САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

**КАФЕДРА ТЕХНОЛОГИИ ПРОГРАММИРОВАНИЯ**

**Широбоков Михаил Владимирович**

**Магистерская диссертация**

# Реализация высокопроизводительных растровых вычислений для Web ГИС

Направление 02.04.02

Фундаментальная информатика и информационные технологии

Научный руководитель,
канд. техн. наук,
доцент
Блеканов И. С.

Санкт-Петербург

2018

# Содержание

# Введение

Различные геоинформационные системы уже давно стали неотъемлемой частью современного информационного мира. Они используются в совершенно различных областях теоретических и, главным образом, прикладных наук, находя себе применение в таких дисциплинах, как урбанистика, экология, биология и экономика. Их история берет начало еще в 80-х годах прошлого века, когда были предприняты первые успешные попытки эколого- и биогеографического моделирования для исследования и выявления разнообразных закономерностей в распределении видов животных и его зависимости от различных природно-экологических факторов. Впоследствии подобные системы стали использоваться очень широко, и в настоящее время технологии растрового геоинформационного моделирования и анализа применяются в совершенно не связанных между собой областях, начиная от прогнозирования ареалов распространения биологических объектов и метеорологии и заканчивая анализом автомобильного трафика в мегаполисах.

Все сильнее ускоряющееся развитие цифрового мира неизбежно влечет за собой увеличение общего объема данных, подлежащих обработке. Справиться с их анализом силами одного компьютера или даже целого кластера часто бывает сложно, а иногда и невозможно по финансовым или техническим причинам. Это явление не обошло стороной и рынок географических информационных систем, нагрузка которых часто включает в себя работу не только с относительно легковесными векторными объектами, но и зачастую с космическими снимками поверхности, могущими иметь очень высокие разрешения. В настоящее время существуют космические аппараты, способные получать информацию об объектах, отстоящих друг от друга на расстоянии в 31 сантиметр, что дает примерно 10 пикселей на квадратный метр земной поверхности, и этот показатель постоянно увеличивается [1].

На данный момент существует достаточно много программных продуктов, так или иначе взаимодействующих с этим типом географических данных. Среди них есть как известные всем продукты массового использования, такие как Google Maps и Яндекс.Карты, так и менее известные во многом благодаря своей специфике сервисы, позволяющие применять подход геоинформационных систем, например, к проблемам урбанистики [2, 3, 4] и экономики [5]. Одним из таких проектов является O-GIS [6] — геоинформационная система с открытым исходным кодом, в рамках которой в данной работе разрабатывается вычислительный модуль.

Но, как и любой проект, находящийся в стадии разработки, O-GIS имеет нерешенные проблемы, одной из которых является невысокая скорость выполнения операций растровой алгебры, включающих в себя как различные прямые пользовательские запросы, так и проективные преобразования. В рамках данной работы предпринята попытка переработки вычислительного модуля системы, включающего в себя как общие методы реализации высокопроизводительных систем, так и специфичные, могущие найти свое применение непосредственно в рамках обозначенной системы.

# Постановка задачи

Основной целью данной работы является разработка архитектуры вычислительного модуля для Географической Информационной Системы O-GIS. Модуль представляет собой комплексную систему, состоящую из нескольких отдельных частей, каждая из которых позволяет решить ту или иную подзадачу. Набор таких подсистем определяется основными факторами, определяющими производительность выполнения операций над растрами в рамках ГИС, которыми являются:

- возможность горизонтального и вертикального масштабирования Географической Информационной Системы;
- распределение «тяжелых» данных по отдельным машинам внутри вычислительной сети;
- распределение нагрузки между отдельными вычислительными узлами в процессе работы;
- приоритизация задач с учетом специфики порядка получения результатов конечным пользователем;
- возможность параллелизации операций доступа к данным, их изменения и произведения вычислительных операций над ними в рамках одной машины.

Общую схему системы и порядок взаимодействия подсистем между собой можно увидеть в разделе «Общая архитектура вычислительного модуля». Более детально подзадачи, решаемые в рамках какой-либо части разрабатываемой системы, описаны в начале соответствующих разделов.

# Обзор литературы

Тематика высокопроизводительных вычислений в рамках геоинформационной системы является достаточно узкой; кроме того, большинство существующих географических информационных систем являются коммерческими продуктами и не предоставляют информацию о механизмах их внутренней работы. По этим причинам в открытом доступе практически нет литературы, целиком покрывающей предметную область. Тем не менее, разбив задачу на части, можно обнаружить множество связей с другими предметными областями, литература по которым является не только общедоступной, но и зачастую общей и фундаментальной в смысле характера содержащейся в ней информации.

Так, например, задача распределения нагрузки по вычислительной сети имеет глубокие корни в теории игр и еще в прошлом веке приобрела статус NP-полной [7]. Существуют полноценно описанные алгоритмы для какого-либо частного случая (например, в работе [8] предполагается, что время передачи данных, необходимых для выполнения задачи, существенно меньше времени их выполнения, а в [9] время коммуникации считается не зависящим от конкретных узлов), а также множество работ, посвященных утилизации различных эвристических предположений [10, 11, 12]. Идеология сведения задачи к оптимизационной, используемая в данной работе, была подчерпнута в [13].

В рамках работы используются инструменты теории вероятностей и математической статистики, как общие, так и специальные. Существует множество работ, описывающих основу примененных техник, однако автор данной работы рекомендует труд [14] как написанный понятным и доступным языком, но в тоже время логичный и последовательный источник сведений по предмету. Для получения более специализированных сведений, описывающих

доказательную базу пуассоновского процесса рекомендуется к ознакомлению работа [15].

Использование мощностей графического процессора для обработки данных является больше инженерной и утилитарной задачей, чем научной, но, тем не менее, в работе [16] содержатся полезные сведения о методологии внедрения вычислений с использованием CUDA, а [17] и [18] предлагают теоретический и практический анализ возможностей CUDA и OpenCL.

# 1. Общая архитектура вычислительного модуля

Следующие утверждения принимаются в качестве базовых в процессе проектирования архитектуры вычислительного модуля:

- система должна иметь минимальное количество единых точек отказа, в идеале — не иметь их вообще;
- помимо вычислительных мощностей выделенных серверов должны быть по возможности использованы ресурсы пользовательских устройств, но без создания неудобств для конечного пользователя;
- ресурсы системы должны быть использованы максимально полно за счет утилизации возможностей распределенной системы как системы параллельного доступа.

Таким образом, общий вид системы схематично можно представить диаграммой, изображенной на Рис. 1. Условные обозначения, принятые на схеме:

- T-CL: «тонкий» веб-клиент без возможности обработки растров;
- H-CL: «толстый» клиент, имеющий возможность обработки данных на месте;
- L-S: сервер, отвечающий за бизнес-логику приложения, первичную обработку запросов и распределение задач по вычислительным узлам;
- W-S: веб-сервер, необходимый для работы с веб-клиентами; не занимается распределением задачи, являясь, по сути, лишь необходимой прослойкой между веб-клиентом и хранилищем мета-данных;
- M-DB: база данных, хранящая мета-информацию о клиентах, их запросах, а также информацию о распределении растровых данных по шардам;

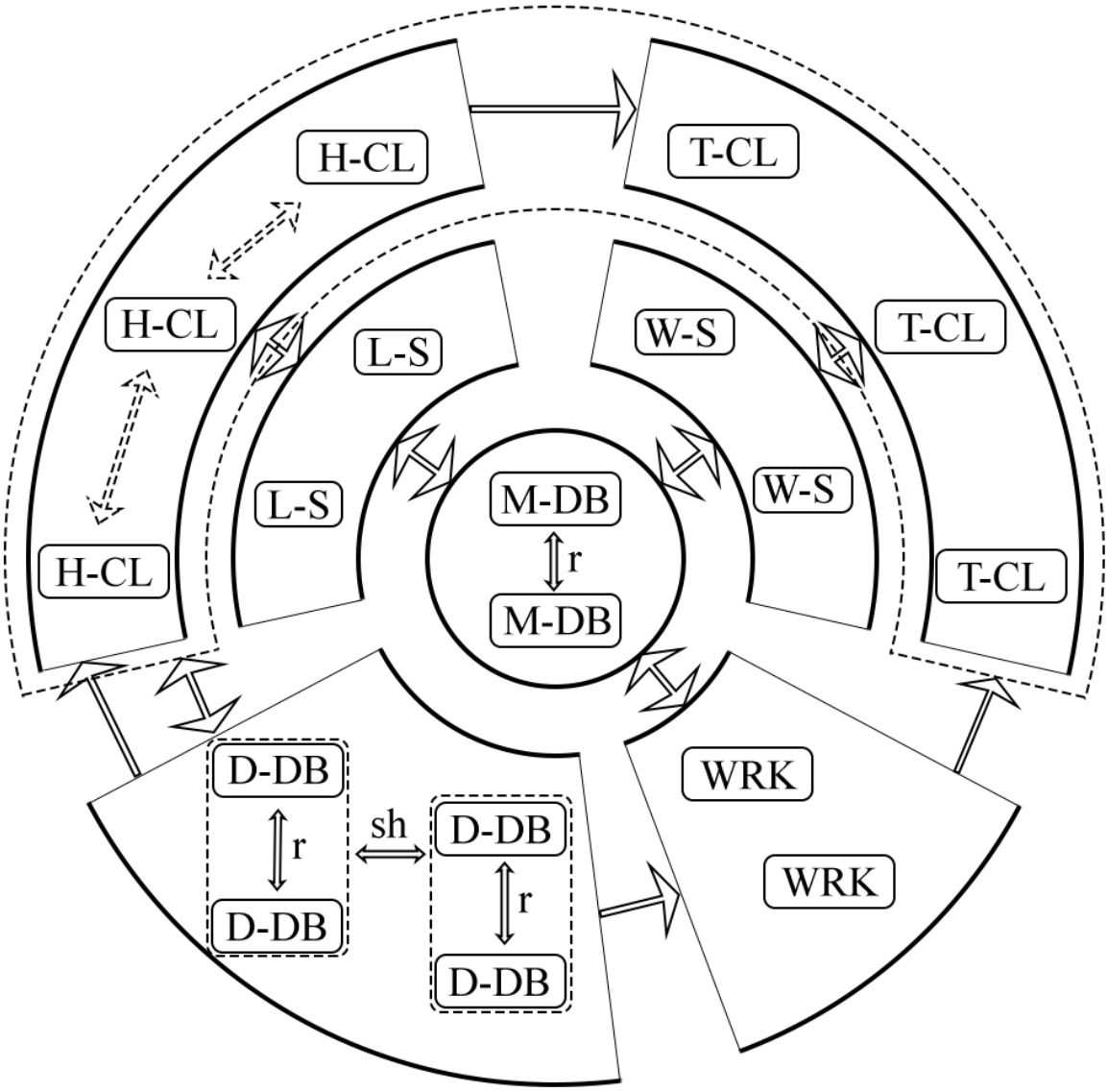


Рис. 1. Общая схема работы вычислительного модуля.

- D-DB: база данных, содержащая отдельный шард с частью растровых данных;
- WRK: выделенный вычислительный узел, полностью находящийся в распоряжении системы;
- r: метка «r» возле стрелки обозначает процесс репликации между узлами СУБД;
- sh: метка «sh» возле стрелки означает, что соединенные ей узлы хранят часть коллекции растров, т. е. шардированные данные

растровой «мозаики» (подробнее процесс распределения растровых данных по кластеру будет рассмотрен в разделе 3).

В разделе «Обзор существующих решений и пакетов» даны краткие описания и разобраны преимущества и недостатки существующих решений, допускающих свободное использование в проектах с открытым исходным кодом, которые могут взять на себя ответственность за реализацию стандартных методов поддержания целостности системы, а также помочь максимально эффективно использовать имеющиеся вычислительные ресурсы.

Принципиальной особенностью системы является использование мощностей пользовательских устройств с установленными «толстыми» клиентами. При этом на клиентском устройстве в первую очередь должны обрабатываться данные, необходимые непосредственно на данном узле, но, помимо этого, могут обрабатываться и данные, результат обработки которых требуется на других узлах. Подробнее процесс распределения задач по кластеру рассмотрен в разделе 4.

При выполнении вычислений не на стороне клиента они производятся в отложенном режиме, выгружая пользователю необходимые данные по мере их фактического требования, за исключением небольшого числа вычислений, выполняемых над еще не запрошенными данными, которые с высокой вероятностью будут запрошены пользователем на следующем шаге с целью минимизации времени ожидания отклика от системы. Подробнее данный процесс рассмотрен в разделе 5.

Пользовательские запросы, полученные одним из серверов, отвечающих за логику работы системы или веб-сервером, перед исполнением проходят дополнительную обработку на одном из серверов, отвечающих за распределение задач при помощи символьного анализа запроса, и по возможности упрощаются во избежание произведения ненужных вычислений.

Средства, позволяющие выполнить оптимизацию запроса, рассмотрены в разделе 6.

# 2. Обзор существующих решений и пакетов

В данной главе будет проведен обзор существующих продуктов и пакетов, упрощающих реализацию компонент системы, отмеченных в предыдущем разделе, либо помогающих в решении задач, возникающих в ходе разработки системы.

## 2.1. СУБД с возможностью хранения пространственных данных

Логическая организация данных осуществляется при помощи СУБД, имеющей возможность хранения географических или просто растровых данных (в случае использования неструктурированных карт-изображений). В то время как очевидным представляется решение проблемы их хранения путем самостоятельного описания структуры данных с использованием примитивов, предоставленных интерфейсом СУБД, такой подход может снизить производительность из-за дополнительных издержек на выполнение внешнего кода. В связи с этим, имеет смысл рассмотреть СУБД, поддерживающие хранение географических данных «из коробки» или имеющие расширения, позволяющие добавить требуемую функциональность. Несмотря на то что в рамках данной работы возможности СУБД по хранению и обработке пространственных данных не используются, данный критерий важен в контексте объединения вычислительного модуля с O-GIS, подобный функционал в которой необходим.

Другим критерием оценки рассматриваемых продуктов является возможность распределенного хранения данных. Несмотря на то, что при работе с любой СУБД можно организовать работу нескольких отдельно стоящих экземпляров как распределенной БД на уровне приложения, работающего с ними, при наличии возможности распределения данных силами самой СУБД этот вариант является предпочтительным.

### 2.1.1. MySQL

MySQL хранит географические данные в формате WKT (Well-known Text) или в бинарном эквиваленте, WKB (Well-known Binary). Имеются примитивы для хранения точек, линий и многоугольников, а также встроенные средства для нахождения пересечений и расстояний между объектами [19]. WKT — достаточно старый стандарт хранения географических данных, представляющий точки и геометрию в виде набора текстовых инструкций. Например, для хранения точки с широтой и долготой 12 и -75, соответственно, в WKT-файле будет содержаться строка «POINT (12 -75)» [20]. MySQL предоставляет множество функций для вычисления расстояния между точками, выбора точек внутри квадрата и т. д. Недостаток хранения данных в MySQL заключается в том, что все эти вычисления происходят в евклидовом пространстве, а значит — на плоскости. Поэтому для работы именно с географическими данными, а не просто с абстрактной геометрией, недостаточно имеющихся процедур обработки данных и возникает необходимость написания процедур на языке SQL или внешнего кода, что снова приводит к дополнительным издержкам.

«Из коробки» MySQL не имеет возможностей для работы в режиме распределенной БД, но существуют готовые решения, позволяющие добиться требуемой функциональности. Наиболее популярным решением на сегодняшний день является MySQL Cluster, имеющий свободно распространяемую Community-версию [21].

### 2.1.2. PostGIS

PostGIS — это расширение для СУБД PostgreSQL, добавляющее функции работы с географическими данными [22]. Оно работает во многом схоже с MySQL, когда хранит данные в формате WKT или WKB и имеет такие же функции для определения дистанций, пересечений и т. п. Основное отличие заключается в том, что PostGIS поддерживает специальные «географические»

координаты, что означает поддержку PostGIS работы с точками на глобусе, а не в Евклидовом пространстве.

Для PostgreSQL существует множество различных проектов, решающих задачу построения распределенной системы на ее базе. Наиболее интересными из всего множества являются Citus [23] (поглотивший pg_shard) и Postgre-XL [24], поскольку они позволяют производить шардирование данных при помощи внутренних инструментов.

### 2.1.3. MongoDB

MongoDB с 2010 года поддерживает индексацию географических данных и сохраняет местоположение в виде т. н. географического хэша [25], что позволяет хранить широту и долготу в виде одномерного значения. На практике геохэш хранит не просто точку, а некоторый участок пространства на глобусе, в котором находится объект. MongoDB позволяет хранить только точки, хранение многоугольников и линий недоступно. В то время как в общем случае это может не быть удовлетворительным решением, во многих частных ситуациях подобной функциональности вполне хватает. Из встроенных функций следует отметить возможность поиска точек вблизи некоторой заданной или внутри прямоугольника, работающую с достаточно высокой производительностью [26]. Минусом реализации является предположение о сферической форме Земли.

Лежащих на поверхности проблем с распределенностью MongoDB не имеет, поскольку эта СУБД изначально создавалась в данной парадигме [27].

### 2.1.4. SpatiaLite

SpatiaLite является свободно-распространяемым кроссплатформенным расширением SQLite для работы с географическими, в первую очередь векторными, данными. Эта база данных добавляет в SQLite возможности по работе с проекциями и позволяет выполнять более сложные операции над

векторными типами данных. Поддержка растровых типов данных предоставлена только в базовом виде [28]. Преимуществом SQLite является хорошая поддержка векторных типов данных. Недостатком же служим слабая поддержка растровых типов данных. Простая архитектура данной БД (модуль встраивается в само приложение, а данные хранятся как обычные файлы) может быть как преимуществом, так и недостатком, в зависимости от того, в каких условиях SpatiaLite будет использоваться. В случае с разрабатываемой в рамках данной работы системы, это является именно недостатком, поскольку для нее требуется полноценная и независимая от каких-либо контейнеров приложений база данных.

SQLite не является распределенной системой, и единственное поддерживаемое на данный момент решение rqlite [29] не позволяет производить шардирование данных между несколькими машинами, ограничиваясь репликацией. Таким образом, возможность производить шардинг с использованием данной СУБД остается на совести разработчика приложения-посредника.

### 2.1.5. CouchDB

CouchDB является свободно распространяемой NoSQL объектно-ориентированной базой данных на основе BerkeleyDB. Для хранения данных в CouchDB используются контейнеры JSON, а запросы в формате функций MapReduce могут быть написаны на Java, JavaScript или других подобных языках. Встроенной поддержки географических типов данных не имеет, но необходимый функционал может быть реализован через функции внутри базы данных или же как часть функции Reduce запроса [30, 31]. Широкий спектр возможностей, который может быть запрограммирован, также является преимуществом CouchDB [30]. К недостаткам стоит отнести достаточно сложные запросы и высокие требования к уровню подготовки администратора

БД. В рамках решаемой задачи отсутствие нативной поддержки растровых и векторных типов данных тоже можно считать недостатком CouchDB.

CouchDB «из коробки» поддерживает возможность репликации (в том числе и master-master), но не поддерживает возможность шардирования данных. Для использования этой функциональности существует расширение CouchDB Lounge [32].

### 2.1.6. Выводы

Из рассмотренных СУБД наиболее подходящими по критериям наличия средств для работы с пространственными и растровыми географическими и возможности построения распределенной системы являются PostgreSQL (с расширениями) и MongoDB. Для построения прототипа системы была выбрана MongoDB, поскольку на данный момент использование Citus не позволяет обрабатывать специфичные для PostGIS агрегатные функции, а Postgre-XL конфликтует с последней версией PostGIS на этапе установки.

## 2.2. Пакеты компьютерной алгебры

При решении задач оптимизации в процессе построения системы представляется необходимым использование систем компьютерной алгебры (СКА), берущих на себя ответственность за численную или аналитическую обработку данных. Одной из особенностей разрабатываемой системы является использование нативных C++ приложений на вычислительных хостах, поэтому возможность работы с C++ кодом является одним из критериев, по которым оцениваются различные СКА.

Поскольку одной из идеологий разработки O-GIS является использование только свободно распространяемого программного обеспечения, существующие проприетарные решения, подобные Wolfram Mathematica, Maple или MATLAB, не включены в данный раздел. Все

обозреваемые программные комплексы являются проектами с открытым исходным кодом.

### 2.2.1. Maxima

Maxima [33] — обособившаяся в 1982 году ветка развития проекта Macsyma, имеющего своим началом 60-ые годы прошлого века и разрабатывающегося в Массачусетском технологическом институте. Проект развивается и сейчас, на момент написания текста последняя доступная версия продукта выпущена в октябре 2017 года. Имеет достаточно широкий круг применения, включая возможности упрощения символьных выражений. Основная часть кода написана на LISP, и крупным недостатком системы в контексте настоящей работы является отсутствие возможности вызова функций Maxima из C++ кода.

### 2.2.2. Scilab

Scilab [34] — свободно распространяемый пакет компьютерной алгебры с открытым исходным кодом, разработка которого была начата в 1990 году под именем Ψlab (Psilab). На данный момент он представляет собой один из немногих аналогов проприетарной системы компьютерной алгебры Matlab. Функциональность пакета, хоть и существенно меньшая в таких областях применения, как, например, управление, позволяет проводить все стандартные алгебраические операции. В настоящее время существует отдельный инструментарий для работы с символьными вычислениями, основанный на Maxima, а также существует API доступа к ресурсам системы из нативного C++ кода.

### 2.2.3. JACAL

JACAL [35] — интерактивная система компьютерной алгебры, разрабатываемая в качестве UNIX-пакета. JACAL написан на алгебраическом языке SCHEMA, разработанным той же командой разработчиков. Его

основная цель — упрощение алгебраических уравнений и манипуляции с ними, а также работа с векторами и матрицами. Несмотря на то, что его функциональности достаточно для нужд разрабатываемого в рамках данной работы модуля, отсутствие какого-либо внешнего интерфейса не позволяет использовать его в качестве средства оптимизации запросов. Проект не разрабатывается с января 2015 года.

### 2.2.4. GiNaC

GiNaC (GiNaC is not a CAS) [36] — библиотека, написанная на C++, специально разрабатываемая для расширения возможностей языка в области прикладных и теоретических алгебраических расчетов, интегрированных в нативные приложения. Несмотря на то, что возможности GiNaC достаточно широки в контексте численного и аналитического решения алгебраических задач, упрощение символьных выражений изначально не входило в планы разработчиков системы [37], поскольку, по их мнению, «простота» выражения не имеет строгого определения и, таким образом, не подлежит имплементации.

### 2.2.5. YACAS

YACAS (Yet Another Computer Algebra System) [38] — система компьютерной алгебры, имеющая написанное на C++ ядро и представляющая значительные возможности по работе с символьными выражениями, в том числе и возможности по их упрощению. Отдельно распространяется как библиотека для Java и может быть подключен в качестве статической библиотеки в C++ приложение. Имеет встроенную поддержку упрощения символьных выражений.

### 2.2.6. Octave

Octave [39] — изначально разрабатываемая в качестве свободно распространяемого аналога Matlab система компьютерной алгебры, имеющая

возможности вызова своего интерпретатора из C++ кода. Предназначена для численных решений ряда линейных и нелинейных задач и имеет сопоставимые с Matlab возможности визуализации данных. К сожалению, ядро Octave не позволяет выполнять символьные операции, и инструментарий для работы с ними существует в качестве одного из пакетов Octave Forge, что усложняет линковку.

### 2.2.7. Выводы

Однозначно выбрать систему компьютерной алгебры на данном этапе не удастся, поскольку для различных задач могут потребоваться различные возможности. В последующих разделах данные продукты будут рассматриваться в контексте более специфичных требований.

## 2.3. Параллельная обработка растровых данных внутри одной машины

Нативная сущность географических данных, представленных в виде карты, предоставляет широкие возможности для ускорения операций растровой реклассификации и растровой алгебры путем параллелизации и использования SIMD-инструкций. Кроме того, существующие графические процессоры позволяют, при соответствующих модификациях алгоритмов, существенно ускорить подобные операции [40, 41]. В настоящее время наиболее широко применяемыми технологиями параллельной обработки данных являются следующие:

### 2.3.1. Advanced Vector Extensions

Advanced Vector Extensions (AVX, AVX2) — это расширение к набору инструкций для микропроцессоров от Intel и AMD, предложенное в 2008 году компанией Intel и предлагающее новый набор инструкций, функций и схем программирования [42]. Начиная с микроархитектур Sandy Bridge и Bulldozer,

соответственно, процессоры от Intel и AMD имеют набор векторных регистров с шириной в 128 или 256 бит, позволяющих загрузить множество данных для однотипной обработки.

### 2.3.2. Nvidia CUDA

CUDA (Compute Unified Device Architecture) — программно-аппаратная архитектура параллельных вычислений, которая позволяет существенно увеличить вычислительную производительность благодаря использованию графических процессоров фирмы Nvidia [43]. CUDA SDK позволяет программистам реализовывать на специальном упрощённом диалекте языка программирования C алгоритмы, выполнимые на графических процессорах Nvidia, и включать специальные функции в текст программы на C. Архитектура CUDA даёт разработчику возможность по своему усмотрению организовывать доступ к набору инструкций графического ускорителя и управлять его памятью.

### 2.3.3. OpenCL

Open Computing Language — фреймворк для написания компьютерных программ, связанных с параллельными вычислениями на различных графических и центральных процессорах, а также FPGA [44]. В OpenCL входят язык программирования, который основан на стандарте языка программирования C-99, и интерфейс программирования приложений. OpenCL обеспечивает параллелизм на уровне инструкций и на уровне данных и является осуществлением техники GPGPU. OpenCL является полностью открытым стандартом, его использование не облагается лицензионными отчислениями. Цель OpenCL состоит в том, чтобы дополнить открытые отраслевые стандарты для трёхмерной компьютерной графики и звука OpenGL и OpenAL возможностями GPU для высокопроизводительных вычислений. OpenCL разрабатывается и поддерживается некоммерческим консорциумом Khronos Group, в который входят много крупных компаний,

включая AMD, Apple, ARM, Intel, Nvidia, Sony Computer Entertainment и другие.

### 2.3.4. Выводы

Несмотря на то, что по своей структуре наиболее подходящим способом выполнения растровых операций является их обработка на графическом процессоре, необходимо провести тестирование различных подходов к вычислениям в рамках поставленной задачи.

# 3. Распределение данных по кластеру

Использование растровых географических данных влечет за собой возникновение таких проблем, как плохая масштабируемость и большие объемы используемой памяти, в первую очередь, оперативной. Для решения или обхода этих проблем используются техники хранения в виде пирамиды и мозаики. Использование пирамиды уменьшает объем производимых вычислений за счет избыточности данных: при меньшем приближении используются заранее подготовленные растры меньшего разрешения. Всего в пирамиде может быть не более $\log_2(\min(rows, columns))$ уровней [45]. Использование пирамиды увеличивает объем хранимых на диске данных не более, чем в два раза.

Мозаика используется для уменьшения нагрузки на оперативную память при работе с большими растрами путем разбиения файла, содержащего в себе все данные одного слоя или уровня пирамиды, на набор файлов меньшего размера [46]. Вместе эти две техники позволяют минимизировать использование оперативной памяти и уменьшить нагрузку на процессор при получении данных из указанной области слоя.

Для максимизации степени параллельности доступа к данным необходимо предоставить клиентской стороне возможность одновременной загрузки всех необходимых данных с максимально большого числа серверов. Предположим, что изображение, для которого необходимо произвести разложение на мозаику и пирамиду, имеет размерность $M$ на $N$ точек. При решении задачи оптимального разбиения изображения и представлении его в виде совокупности пирамиды и мозаики необходимо учитывать следующие требования:

- Минимальный размер изображения, хранимого отдельно, не должен быть менее $M_{min}$ по большей стороне. Данное требование вытекает из

практического применения растра небольшого размера: нет смысла в растре размером меньше, чем экран пользовательского устройства. Оно не позволит существенно сэкономить размер занимаемого пирамидой дискового пространства, поскольку он определяется частичной суммой геометрического ряда, «хвост» которого и составляют отброшенные небольшие растры.

- Для расчетов на вычислительных узлах удобно предоставлять тайлы одинакового размера.

При использовании кластера для обработки растра, разбитого на «пирамиду» и «мозаику», наиболее эффективное использование вычислительных ресурсов будет достигаться в том случае, когда все одновременно используемые элементы мозаики располагаются на различных узлах кластера, поскольку такая архитектура позволит производить загрузку и выгрузку информации с максимально доступной степенью параллельности.

Таким образом, алгоритм обработки нового растра для хранения его внутри СУБД можно представить следующим образом:

- Производится построение «пирамиды» из исходного растра. Каждый последующий слой содержит новый растр, уменьшенный в $\frac{1}{q}$ раз по вертикали и горизонтали. Процесс останавливается, как только на $k$-ой итерации $\min(M, N)\ q^{k+1}$ становится меньше $M_{min}$.
- Каждый слой полученной «пирамиды» разбивается на участки размера $M_{tile} \times N_{tile}$. В случае, когда размеры какого-либо слоя «пирамиды» $M_k$ и $N_k$ не делятся нацело на $M_{tile}$ или $N_{tile}$ соответственно, остаток растра разбивается на участки, не превышающие по обеим размерностям $M_{tile}$ и $N_{tile}$.
- Получившейся на каждом слое пирамиды «мозаике» ставится в соответствие вектор индексов элементов по правилу: на $i$-ой позиции

в векторе стоит значение $\left\lceil \frac{M_k}{M_{tile}} \right\rceil i + j$, где $i$ и $j$ соответствуют индексу элемента в матрице тайлов в «мозаике». Операцией $\lceil \cdot \rceil$ обозначено взятие ближайшего целого, большего либо равного аргументу. Ее результат соответствует количеству элементов в строках матрицы.

- Векторы объединяются в один, после чего для каждого тайла определяется индекс физического хранилища, на котором он должен быть сохранен, как остаток от деления его индекса в этом векторе на количество доступных серверов $K$. Объединение векторов позволяет избежать неравномерности загрузки данных на сервера, которая может возникнуть при определении индекса хранилища по индексу элемента внутри вектора, соответствующего одному слою из-за небольшого количества тайлов на верхних уровнях «пирамиды». Итоговый индекс тайла, находящегося на позиции $(i, j)$ в матрице, соответствующей мозаике на $k$-ом уровне пирамиды, определяется по правилу:

$$(i,j)_k \to \sum_{k=0}^{m} \left\lceil \frac{M_k}{M_{tile}} \right\rceil \cdot \left\lceil \frac{N_k}{N_{tile}} \right\rceil + \left\lceil \frac{M_k}{M_{tile}} \right\rceil i + j.$$

Доля избыточных данных при таком хранении составляет

$$\frac{q(1-q^k)}{1-q},$$

где $k$ определяется из условия

$$k = \underset{i}{\operatorname{argmin}}\left(\min(M,N)\, q^{i+1} < M_{min}\right)$$

и равняется

$$\left\lfloor \frac{\log M_{min} - \log \min(M,N)}{\log q} \right\rfloor - 1,$$

где оператор $\lfloor \cdot \rfloor$ возвращает ближайшее целое, меньшее либо равное аргументу. Значение $q$ определяется эмпирически и может быть принято равным $\frac{1}{2}$ исходя из практики работы таких картографических сервисов, как Яндекс.Карты и Google Maps.

# 4. Оптимальное распределение текущей нагрузки между хостами в вычислительной сети

В последние десятилетия все большее распространение получают технологии распределенных и облачных вычислений, призванные облегчить и ускорить выполнение ресурсоемких задач, не привлекая к их выполнению суперкомпьютеры и специальное оборудование [47, 48]. Одним из ключевых отличий таких систем от классического кластера часто является принципиальная неоднородность устройств, вовлеченных в процесс: например, для вычислений на движке браузера могут быть использованы практически произвольные устройства, вплоть до смартфонов и иных видов КПК [49]. При этом обычно задача распределения нагрузки в таких системах отдельно не решается, что может приводить как к чрезмерной загрузке отдельных устройств, что в случае добровольного согласия на предоставление вычислительных мощностей может обернуться рядом неудобств для пользователей системы, так и к появлению «бутылочного горлышка» в точке распределения. В рамках данного раздела рассмотрим способ оценки вычислительной мощности устройств в распределенной сети с учетом ее возможного изменения во времени.

## 4.1. Постановка задачи

В рамках данного раздела решается задача описания статистической модели вычислительной сети, необходимой для определения фактической вычислительной мощности устройств, входящих в ее состав. При построении модели принимаются в качестве базовых следующие утверждения:

- устройства в сети производят вычисления независимо друг от друга;
- внутри одного устройства время последующего вычисления не зависит от времени предыдущего;

- время работы, подлежащее измерению, значительно превосходит доступный квант времени;
- вычислительная сеть имеет большую нагрузку и всегда имеет очередь задач, ожидающих выполнения.

Для оценки вычислительной способности устройства производится оценка времени выполнения им отдельного практически значимого задания. При построении модели будем предполагать эквивалентность таких операций, в противном случае придется каким-либо образом производить нормировку полученных сведений — например, измерять время выполнения какого-либо числа заданных элементарных операций или работы над заданным объемом информации.

Целью раздела является моделирование системы, способной в автоматическом режиме оценивать вычислительную мощность доступных устройств, входящих в состав распределенной сети, изменяющуюся с течением времени.

## 4.2. Построение модели

Разобьем процесс построения модели на две части: изначально будем полагать параметры модели постоянными, а затем изменим получившуюся модель таким образом, чтобы она учитывала их возможное изменение во времени.

### 4.2.1. Статическая модель

Независимость устройств друг от друга позволяет моделировать их по отдельности. Предположим на данном этапе, что выбранное устройство статично в том смысле, что не меняет своих параметров с течением времени. В качестве объекта наблюдения будем использовать момент окончания выполнения отдельной задачи. Тогда из приведенных выше утверждений следует независимость количества таких моментов на произвольно взятом

отрезке временной шкалы от других, а в сочетании с очевидными свойствами процесса, такими как неотрицательное число измерений, следует применимость и адекватность модели пуассоновского процесса [15], в котором на произвольно взятом отрезке времени $[T_1, T_2]$ будет ожидаться $\lambda\,(T_2 - T_1)$ законченных вычислений, где $\lambda$ — это частота выполнения операций, полагаемая постоянной.

Для оценки этого параметра перейдем от непосредственно распределения числа вычислений к распределению времени между двумя последовательными их окончаниями. Для пуассоновского процесса это распределение является экспоненциальным, при этом параметр этого распределения $\lambda$ совпадает с параметром «родительского» распределения Пуассона [15].

Пусть $(t_1, \dots, t_n)$ — промежутки времени, в течение которых выполнялись $n$ операций. Воспользуемся байесовским подходом [14] для определения распределения этого параметра:

$$p(\lambda|t_1, \dots, t_n) = \frac{p(t_1, \dots, t_n|\lambda)p(\lambda)}{p(\,t_1, \dots, t_n)} =$$
$$= \frac{\lambda^n \exp(-\lambda \sum_{i=1}^{n} t_i)\, p(\lambda)}{\int_0^\infty \lambda'^n \exp\left(-\lambda' \sum_{i=1}^{n} t_i\right) p(\lambda') d\lambda'}.$$

В качестве априорного распределения параметра $\lambda$ имеет смысл выбрать гамма-распределение [38]: про него изначально известна лишь неотрицательность принимаемых значений, и, кроме того, при таком выборе апостериорное значение параметра тоже будет иметь вид гамма-распределения:

$$\frac{\lambda^n \exp(-\lambda \sum_{i=1}^{n} t_i)\beta^\alpha \lambda^{\alpha-1} e^{-\beta\lambda}\Gamma(\alpha)}{\Gamma(\alpha)\int_0^\infty \lambda'^n \exp\left(-\lambda' \sum_{i=1}^{n} t_i\right)\beta^\alpha \lambda'^{\alpha-1} e^{-\beta\lambda'} d\lambda'} =$$

$$= \frac{\lambda^{\alpha+n-1}\exp(-(\beta+\sum_{i=1}^{n} t_i)\lambda)}{\int_0^{\infty} \lambda'^{\alpha+n-1}\exp\left(-\left(\beta+\sum_{i=1}^{n} t_i\right)\lambda'\right) d\lambda'} =$$

$$= \frac{(\beta+\sum_{i=1}^{n} t_i)^{\alpha+n}\lambda^{\alpha+n-1}\exp(-(\beta+\sum_{i=1}^{n} t_i)\lambda)}{\int_0^{\infty} \left(\left(\beta+\sum_{i=1}^{n} t_i\right)\lambda'\right)^{\alpha+n-1} \exp\left(-\left(\beta+\sum_{i=1}^{n} t_i\right)\lambda'\right) d\left(\left(\beta+\sum_{i=1}^{n} t_i\right)\lambda'\right)} =$$

$$= \frac{(\beta+\sum_{i=1}^{n} t_i)^{\alpha+n}\lambda^{\alpha+n-1}\exp(-(\beta+\sum_{i=1}^{n} t_i)\lambda)}{\Gamma(\alpha+n)} \sim$$

$$\sim \Gamma\left(\lambda; \alpha+n, \beta+\sum_{i=1}^{n} t_i\right).$$

Это свойство позволяет легко интерпретировать априорные значения параметров $\alpha$ и $\beta$. Поскольку при наличии серии измерений времени выполнения операций $(t_1, \dots, t_n)$ параметр $\alpha$ дополняет их количество, а $\beta$ — сумму, их значения стоит выбирать так, чтобы они отражали ожидаемое число выполненных операций $\alpha$ за промежуток времени длиной $\beta$. Таким образом, выбрав начальные значения $\alpha$ и $\beta$ из имеющихся априорных сведений об устройстве, мы можем получить начальное значение $\lambda$ как реализацию случайной величины, подчиняющейся гамма-распределению с заданными параметрами, после каждой новой выполненной операции обновлять сведения о распределении $\lambda$ и получать его новое значение как реализацию новой случайной величины.

### 4.2.2. Добавление динамики

Для учета динамики параметров будем вносить «поправки» в модель в случае, если реальное время обработки не согласуется с ней. Поскольку реальное распределение времени обработки запроса нам неизвестно, в качестве подхода к определению «несогласованности» можно использовать относительное отклонение от текущего среднего значения. «Исправление» модели заключается в «забывании» старых значений, при этом чем хуже

модель согласуется с реальностью, тем больше старых значений будет «забыто». При этом параметры $\hat{\alpha} = \alpha + n$ и $\beta = \beta + \sum_{i=1}^{n} t_i$ изменяются соответствующим образом, и значение $\hat{\alpha}$ имеет смысл «уверенности» в модели, поскольку его увеличение уменьшает дисперсию распределения оценки исходного параметра. Воспользуемся следующей эвристикой: при относительной погрешности $\varepsilon$ более заданного заранее $\varepsilon_0$ будем оставлять $1 - \frac{\varepsilon}{2}$ последних полученных значений, при этом под $\varepsilon$ будем понимать ошибку относительно максимального из текущего среднего и полученного значения, что даст модели некоторую устойчивость к выбросам.

На рис. 2 слева показана динамика параметров при изменении среднего времени выполнения операций на устройстве с 200 до 600 условных единиц, происходящем на 50-ом отсчете, а справа — вид кривой распределения ожидаемого времени выполнения операции по окончании моделирования.

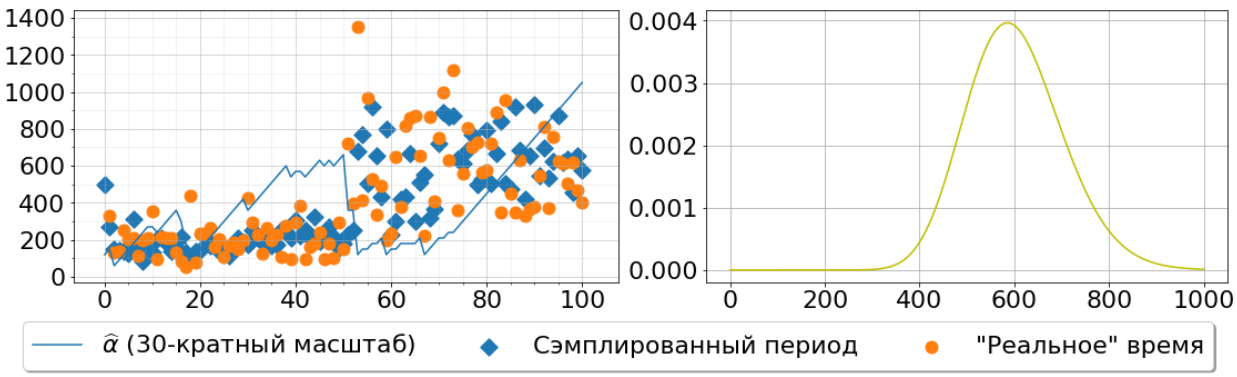


Рис. 2. Результаты имитационного моделирования.

### 4.2.3. Выводы

Предложенный метод оценки вычислительной мощности устройств, входящих в состав распределенной вычислительной сети, может найти наилучшее применение в добровольных и гибридных вычислительных сетях. В то время как в первых он позволяет динамически отслеживать мощность доступных ресурсов и управлять загруженностью устройств, в сетях второго типа он может понадобиться при оценке реальных вычислительных

возможностей сети и в разгрузке основных серверов с переносом части нагрузки на клиентскую сторону и вторичные устройства, находящиеся в сети.

# 5. Реализация отложенных вычислений в контексте Геоинформационной системы

Отложенные вычисления, зачастую в комбинации с мемоизацией, в общем случае применяются для уменьшения времени работы произвольной вычислительной системы за счет отсутствия необходимости производить вычисления в точке объявления результата и его кэширования после успешного выполнения операции в том случае, когда высока возможность его повторного использования. Существует множество языков программирования, поддерживающих ленивые вычисления на уровне ядра, но, поскольку специфика поставленной задачи подразумевает распределенные вычисления, в общем случае они не принесут существенной пользы.

В контексте распределенных растровых вычислений отложенные вычисления в первую очередь связаны с частым отсутствием необходимости производить заданные пользователем системы операции над всем требуемым объемом данных, поскольку их результаты не всегда требуются для отображения на устройстве аналитика, использующего сервис. Используемая в предыдущем разделе модель гетерогенной сети позволяет разделить задачу отложенных вычислений как минимум на две части:

- реализация отложенных вычислений на серверной стороне системы;
- реализация отложенных вычислений при использовании клиентской стороны в качестве вычислительной.

При этом необходимо учитывать разнородность клиентских устройств, поскольку даже при наличии возможности производить часть вычислений на стороне мобильного устройства их выполнение может быть связано с рядом трудностей, возникающих из-за сторонних причин, таких как нестабильное сетевое соединение или недостаточная пропускная способность сетевого

канала. В этом случае, несмотря на большую вычислительную мощность, получающуюся при добавлении в систему нового узла в виде клиентской машины, необходимо структурировать вычисления таким образом, чтобы сбалансировать время, требующееся на передачу неструктурированных и структурированных данных между клиентским и серверным устройствами, и время, необходимое для непосредственного выполнения запрашиваемых операций, опять же, на серверной и клиентской стороне.

## 5.1. Определение структуры вычислительной схемы

В этом пункте полученная в предыдущем разделе модель будет применяться для определения типа вычислительной схемы (серверная, клиентская или гибридная) при выполнении запрошенных операций.

Исходно будем основываться на предположении о том, что для каждого типа устройства $m_i \in M$, где $M$ — множество всех известных типов устройств, существуют и определены в процессе работы системы априорные значения гиперпараметров $\alpha_i$ и $\beta_i$ распределения параметра $\lambda_i^{apriori}$.

Поскольку принципиальным критерием, по которому должны выбираться тот или иной подход к реализации вычислений, является время выполнения операции, необходимо учитывать как время работы системы, так и время пересылки данных по сети. В модели системы, предложенной в предыдущем разделе, оба типа временных затрат учитываются при оценке единственного параметра — ожидаемой частоты выполнения операций. При продолжительном времени работы системы априорная частота для определенного типа устройств будет усредненной как по их мощности, так и по сетевому соединению, характерному для них. В рамках определения схемы вычислений такой подход может быть не совсем корректным: из-за возможного расхождения в параметрах сети для похожих устройств, относящихся к одному классу, при ориентировании только лишь на априорные

сведения время сетевой передачи растровых данных может быть сильно недооценено.

Эта проблема имеет как минимум два решения: предоставление возможности конечному пользователю самому дать изначальную информацию о типе и скорости своего соединения и оценка его при помощи отправки какого-либо набора тестовых данных. Первый случай достаточно тривиален, но допускает возможность человеческой ошибки, поэтому может быть скомбинирован со вторым для достижения уверенности в результате.

Дальнейшая работа с устройством зависит от того, насколько наши ожидания от работы с ним совпадают с реальностью. Для определения степени этого совпадения применим следующую процедуру:

1. Устройству пользователя посылается небольшое тестовое «задание», имеющее априорную оценку времени работы $\frac{N}{\lambda_i^{apriori}}$, где $N$ — количество элементарных операций внутри тестового блока, в которых измерялась сложность операции внутри статистической модели.
2. Измеряется реальное время работы над тестовыми данными $T_{test}$.
3. Выдвигается гипотеза о принадлежности полученного значения к генеральной совокупности, подчиняющейся гамма-распределению с параметрами $\alpha_{test} = N$ и $\beta_{test} = \frac{N}{\lambda_i^{apriori}}$.
4. При выполнении условий применимости априорного распределения гиперпараметры $\alpha_i$ и $\beta_i$ применяются для дальнейшего построения апостериорной модели устройства, в противном случае — отбрасываются и модель работы устройства строится «с нуля».

Таким образом, посланные тестовые данные могут быть использованы сразу для двух поставленных целей: определения качества сетевого соединения и построения статистической модели устройства. При этом объем

переданных данных должен быть достаточно небольшим, чтобы не сильно повлиять на общее время работы, которое в этом случае должно максимально возможным образом состоять из времени пересылки данных по сети.

Будем считать, что на данном этапе модель устройства определена (либо из априорных соображений, либо путем ее перестроения в случае неудовлетворительной точности априорной модели), как определено и качество сетевого соединения. В качестве критерия оптимальности работы системы примем общее время решения некоторой поставленной задачи $V$, которую можно разбить на независимые подзадачи $v_i$, которые, в свою очередь, могут быть распространены по узлам вычислительной сети. Задача $V$ состоит из запросов пользователей к системе, и результаты ее выполнения требуется переслать на соответствующие узлы сети. Под распространением подзадачи $v_i$ будем понимать пересылку по сети как информации о самой подзадаче, так и данных, над которыми будут производиться манипуляции. Будем полагать, что пропускной канал головных серверов с данными достаточно широк для удовлетворения потребности в их пересылке на вычислительные узлы в параллельном режиме. Тогда общее время работы над задачей $V$ будет равно максимуму времени обработки своей подзадачи среди всех вычислительных узлов, что и предстоит минимизировать.

Подзадача $v_i$, выполняемая $i$-ым устройством в сети, состоит из суммы независимых подзадач, результат части из которых необходим на самом узле, а остальные необходимо переслать другим участникам сети напрямую или через головной сервер. Обозначим эти части как $v_{i,s}$ и $v_{i,ns}$; $v_{i,s} + v_{i,ns} = v_i$. Тогда время работы произвольного узла над своей задачей можно представить в виде суммы времен (см. Рис. 1):

- пересылки по сети подзадачи, результат которой необходим на самом узле;
- работы над этой частью задачи;

- пересылки по сети подзадачи, результат выполнения которой необходим на другом узле;
- работы над ней;
- минимуму из времен пересылки результатов, необходимых на других узлах:
    - непосредственно;
    - через один из центральных узлов.

В формульном виде это можно представить следующим образом:

$$T_i = t^{\leftarrow}_{i,N} v_{i,s} + \frac{v_{i,s}}{\lambda_i} + t^{\leftarrow}_{i,N} v_{i,ns} + \frac{v_{i,ns}}{\lambda_i} +$$

$$+\delta \min\left[\sum_j t_{ij,N} v_{ij,ns}\,,\quad t^{\rightarrow}_{i,N} v_{i,ns} + \sum_j t^{\leftarrow}_{j,N} v_{ij,ns}\right], \qquad (1)$$

где

- $t^{\leftarrow}_{i,N}$ — время пересылки единицы данных по сети с одного из центральных узлов на $i$-ый вычислительный узел;
- $t^{\rightarrow}_{i,N}$ — время пересылки единицы данных по сети с $i$-ого узла на один из центральных (в общем случае $t^{\leftarrow}_{i,N} \neq t^{\rightarrow}_{i,N}$);
- $\delta$ — коэффициент, показывающий, как в среднем соотносится количество информации в ответе с ней же в задаче; поскольку под задачами в рамках данной работы подразумевается работа с растрами, этот коэффициент равен $\frac{1}{\bar{n}}$, где $\bar{n}$ — среднее количество растров, оперирование над которыми в конечном итоге даст один растр, содержащий результаты вычислений. Применимость такого «усредненного» подхода к отношению количества информации в задаче и результате следует из статистического характера применяемой модели, призванной работать в условиях большого числа как вычислительных узлов, так и данных;

- $t_{ij,N}$ — время пересылки единицы данных по сети с $i$-ого узла на $j$-ый (в общем случае $t_{ij,N} \neq t_{ji,N}$);
- $v_{ij,ns}$ — часть данных, обрабатываемых на $i$-ом узле, результат выполнения операций над которыми ожидается на $j$-ом.

Центральные выделенные сервера также входят во множество проиндексированных вычислительных узлов; их особенность состоит в нулевых значениях $t_{i,N}^{\leftarrow}$ и $t_{i,N}^{\rightarrow}$.

Сформированный вектор $T = (T_i)$ представляет собой набор времен выполнения выданных заданий на всех узлах. Поскольку операции на различных узлах выполняются параллельно и независимо друг от друга, в качестве функционала качества на данном этапе имеет смысл выбрать максимум по $i$ из всех $T_i$ (задача $V$ считается выполненной тогда, когда закончены все вычисления на всех узлах):

$$\mathbb{U}(\mathbf{V}) = \max_i T_i. \qquad (2)$$

Здесь за $\mathbf{V}$ обозначена матрица, элементами которой являются переменные $v_{ij}$, отвечающие за количество данных, работа над которыми будет произведена на $i$-ом узле, а результаты необходимо переслать на $j$-ый узел. Эта матрица является параметром, по которому может быть произведена оптимизация.

Заметив, что все элементы вектора $T$ являются неотрицательными, выражение $(2)$ можно переписать как векторную норму:

$$\mathbb{U}(\mathbf{V}) = \|T\|_\infty,$$

где $\|\cdot\|_\infty$ обозначает стандартную норму в Евклидовом пространстве вида

$$\|T\|_\infty = \max_i |T_i|.$$

При этом саму задачу можно описать как попытку минимизации этого функционала:

$$\mathbb{U}(\mathbf{V}) \underset{\mathbf{V}}{\rightarrow} min.$$

Явный вид функционала зависит от выражения, заменяющего собой функцию минимума из (1). Саму задачу его минимизации можно решать различными способами, но независимо от его выбора элементы вектора $T$ имеет смысл упростить. Рассмотрим для начала случай, когда для $i$-го вычислительного узла выбран режим работы, при котором он пересылает результаты вычислений непосредственно на другие узлы, минуя центральный сервер:

$$T_i = t_{i,N}^{\leftarrow} v_{i,s} + \frac{v_{i,s}}{\lambda_i} + t_{i,N}^{\leftarrow} v_{i,ns} + \frac{v_{i,ns}}{\lambda_i} + \delta \sum_j t_{ij,N} v_{ij,ns} =$$

$$\left(t_{i,N}^{\leftarrow} + \frac{1}{\lambda_i}\right)\left(v_{i,s} + v_{i,ns}\right) + \delta \sum_j t_{ij,N} v_{ij,ns}.$$

В векторно-матричном виде:

$$\left(diag(t_N^{\leftarrow} + \lambda^{-1})\,\mathbf{V} + (\delta \mathbf{t}_N \circ \mathbf{V})\right)\mathbb{1}.$$

Здесь

- $t_N^{\leftarrow}$ — вектор, состоящий из $t_{i,N}^{\leftarrow}$;
- $\lambda^{-1}$ — вектор, состоящий из $\frac{1}{\lambda_i}$;
- $diag(t_N^{\leftarrow} + \lambda^{-1})$ обозначает диагональную матрицу, на главной диагонали которой находятся элементы вектора $t_N^{\leftarrow} + \lambda^{-1}$, а остальные являются нулями;
- $\mathbb{1}$ — вектор-столбец из единиц;
- $\mathbf{t}_N$ — матрица, на позиции $(i, j)$ которой стоит время пересылки единицы данных с $i$-го узла на $j$-ый (очевидно, по диагонали матрицы стоят нули);
- за $\circ$ обозначено произведение Адамара.

В случае, если на $i$-ом вычислительном узле выбран режим работы, при котором он пересылает результаты вычислений на центральный узел, выражение преобразовывается несколько по-другому:

$$T_i = t_{i,N}^{\leftarrow} v_{i,s} + \frac{v_{i,s}}{\lambda_i} + t_{i,N}^{\leftarrow} v_{i,ns} + \frac{v_{i,ns}}{\lambda_i} + \delta\left(t_{i,N}^{\rightarrow} v_{i,ns} + \sum_j t_{j,N}^{\leftarrow} v_{ij,ns}\right) =$$

$$\left(t_{i,N}^{\leftarrow} + \frac{1}{\lambda_i}\right)\left(v_{i,s} + v_{i,ns}\right) + \delta t_{i,N}^{\rightarrow} v_{i,ns} + \delta \sum_j t_{j,N}^{\leftarrow} v_{ij,ns}.$$

В векторно-матричном виде это выражение удобно представить, разложив матрицу $\mathbf{V}$ на сумму матриц $\mathbf{V}_s$ и $\mathbf{V}_{ns}$, где $\mathbf{V}_s$ состоит из главной диагонали матрицы $\mathbf{V}$ с нулями на всех недиагональных элементах, а матрица $\mathbf{V}_{ns}$ аддитивно дополняет ее до матрицы $\mathbf{V}$. В этом случае его можно переписать в виде

$$diag(t_N^{\leftarrow} + \lambda^{-1})(\mathbf{V}_s + \mathbf{V}_{ns})\mathbb{1} + \delta diag(t_N^{\rightarrow})\mathbf{V}_{ns}\mathbb{1} + \mathbf{V}_{ns}t_N^{\leftarrow}.$$

Обобщением рассмотренных выше случаев является гибридный вариант, при котором часть данных передается напрямую узлу, нуждающемуся в них, а часть при этом проходит через один из центральных узлов. В этом случае, воспользовавшись равенством

$$\sum_j v_{ij,ns} = v_{i,ns}$$

выражение (1) можно переписать:

$$T_i = t_{i,N}^{\leftarrow} v_{i,s} + \frac{v_{i,s}}{\lambda_i} + t_{i,N}^{\leftarrow} v_{i,ns} + \frac{v_{i,ns}}{\lambda_i} +$$

$$+\delta \min\left[\sum_j t_{ij,N} v_{ij,ns}, \quad t_{i,N}^{\rightarrow} v_{i,ns} + \sum_j t_{j,N}^{\leftarrow} v_{ij,ns}\right] =$$

$$= \left(t_{i,N}^{\leftarrow} + \frac{1}{\lambda_i}\right)\left(v_{i,s} + v_{i,ns}\right) +$$

$$+\delta \min\left[\sum_j t_{ij,N} v_{ij,ns}\,,\quad \sum_j \left(t^{\rightarrow}_{i,N} + t^{\leftarrow}_{j,N}\right) v_{ij,ns}\right].$$

Далее его можно усилить (в силу необходимости его дальнейшей минимизации):

$$\left(t^{\leftarrow}_{i,N} + \frac{1}{\lambda_i}\right)\left(v_{i,s} + v_{i,ns}\right) +$$

$$+\delta \min\left[\sum_j t_{ij,N} v_{ij,ns}\,,\quad \sum_j \left(t^{\rightarrow}_{i,N} + t^{\leftarrow}_{j,N}\right) v_{ij,ns}\right] \geqslant$$

$$\geqslant \left(t^{\leftarrow}_{i,N} + \frac{1}{\lambda_i}\right)\left(v_{i,s} + v_{i,ns}\right) +$$

$$+\delta \sum_j \min[t_{ij,N},\quad t^{\rightarrow}_{i,N} + t^{\leftarrow}_{j,N}]\, v_{ij,ns}.$$

Окончательно, обозначив $\min[t_{ij,N},\quad t^{\rightarrow}_{i,N} + t^{\leftarrow}_{j,N}]$ за $\widetilde{t_{ij,N}}$, можно записать

$$T_i = \left(t^{\leftarrow}_{i,N} + \frac{1}{\lambda_i}\right)\left(v_{i,s} + v_{i,ns}\right) + \delta \sum_j \widetilde{t_{ij,N}} v_{ij,ns},$$

что в векторно-матричном представлении дает

$$\left(diag\left(t^{\leftarrow}_{N} + \lambda^{-1}\right)\mathbf{V} + \left(\delta \widetilde{\mathbf{t}_N} \circ \mathbf{V}\right)\right)\mathbb{1}, \qquad (3)$$

где за $\widetilde{\mathbf{t}_N}$ обозначена матрица, на позиции $(i,j)$ которой стоит $\widetilde{t_{ij,N}}$.

### 5.1.1. Решение формализованной задачи

Несмотря на очевидную линейность получившихся во всех трех случаях выражений, только вариант передачи данных через сервер не предполагает использования произведения Адамара. Кроме того, матрица переменных всегда домножается на константы с обеих сторон. Тем не менее, сама по себе задача является типичной задачей на минимакс и может быть решена стандартными методами (вплоть до перебора по конечной сетке в пространстве допустимых значений передаваемой информации) при помощи

какой-либо системы компьютерной алгебры, поддерживающей решение задач на условную оптимизацию.

### Scilab

Scilab в своей базовой комплектации не имеет возможностей оптимизации функций с ограничениями, но в одном из свободно распространяемых пакетов — FOSSEE — существует специальный инструментарий, предназначенный для оптимизации. В частности, в арсенале тулбокса имеются функции `fmincon` и `fminimax`. `fmincon` позволяет решить следующую задачу (все обозначения соответствуют официальной документации):

$$\min_x f(x)$$
$$A \cdot x \leqslant b$$
$$Aeq \cdot x \leqslant beq$$
$$c(x) \leqslant 0$$
$$ceq(x) = 0$$
$$lb \leqslant x \leqslant ub.$$

Здесь $f(x)$, $c(x)$ и $ceq(x)$ могут быть нелинейными. `fminimax` позволяет решать задачу следующего вида:

$$\min_x \max_i F_i(x)$$
$$c(x) \leqslant 0$$
$$ceq(x) = 0$$
$$A \cdot x \leqslant b$$
$$Aeq \cdot x = beq$$
$$lb \leqslant x \leqslant ub.$$

Таким образом, обе эти функции могут быть использованы для решения поставленной задачи. Стоит отметить, что `fminimax`, с одной стороны, лучше

описывает задачу и, в теории, должна более качественно работать благодаря учету специфики недифференцируемой и нелинейной функции максимума, но в технической документации указано [50], что на текущий момент `fminimax` интернально использует `fmincon`, которая, в свою очередь, использует алгоритм ip-opt (Interior Point OPTimizer), не специализированный для работы с функцией максимума.

### Maxima

В инструментарии Maxima имеется метод COBYLA (Constrained Optimization BY Linear Approximations), позволяющий решать задачи подобного типа. Постановка задачи описывается несколько иначе, чем в Scilab, и содержит только нелинейные ограничения:

$$\begin{aligned} &\min_x F(x) \\ &s_i(x) \leqslant h_i^s(x), \qquad i = 1..M_s \\ &g_i(x) \geqslant h_i^g(x), \qquad i = 1..M_g \\ &e_i(x) = h_i^e(x), \qquad i = 1..M_e. \end{aligned}$$

Хотя поставленная задача и может быть описана с использованием данных обозначений (являющихся достаточно общими), техническая реализация метода, использующая линейные приближения как целевой функции, так и ограничений, ставит адекватность использования данного метода под вопрос. Хотя ограничения и отдельные компоненты вектора $T$ и представляют собой линейные комбинации переменных, недиффиренцируемость функции максимума и ограничений на значения отдельных переменных не позволяют применять линейную аппроксимацию вблизи точек экстремума функции и границ допустимого множества.

### JACAL

Пакет JACAL не содержит необходимого для решения задачи оптимизации с ограничениями инструментария. С использованием синтаксиса

языка SCHEMA может быть написано расширение для него, но в рамках текущей работы такой вариант не рассматривается как не соответствующий целям.

### GiNaC

GiNaC также не поддерживает возможности работы с оптимизационными функциями, что отражено в официальной документации.

### YACAS

YACAS не имеет встроенных функций для оптимизации с ограничениями в частности и для оптимизации вообще.

### Octave

В своей стандартной комплектации пакет Octave имеет встроенную поддержку некоторых оптимизационных методов, одним из которых является метод `sqp` (sequential quadratic programming, последовательное квадратичное программирование). Он позволяет решать задачи вида

$$\min_x \varphi(x)$$
$$g(x) \geqslant 0$$
$$h(x) = 0.$$

Специфика последовательного квадратичного программирования требует от пользователя при запросе на решение проблемы определить, помимо целевой функции, функцию градиента и гессиан от нее. В случае отсутствия этой информации оценка градиента и гессиана производится методом конечных разностей.

Помимо стандартного набора функций, в СКА Octave существует отдельный инструментарий для оптимизации, который можно дополнительно установить с использованием Octave Forge [51]. Он имеет своей целью расширить функционал Octave до возможностей соответствующего тулбокса

для системы Matlab, но на данный момент его функциональность практически никак не имплементирована (в частности, отсутствует реализация функций `fmincon` и `fminimax`) [52], хотя разработка ведется уже достаточно давно.

### 5.1.2. Приведение к задаче смешанного целочисленного программирования

Более аккуратное решение задачи можно получить, учитывая ее специфику, а именно — линейность по всем переменным. Одним из возможных преобразований является «вытягивание» матрицы $\mathbf{V}$ в вектор $v$ по правилу:

$$v_{in+j} = v_{ij,ns},$$

где $n$ — число узлов в сети. Далее, поскольку гибридная задача может быть заменой $\widetilde{t_{ij,N}}$ на $t_{ij,N}$ или $t_{i,N} + t_{j,N}$ преобразована как к задаче распределения нагрузки при непосредственной передаче данных, так и к ее варианту с использованием одного из центральных серверов, преобразуем только выражение (3). Матрицу $diag(t_N^{\leftarrow} + \lambda^{-1})_{n\times n}$ заменим на матрицу $\mathrm{H}^1 = \left(h_{ij}^1\right)_{n\times n^2}$ по правилу

$$h_{i,in+j}^1 = t_{i,N}^{\leftarrow} + \frac{1}{\lambda_i}$$

для всех $i, j$ от 0 до $n$; оставшиеся элементы матрицы установим равными нулю. Произведение Адамара матриц $\widetilde{\mathbf{t}_N}$ и $\mathbf{V}$, каждая из которых имеет размерность $n \times n$, домноженное на вектор из единиц, также имеющий размерность $n$, заменим на матрицу $\mathbf{H}^2 = \left(h_{ij}^2\right)_{n\times n^2}$ по правилу

$$h_{i,in+j}^2 = \widetilde{t_{ij,N}}.$$

Аналогично предыдущему случаю, $i$ и $j$ меняются от $0$ до $n$, а оставшиеся элементы матрицы устанавливаются равными нулю. Теперь, с использованием новых обозначений, можно переписать (3):

$$T = \mathbf{H}^1 v + \mathbf{H}^2 v.$$

Введя матрицу $\mathbf{H}$ как сумму матриц $\mathbf{H}^1$ и $\mathbf{H}^2$, можно переписать выражение (2), используя новые обозначения:

$$\mathbb{U}(v) = \|\mathbf{H}v\|_\infty.$$

Задача минимизации

$$\mathbb{U}(v) \underset{v}{\rightarrow} min$$

должна быть решена при условии

$$\sum_{i=1}^{n} v_{ij} = V_j, \qquad j = 1..n,$$

вытекающем из требования каждого пользователя полностью решить поставленную им задачу и только ее. Она может быть сведена к задаче линейного смешанного целочисленного программирования классическим способом [53] посредством введения искусственной переменной $t$, представляющей собой максимальное время работы и передачи информации на всех устройствах:

$$\min t$$

$$t - \sum_{j=1}^{n} h_j^i v_{ij} \geqslant 0, \qquad i = 1..n$$

$$\sum_{i=1}^{n} v_{ij} = V_j, \qquad j = 1..n$$

$$v_{ij} \geqslant 0, \qquad i, j = 1..n$$

$$v_{ij} \in \mathbb{Z},$$

где $h^i$ — строки матрицы **H**.

Особенностью данной задачи является отсутствие прямой необходимости нахождения оптимального решения: в силу принципиальной хаотичности как поведения устройств по отдельности, так и пропускной способности сети нас устроит любое решение, близкое к оптимальному. В силу NP-полноты задачи целочисленного программирования, этим стоит воспользоваться для облегчения поиска «удовлетворительного» решения. Во-первых, следует решить соответствующую линейно ослабленную задачу, а затем воспользоваться одним из методов:

- при решении задачи методом «ветвей и границ» после наперед заданного числа итераций при условии, что процесс поиска решения еще не завершен, начинать проверять решение на близость к оптимальному решению соответствующей задачи линейного программирования и при достаточно высокой точности завершать работу алгоритма, не доводя его до конца;
- округлить все полученные значения переменных к ближайшему большему целому, после чего
    1. найти все $j$ такие, что
    $$\sum_{i=1}^{n} v_{ij} > V_j; \qquad (4)$$
    2. уменьшить на единицу значение $v_{ij}$ для узла с наибольшим значением $\widetilde{t_{\iota J,N}} + \frac{1}{\lambda_i}$;
    3. в случае, если условие (4) все еще выполняется, уменьшить значение $v_{ij}$ для следующего узла с наибольшим значением $\widetilde{t_{\iota J,N}} + \frac{1}{\lambda_i}$ без учета первого; этот шаг необходимо проделать не более $n$ раз в сумме для всех узлов, поскольку в любом случае при

округлении всех переменных в большую сторону выполняется неравенство

$$\sum_{i,j=1}^{n} v_{ij} \leqslant n + \sum_{j=1}^{n} V_j ;$$

4. как только решение начнет удовлетворять ограничениям, закончить процесс и принять его в качестве оптимального.

Несмотря на то, что первый метод имеет под собой лучшую теоретическую основу, второй может работать быстрее, поскольку NP-трудная задача не решается в принципе: вместо нее решается задача линейного программирования и выполняется не более $n$ дополнительных шагов для нахождения близкого к оптимальному решения.

Свободные системы компьютерной алгебры, рассматриваемые в данной работе, имеют следующие возможности для решения задачи в новой постановке:

### Maxima

В составе СКА Maxima имеется модуль Simplex, содержащий имплементацию симплекс-метода, позволяющую решать задачи линейного программирования. В частности, метод `minimize_lp` [54] решает задачу минимизации функции при линейных ограничениях и имеет возможность задания неотрицательности отдельных переменных. Отличительной особенностью Maxima от всех остальных исследованных оптимизационных пакетов является возможность задания ограничений в явном виде, то есть в виде набора неравенств. Во всех остальных системах для возможности вызова требуемого функционала необходимо задавать матрицы коэффициентов, что не всегда является наиболее удобным способом описания проблемы в силу их большой размерности и сильной разреженности. Стоит отметить, что при использовании матричного способа задания ограничений Maxima может

работать только с задачей в канонической форме, то есть ответственность за грамотное введение слабых и искусственных переменных ложится на плечи пользователя системы.

В то же время, на данный момент в Maxima отсутствуют методы решения задач целочисленного и смешанного целочисленного линейного программирования, хотя соответствующий запрос на добавление функционала существует в системе с 2013 года [55].

### Scilab

Как уже было отмечено выше, в своей стандартной комплектации Scilab не поддерживает решение оптимизационных задач, но в состав пакета FOSSEE, помимо прочего, входят и методы решения задач линейного программирования. Метод `linprog` позволяет решить задачу в стандартном виде:

$$\min_x C^T \cdot x$$
$$A \cdot x \leqslant b$$
$$Aeq \cdot x \leqslant beq$$
$$lb \leqslant x \leqslant ub.$$

Для решения задачи смешанного целочисленного программирования в том же модуле существует два интерфейса доступа к процедуре Clp [56]: `symphony` и `symphonymat` (второй интерфейс приближен к интерфейсу `linprog`). К ограничениям задачи линейного программирования добавлены ограничения вида

$$x_i \in \mathbb{Z}, \qquad i \in I,$$

где множество $I$ задается отдельно, что и позволяет решать задачу в смешанной постановке. Помимо функционала FOSSEE, к Scilab может быть

подключен модуль `lp_solve` [57], не имеющий к нему непосредственного отношения и рассмотренный далее отдельно.

### Octave

Octave имеет встроенный метод `glpk`, позволяющий решать задачу линейного программирования как в канонической, так и в стандартной форме. К сожалению, задать целочисленность всех или отдельных переменных нельзя, и единственный способ решения задачи целочисленного линейного программирования в рамках Octave состоит в ее линковке со сторонними библиотеками, такими как `lp_solve`.

### YACAS, GiNaC и JACAL

Указанные системы компьютерной алгебры не имеют встроенных средств для решения задачи линейного или целочисленного линейного программирования, как и не существует для них дополнительных подключаемых модулей.

### lp_solve

lp_solve — свободно-распространяемый под лицензией LGPL [58] набор библиотек для решения задачи смешанного целочисленного линейного программирования, изначально разрабатывавшийся в технологическом университете Эйндховена, но впоследствии развившийся в самостоятельный проект. Он может быть использован как в виде отдельно стоящей легковесной IDE, так и в качестве подключаемой библиотеки.

### Выводы

Из рассмотренных свободных систем компьютерной алгебры полноценно решить поставленную задачу смешанного целочисленного программирования способен лишь Scilab, но по простоте линковки он значительно проигрывает отдельно стоящей библиотеке lp_solve, которая,

будучи написанной на C++, имеет оболочки, позволяющие использовать ее и в других языках программирования. В то же время, внутри lp_solve реализован и обычный симплекс-метод, позволяющий решить задачу в ослабленной форме, что позволяет использовать эту библиотеку как универсальную в рамках поставленной задачи.

## 5.2. Реализация отложенных вычислений

В случае произведения вычислений на стороне сервера необходимо учитывать специфику выдачи результатов пользователю: например, при запросе на произведение вычислительной операции над всем слоем, пользователю системы, скорее всего, не потребуются данные, полученные для всех тайлов в режиме минимально доступного масштабирования. Имея сведения об области, просматриваемой в данный момент и о предпочтениях пользователя в целом, можно существенно уменьшить число выполняемых операций, которые, тем не менее, с высокой вероятностью будут удовлетворять потребностям пользователя системы в данный момент.

Предположим, что в систему обработки запросов пришел новый запрос вида

$$f(A_1 * S_1, \dots, A_n * S_n) \to S_f,$$
$$x, y \in (x_{min}, y_{min}; x_{max}, y_{max}) = M_q,$$

где $A_i$ — один из доступных в системе запросов операторов над данными в слое, $S_i$ — один из слоев, над которыми производятся вычислительные операции, $f(\cdot)$ — некоторая функция, аккумулирующая в некотором смысле данные, полученные со всех участвующих в запросе слоев и производящая новый результирующий слой $S_f$, который и должен быть отправлен пользователю. Помимо запроса о пользователе известны уровень приближения и координаты прямоугольника, который он просматривает в данный момент:

$$z, (x_1, y_1), (x_2, y_2).$$

Обозначим за $M_s$ множество просматриваемых точек, а за $M_s^\sigma$ — расширение этого множества по горизонтали и вертикали в $\sigma$ раз. Обозначим за $(M, z)$ множество точек с координатами из $M$, рассматриваемых на уровне приближения $z$. В этом случае вместо выполнения операции по всему множеству точек $M_q$ запрос можно выполнить на множестве

$$\bigcup_{k=1}^{k_{max}} \left(M_q \cap M_s^{k\sigma}, z - k + 1\right) \cup \left(M_q \cap M_s, z + 1\right),$$

обеспечивая, таким образом, сбалансированные вычисления по всем уровням «пирамиды» с $z - k_{max}$ до $z$ включительно и затрагивая $z + 1$-ый уровень в текущей просматриваемой области.

Изменение пользователем масштаба в данной ситуации следует рассматривать как повторный запрос с теми же параметрами, но новыми значениями $z$, $(x_1, y_1)$ и $(x_2, y_2)$. При этом закэшированные результаты предыдущих запросов позволят не выполнять вычисления в области

$$\bigcup_{k=1}^{k_{max}} \left(M_q \cap M_s^{k\sigma}, z - k + 1\right),$$

ограничившись лишь произведением новых вычислений над еще не видимой пользователю области данных $\left(M_q \cap M_s, z + 1\right)$.

В случае, когда данные, необходимые клиенту, обсчитываются на клиентской же стороне, отсутствует явно выраженная необходимость в экономии вычислительной мощности. Тем не менее, описанный выше способ в данном контексте позволит пользователю быстрее получать нужные данные, производя необходимые расчеты «на лету». Вместе с этим, при расчете данных на клиентской машине можно использовать стандартные подходы «ленивых

вычислений» с использованием пригодных для этого языков программирования. В рамках данной работы этот подход отдельно не рассматривается.

# 6. Оптимизация пользовательских запросов

Одной из задач, встающих перед архитектором любой системы, позволяющей пользователям задавать практически произвольные операции над данными в рамках предоставленных примитивов, всегда является автоматизация упрощения подобных запросов. Это в полной мере справедливо и для геоинформационных систем и, в частности, для O-GIS. Пользователь системы имеет возможность производить необходимые вычисления над имеющимися в его распоряжении растровыми полями и массивами данных, декларируя операции в строковом виде.

Спецификой, предлагаемой в рамках создаваемого модуля реализации вычислений, является использование нативного кода на C++ для обработки растровых операций на вычислительных узлах. Поскольку в случае невозможности использования системы компьютерной алгебры из C++ кода появляется необходимость либо в использовании дополнительной программной прослойки, обеспечивающей взаимодействие системы компьютерной алгебры с кодом, ответственным за непосредственное исполнение растровых операций, неизбежно ведущая к уменьшению производительности системы, одним их параметров исследования существующих продуктов является именно возможность взаимодействия с C++ кодом. Другим вариантом решения проблемы является оптимизация запросов перед их отправкой на вычислительный узел, что так же негативно скажется на производительности системы в целом.

## 6.1. Обзор возможностей символьных вычислений в свободных СКА

В O-GIS в списке поддерживаемых операций находятся взятие синуса, косинуса, тангенса, котангенса, извлечения квадратного корня, возведение в степень, сложение, вычитание, деление и умножение [59]. В связи с этим возможности упрощения символьных выражений в различных системах компьютерной алгебры рассматриваются в контексте работы с этими операциями.

### 6.1.1. Maxima

Возможности Maxima по символьному упрощению выражений крайне богаты, и включают в себя не только возможности упрощения выражений, включающих элементарные операции, но и более гибкую систему упрощения функциональных выражений при помощи утилизации различных известных свойств неявно заданных или не заданных вовсе функций.

Для упрощения выражений, содержащих дроби и тригонометрические выражения, из арсенала Maxima лучше всего подходят функции `ratsimp`, `trigsimp` и `expand`.

### 6.1.2. Scilab

В Scilab существует отдельный инструментарий для работы с символьными вычислениями — SciMax. Его задача заключается в подключении модулей из Maxima в среду исполнения команд Scilab, поэтому, помимо необходимости дополнительной линковки, в части выполнения символьных операций (в том числе и их упрощения) Scilab полностью эквивалентен Maxima.

### 6.1.3. JACAL

JACAL отличается от других СКА в том, что формат входных данных совпадает с выходными. Он не имеет отдельных операций упрощения выражений, а вместо этого упрощает их в режиме «по умолчанию» без возможностей настройки этого процесса. При тестировании его возможностей было обнаружено, что он упрощает рациональные выражения, но при этом игнорирует возможности символьного упрощения тригонометрических. В связи с этим JACAL не может быть полноценно использован в рамках настоящей работы.

### 6.1.4. GiNaC

К сожалению, разработчики GiNaC сознательно отказались от имплементации алгоритмов, позволяющих упрощать символьные выражения. Хотя в пакете и присутствуют некоторые возможности манипуляций с символьными выражениями, которые можно использовать с целью их упрощения, они крайне лимитированы и не предназначены для использования в этих целях. Конечно, как и всякий другой проект с открытыми исходными кодами, GiNaC дает возможность пользователю самостоятельно реализовать недостающую функциональность, но в рамках данной работы это не представляется рациональным решением, с учетом отсутствия необходимости произведения сложным манипуляций над пользовательскими запросами.

### 6.1.5. YACAS

YACAS имеет встроенную поддержку упрощения символьных выражений. Для целей, указанных в данном разделе, наиболее полезными функциями являются `Simplify`, `TrigSimpCombine`, `Expand` и `RadSimp`, отдельно упрощающая выражения, содержащие радикалы.

### 6.1.6. Octave

Ядро Octave не позволяет выполнять символьные операции, и инструментарий для работы с ними существует в качестве одного из пакетов Octave Forge (symbolic package), что усложняет линковку. Кроме того, этот пакет во время использования фактически обращается к интерпретатору языка Python, что тоже отрицательно сказывается как на простоте архитектуры системы, так и на ее быстродействии.

### 6.1.7. Выводы

Таким образом, из рассмотренных систем компьютерной алгебры наиболее подходящей для задач символьного упрощения выражений является YACAS, поскольку в его ядре уже содержатся необходимые функции, а его линковка возможна как с приложением, написанным на Java, так и с нативным C++-приложением в качестве статической или динамической библиотеки.

# 7. Использование ГПУ для выполнения задач растровой алгебры

Как уже было отмечено в параграфе Параллельная обработка растровых данных внутри одной машины, специфика операций, производимых в рамках геоинформационной системы O-GIS позволяет предположить, что использование на вычислительных узлах мощностей графических процессоров, при их наличии, может дать прирост производительности. Поскольку это утверждение не следует непосредственно из априорных данных и не имеет под собой математической основы, необходимо провести тестирование с целью установления степени его истинности в рамках поставленной задачи.

## 7.1. Тестирование

Тестирование производилось на нескольких устройствах, центральные процессоры которых поддерживали SIMD инструкции, а графические процессоры могли быть использованы для произвольных вычислений, а не только для работы с двух- и трехмерной графикой через низкоуровневое API.

Целью тестирования являлось подтверждение или опровержение утверждения о целесообразности использования мощностей графического процессора при выполнении операций растровой алгебры.

В качестве тестовых данных был взят массив чисел одинарной точности с плавающей запятой размером 256 мегабайт, являющийся аналогом растра размером 64 мегапикселя.

Для выполнения операций с использованием SIMD инструкций были использованы библиотеки libsimdpp [60] (для арифметических операций и взятия квадратного корня) и FastTrigo [61] (для вычисления тригонометрических выражений). Вычисления на графическом процессоре

производились при помощи NVIDIA CUDA: поскольку на всех устройствах стояли графические процессоры этого производителя, библиотека OpenCL обращалась к CUDA API и фактически дублировала ее возможности. На графических процессорах для выполнения операций над растрами запрашивалось 1024 потока.

Таблица 1 показывает среднее (после 10 повторов) время выполнения операций с использованием соответствующих технологий на разных устройствах.

| Операция | Способ вычисления | Intel Pentium G4560<br>NVIDIA GeForce GTX 1050 | Intel Core i5-2450M<br>NVIDIA GeForce GT 630M | Intel Core 2 Duo E8400<br>NVIDIA GeForce GTX 550 Ti | Intel Core i5-5200U<br>NVIDIA GeForce GTX 850M |
|---|---|---|---|---|---|
| Сложение | Скалярный | 99 мс | 151 мс | 317 мс | 166 мс |
| | AVX | 101 мс | 162 мс | 344 мс | 173 мс |
| | CUDA | 177 мс | 303 мс | 570 мс | 227 мс |
| Умножение | Скалярный | 99 мс | 152 мс | 325 мс | 166 мс |
| | AVX | 102 мс | 159 мс | 336 мс | 167 мс |
| | CUDA | 175 мс | 308 мс | 584 мс | 227 мс |
| Вычитание | Скалярный | 96 мс | 153 мс | 297 мс | 170 мс |
| | AVX | 98 мс | 162 мс | 295 мс | 169 мс |
| | CUDA | 173 мс | 307 мс | 523 мс | 226 мс |
| Деление | Скалярный | 96 мс | 159 мс | 304 мс | 164 мс |
| | AVX | 99 мс | 164 мс | 299 мс | 166 мс |
| | CUDA | 172 мс | 304 мс | 564 мс | 228 мс |

| | | | | | |
|---|---|---|---|---|---|
| Взятие квадратного корня | Скалярный | 85 мс | 171 мс | 283 мс | 152 мс |
| | AVX | 87 мс | 178 мс | 285 мс | 150 мс |
| | CUDA | 171 мс | 305 мс | 691 мс | 265 мс |
| Взятие синуса | Скалярный | 168 мс | 277 мс | 374 мс | 217 мс |
| | AVX | 168 мс | 252 мс | 372 мс | 228 мс |
| | CUDA | 170 мс | 307 мс | 593 мс | 265 мс |
| Взятие косинуса | Скалярный | 184 мс | 295 мс | 432 мс | 236 мс |
| | AVX | 156 мс | 234 мс | 343 мс | 225 мс |
| | CUDA | 168 мс | 307 мс | 626 мс | 263 мс |
| Взятие тангенса | Скалярный | 309 мс | 528 мс | 720 мс | 408 мс |
| | AVX | 253 мс | 420 мс | 509 мс | 353 мс |
| | CUDA | 172 мс | 307 мс | 637 мс | 264 мс |
| Взятие котангенса | Скалярный | 303 мс | 495 мс | 666 мс | 393 мс |
| | AVX | 247 мс | 416 мс | 441 мс | 383 мс |
| | CUDA | 172 мс | 308 мс | 564 мс | 263 мс |
| Сложное выражение | Скалярный | 952 мс | 1011 мс | 1305 мс | 1230 мс |
| | AVX | 443 мс | 714 мс | 834 мс | 588 мс |
| | CUDA | 181 мс | 311 мс | 689 мс | 324 мс |
| Среднее время копирования данных на графический процессор и обратно | | 165 мс | 305 мс | 560 мс | 228 мс |

Таблица 1. Время выполнения операций над растрами.

## 7.2. Анализ результатов тестирования

Из полученных данных следует несколько выводов:

1. Выполнение операций над растрами на графическом процессоре не всегда дает прирост производительности, а иногда и, наоборот, замедляет вычисления. Во всех случаях вычисления на ЦПУ оказывались эффективнее для элементарных арифметических операций и взятия квадратного корня.

2. С усложнением выполняемых операций преимущества использования графического процессора начинают проявляться, хотя на различных системах для этого требуется различная степень сложности. На более современных устройствах ускорение вычислений начинает проявляться раньше.

3. Основное время, уходящее на выполнение операции с использованием графического процессора — это накладные расходы на передачу данных из оперативной памяти в память устройства и обратно.

Кроме того, не для всех операций обнаружилось преимущество явного использования SIMD инструкций перед обычным скалярным кодом. Это связано с оптимизациями кода, выполняемыми компилятором автоматически.

Указанные выше утверждения позволяют заключить, что для элементарных пользовательских запросов следует использовать SIMD инструкции процессора, нежели мощности графического процессора. В то же время, для более сложных запросов решение об использовании или неиспользовании графического процессора должно зависеть от конкретного вычислительного узла, но для комплексных запросов его использование может быть оправдано почти всегда.

# Выводы

Основными результатами, достигнутыми при построении прототипа вычислительного модуля для O-GIS, являются следующие:

- Благодаря построенной математической модели разработан алгоритм, позволяющий статистически оптимально распределить нагрузку внутри гетерогенной вычислительной сети с учетом специфики выполняемых операций растровой алгебры;
- С использованием техник «мозаики» и «пирамиды» построен ключ шардирования, позволяющий равномерно распределить растровые данные по кластеру и производить операции чтения в параллельном режиме вне зависимости от области запроса;
- Использование техник шардирования и репликации баз данных, хранящих метаинформацию, непосредственно растровые данные, а также использование набора серверов, отвечающих за перенаправление запросов внутри СУБД, обеспечивает отсутствие «узких» мест в процессе выполнения вычислений, возможность масштабирования системы и защиту от выхода из строя отдельных центральных узлов;
- Учет специфики пользовательских запросов позволяет проводить их оптимизацию и производить отложенные вычисления, что, в свою очередь, позволяет избежать лишних вычислений;
- Использование технологии векторизации вычислений и мощностей графических процессоров позволяет для сложных запросов в разы увеличить скорость вычислений на конечных узлах по сравнению со скалярной версией.

# Заключение

В рамках данной работы разработан прототип вычислительного модуля, предназначенного для выполнения операций растровой алгебры в рамках Географической Информационной Системы O-GIS.

Примененные техники распределенных вычислений являются достаточно универсальными в том смысле, что могут быть применены практически к произвольным вычислительным системам, к которым может быть применена соответствующая математическая теория из разделов «Оптимальное распределение текущей нагрузки между хостами в вычислительной сети» и «Определение структуры вычислительной схемы». В то же время методы, описанные в разделах «Распределение данных по кластеру» и «Реализация отложенных вычислений», являются специфичными и применимыми только к растровым данным, имеющим картографическую природу.

Техническая реализация модуля доступна в виде проекта с открытым исходным кодом в виде репозитория на github.com [62].

# Список литературы


1. Описание спутника WorldView-3. http://www.ball.com/aerospace/Aerospace/media/Aerospace/Downloads/D3088-WV3_2.pdf
2. Scholten H. J. Geographical information systems for urban and regional planning. / ed. by J. Stillwell. Vol. 17. Springer Science & Business Media, 2013. 276 p.
3. Gu Y. G., Wang Z. H., Lu S. H., Jiang S. J., Mu D. H., Shu Y. H. Multivariate statistical and GIS-based approach to identify source of anthropogenic impacts on metallic elements in sediments from the mid Guangdong coasts, China. // Environmental pollution, 2012. Vol. 163. P. 248–255.
4. Yeh, A. G. O. Urban planning and GIS. // Geographical information systems, 1999. Vol. 2. P. 877–888.
5. Yi C. S., Lee J. H., Shim M. P. GIS-based distributed technique for assessing economic loss from flood damage: pre-feasibility study for the Anyang Stream Basin in Korea. // Natural hazards, 2010. Vol. 55 (2). P. 251–272.
6. Афонин А. Н., Севрюков С. Ю., Соловьев П. А., Лунева Н. Н. Веб-ГИС для решения задач эколого-географического анализа и моделирования: новые возможности // Вестник Санкт-Петербургского университета. Том 7. Геология. География. 2016. Вып. 4. С. 97–111.
7. Fernández-Baca D. Allocating modules to processors in a distributed system. // IEEE Transactions on Software Engineering, 1989. Vol. 15 (11). P. 1427–1436.
8. Chen Q., Wang L., Shang Z. MRGIS: A MapReduce-Enabled high performance workflow system for GIS. // IEEE Fourth International Conference on eScience. IEEE, 2008. P. 646–651.

9. Kafil M., Ahmad I. Optimal task assignment in heterogeneous distributed computing systems. // IEEE concurrency, 1998. Vol. 6 (3). P. 42–50.
10. Armstrong R., Hensgen D., Kidd T. The relative performance of various mapping algorithms is independent of sizable variances in run-time predictions. // 7th IEEE Heterogeneous Computing Workshop (HCW '98), 1998. P. 79–87.
11. Ibarra O. H., Kim C. E. Heuristic algorithms for scheduling independent tasks on nonidentical processors. // Journal of the ACM (JACM), 1997. Vol. 24 (2). P. 280–289.
12. Freund R. F., Gherrity M., Ambrosius S., Campbell M., Halderman M., Hensgen D., Keith E., Kidd T., Kussow M, Lima J. D., Mirabile F., Moore L., Rust B., Siegel H. J. Scheduling resources in multi-user, heterogeneous, computing environments with SmartNet. // 7th IEEE Heterogeneous Computing Workshop (HCW '98), 1998. P. 184–199.
13. Веремей Е. И. Линейные системы с обратной связью: учебное пособие. СПб.: Изд. «Лань», 2013. 448 с.
14. Буре В. М., Парилина Е. М. Теория вероятностей и математическая статистика. СПб.: Лань, 2013. 416 с.
15. Гардинер К. Стохастические методы. Том 4. Берлин: Springer, 2009. 439 с.
16. Gupta G. Scaling web Services by offloading expensive computation to a many-core GPGPU using CUDA. // International Conference on Computing and Network Communications (CoCoNet). IEEE, 2015. P. 516–519.
17. Kim J., Dao T. T., Jung J., Joo J., Lee J. Bridging OpenCL and CUDA: a comparative analysis and translation. // SC-International Conference for High Performance Computing, Networking, Storage and Analysis. IEEE, 2015. P. 1–12.

18. Su C. L., Chen P. Y., Lan C. C., Huang L. S., Wu K. H. Overview and comparison of OpenCL and CUDA technology for GPGPU. // IEEE Asia Pacific Conference on Circuits and Systems (APCCAS), 2012. P. 448–451.
19. Официальная документация СУБД MySQL. https://dev.mysql.com/doc/refman/5.7/en/spatial-analysis-functions.html
20. Описание формата Well Known Text в GIS Wiki. http://giswiki.org/wiki/Well_Known_Text
21. Официальное описание проекта пространственной СУБД PostGIS. http://postgis.net/docs/manual-2.0/
22. Официальное описание проекта MySQL Cluster. https://dev.mysql.com/doc/index-cluster.html
23. Официальная документация проекта Citus. https://docs.citusdata.com/en/v7.3/
24. Официальный сайт проекта Postgre-XL. https://www.postgres-xl.org
25. Официальное описание проекта NoSQL СУБД MongoDB. https://docs.mongodb.com/manual/core/geospatial-indexes/index.html#calculation-of-geohash-values-for-2d-indexes
26. Geospatial Performance Improvements in MongoDB 3.2. https://www.mongodb.com/blog/post/geospatial-performance-improvements-in-mongodb-3-2
27. Интервью с Элиотом Хоровитцем о причинах создания MongoDB. https://medium.com/s-c-a-l-e/mongodb-co-creator-explains-why-nosql-came-to-be-and-why-open-source-mastery-is-an-elusive-goal-3a138480b9cd
28. Официальное описание проекта пространственной СУБД SpatialLite. http://www.gaia-gis.it/spatialite
29. Проект rqlite на GitHub. https://github.com/rqlite/rqlite
30. Официальное описание проекта NoSQL СУБД CouchDB. http://couchdb.apache.org/

31. Clarence J. M. T., Aravindh S., Shreeharsha A. B. Comparative study of the new generation, agile, scalable, high performance NOSQL databases. 2012. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.258.9602
32. Официальная страница проекта CouchDB Lounge. http://tilgovi.github.io/couchdb-lounge
33. Официальная страница проекта Maxima. http://maxima.sourceforge.net
34. Официальный сайт проекта Scilab. https://scilab.in
35. Страница проекта JACAL на сайте MIT. http://people.csail.mit.edu/jaffer/JACAL
36. Официальный сайт проекта YACAS. http://www.yacas.org
37. Официальный сайт проекта GiNaC. https://www.ginac.de
38. Ответы на часто задаваемые вопросы о GiNaC, https://ginac.de/FAQ.html
39. Официальный сайт проекта Octave. https://www.gnu.org/software/octave/
40. Bajerski P. Optimization of geofield queries. // Proceedings of the 1st International Conference on Information Technology, P. 1–4. IEEE, 2008.
41. Carabaño J., Westerholm J., Sarjakoski T. A compiler approach to map algebra: automatic parallelization, locality optimization, and GPU acceleration of raster spatial analysis. // GeoInformatica. 2017. P. 1–25.
42. Lomont C. Introduction to Intel® Advanced Vector Extensions. https://software.intel.com/en-us/articles/introduction-to-intel-advanced-vector-extensions
43. Malakar R., Vydyanathan N. A CUDA-enabled Hadoop cluster for fast distributed image processing. // National Conference on Parallel Computing Technologies. IEEE, 2013. P. 1–5.
44. Stone, J. E., Gohara, D., & Shi, G. (2010). OpenCL: A parallel programming standard for heterogeneous computing systems. // Computing in science & engineering. No 12 (3). P. 66–73.

45. Oracle Spatial GeoRaster Developer’s Guide: http://docs.oracle.com/cd/B28359_01/appdev.111/b28398/geor_intro.htm
46. Xu Hong, Mangtani P. Managing imagery and raster data using mosaic datasets // ESRI International User Conference. Technical Workshop, 2012.
47. Dhar S. From outsourcing to Cloud computing: evolution of IT services // Management Research Review. 2012. No 35 (8). P. 664–675.
48. Богданов А. В., Е Мьинт Найг. Сравнение нескольких платформ облачных вычислений // Вестник Санкт-Петербургского университета. Прикладная математика. Информатика. Процессы управления. 2013. Т. 10. № 2. С. 40–50.
49. Nov O., Anderson D., Arazy O. Volunteer computing: a model of the factors determining contribution to community-based scientific research // Proceedings of the 19th international conference on World wide web. 2010. P. 741–750.
50. Описание работы функции fminimax в официальной документации Scilab. https://scilab.in/fossee-scilab-toolbox/optimization-toolbox/functions/fminimax
51. Страница проекта Octave Forge на SourceForge. https://octave.sourceforge.io
52. Описание оптимизационного тулбокса на официальном сайте Octave. https://wiki.octave.org/Optimization_package
53. Roughgarden T. The Minimax Theorem and Algorithms for Linear Programming. http://theory.stanford.edu/~tim/w16/l/l10.pdf
54. Официальная документация Maxima. http://maxima.sourceforge.net/docs/manual/de/maxima_68.htm
55. #2543 Feature request: Extend simplex module to allow Integer Linear Programs (ILP) https://sourceforge.net/p/maxima/bugs/2543/
56. Описание оптимизационного тулбокса на официальном сайте Scilab. https://scilab.in/fossee-scilab-toolbox/optimization-toolbox/functions
57. Using lpsolve from Scilab. http://lpsolve.sourceforge.net/5.5/Scilab.htm

58. Официальная документация проекта lpsolve на SourceForge. http://lpsolve.sourceforge.net/5.5/
59. Репозиторий проекта O-GIS на GitHub. https://github.com/Solovyev-Pavel/o-gis
60. Репозиторий библиотеки libsimdpp на GitHub. https://github.com/p12tic/libsimdpp
61. Репозиторий библиотеки FastTrigo на GitHub. https://github.com/divideconcept/FastTrigo
62. Репозиторий проекта на GitHub. https://github.com/bestawperever/RasterComputation