан. – Режим доступа: http://controleng.ru/innovatsii/sistemy-komp-yuternogo-zreniya-sovremenny-ezadachi-metody/. – (дата обращения: 20.05.2019).

Литература на иностранных языках

27. Gunther Neil J., Subramanyam Shanti, Parvu Stefan. A Methodology for Optimizing Multithreaded System Scalability on Multi-cores // CoRR. — 2011. — Vol. abs/1105.4301.

28. King, A. J. M. Communicating Sequential Processes in High Performance Computing. – 2015.

29. D. Kashyap, J. Viradiya, A Survey of Various Load Balancing Algorithms in Cloud Computing // International Journal of Scientific & Technology Research, Vol. 3, Iss. 11. – 2014.

30. G. M. Amdahl. Validity of the single processor approach to achieving large scale computing capabilities. // Proc. AFIPS Conference, Reston, VA, vol. 30, pp. 483-485, April 1967.

31. Stroustrup B. The C++ Programming Language / B. Stroustrup Addison-Wesley – 4th Edition – 2013 – 1346 с.

ПРИЛОЖЕНИЕ А

Исходный код

Исходный код разработанного приложения находится на компакт-диске.