Перевод статьи:
Ponomarenko A.A., Malkov Yu.A., Logvinov A.A., Krylov V.V. An overlay network for distributed exact and range search in one-dimensional space
Business Informatics. 2016. No. 1 (35). P. 26–36.

# Оверлейная сеть для распределенного точного и интервального поиска в одномерном пространстве

**А.А. Пономаренко**
*научный сотрудник, Лаборатория алгоритмов и технологий анализа сетевых структур*
*Национальный исследовательский университет «Высшая школа экономики»*
*Адрес: Российская Федерация, 603093, г. Нижний Новгород, ул. Родионова, 136*
*E-mail: aponomarenko@hse.ru*

**Ю.А. Мальков**
*младший научный сотрудник*
*Институт прикладной физики, Российская академия наук*
*Адрес: Российская Федерация, 603950, г. Нижний Новгород, ул. Ульянова, 46*
*E-mail: yurymalkov@mail.ru*

**А.А. Логвинов**
*руководитель проектов, ООО «МераЛабс»*
*Адрес: Российская Федерация, 603093, г. Нижний Новгород, ул. Родионова, 192*
*E-mail: alogvinov@meralabs.com*

**В.В. Крылов**
*доктор технических наук, профессор, заведующий лабораторией больших данных*
*Нижегородский государственный технический университет им. Р.Е. Алексеева*
*Адрес: Российская Федерация, 603950, г. Нижний Новгород, ул. Минина, 24*
*E-mail: vkrylov@heterarchica.com*

*Для современной компьютерной системы возможность масштабироваться является необходимым бизнес-требованием. Распределенные системы явно демонстрируют это свойство, как и способность легко обрабатывать сверхбольшие объемы данных. Многие системы с распределенной архитектурой имеют в основе распределенную хэш-таблицу (distributed hash table, DHT), которая используется в качестве дополнительной логической сети, объединяющей множество распределенных серверов. Эта логическая сеть используется для поиска узлов и распределения задач между ними. Главное преимущество такого подхода – отсутствие какого-либо центрального элемента или узла, который обладал бы информацией о структуре всей сети. Поиск узлов в сети происходит путем передачи поискового сообщения от одного узла к другому. Несмотря на то, что каждый узел «знает» только небольшое число других узлов, сеть организованна таким образом, что процедура поиска затрагивает логарифмическое число узлов.*

*Существует несколько реализаций концепции DHT, которые регламентируют, каким образом логическая сеть строится и поддерживается. В этой работе мы демонстрируем, как такая сеть может быть сконструирована очень простым способом, с применением (с небольшими изменениями) недавно опубликованного алгоритма метризованного тесного мира (Metrized Small World) для случая одномерного метрического пространства. В работе мы приводим теоретический анализ для случая равномерного распределения данных и эмпирический анализ для других распределений. Главным преимуществом предложенного алгоритма перед аналогами является его устойчивость к различным распределениям данных и отсутствие необходимости поддержания распределения длин ссылок, определенного явным образом.*

*Также в работе описывается, каким образом можно полностью отделить физическое размещение данных от поисковой функциональности. Это разделение важно, например, для построения глобальных хранилищ данных, данными которых владеют несколько сторон, причем каждая сторона заинтересована в том, чтобы иметь полный контроль над физическим размещением и доступу к своим данным. В отличие от DHT, добавление новых данных не требует их перемещение на другие узлы.*

Перевод статьи:
Ponomarenko A.A., Malkov Yu.A., Logvinov A.A., Krylov V.V. An overlay network for distributed exact and range search in one-dimensional space
Business Informatics. 2016. No. 1 (35). P. 26–36.

**Ключевые слова:** поиск ближайшего соседа, метрическое пространство, распределенные вычисления, Интернет-технология и приложния, структура данных, алгоритм.

**Цитирование:** Ponomarenko A.A., Malkov Yu.A., Logvinov A.A., Krylov V.V. An overlay network for distributed exact and range search in one-dimensional space // Business Informatics. 2016. No. 1 (35). P. 26–36. DOI: 10.17323/1998-0663.2016.1.26.36.

## Введение

Масштабируемость (расширяемость, *scalability*) компьютерной системы – это ее способность справляться с растущим объемом работы [10]. Также под этим термином понимают способность системы к увеличению производительности при повышающейся нагрузке. Аналогичное значение используется в экономическом контексте, когда расширяемость предполагает, что лежащая в основе бизнес-модель обеспечивает возможность для экономического роста в пределах компании. Использование концепции масштабируемости желательно и в технологии, и в создании бизнеса.

Центральное место в любой компьютерной системе занимает хранилище данных. Расширяемость компьютерной системы в значительной степени зависит от способности к расширению ее хранилища данных. Традиционно в большинстве случаев разработчики используют в качестве хранилища данных реляционные СУБД (*RDBMS*). К сожалению, расширяемость РСУБД имеет свойственные им ограничения [11]. Причина заключается в том, что для поддержки транзакций и их согласованности РСУБД нуждаются в центре координации – таком как менеджер транзакций, который с ростом числа серверов, на которых размещена РСУБД, становится узким местом. Более того, большинство РСУБД изначально имеют архитектуру, в которой серверы используют одинаковые копии данных, которые в свою очередь, должны быть синхронизированы, а это при большом количестве серверов – трудная задача.

Для решения проблем масштабируемости было решено пожертвовать некоторой частью функциональности в пользу более высокой производительности [19]. Так появились базы данных *NOSQL*. Наиболее важный класс баз данных *NOSQL* – это хранилища «ключ-значение». Такие системы главным образом поддерживают две операции: поиск и хранение данных по заданному ключу. На таких хранилищах «ключ-значение» основаны многие популярные интернет-сервисы. Например, система обмена сообщениями *Facebook* и хранилище сообщений системы *SoundCloud* используют *Apache Cassandra*; многие продукты *Google* – такие как *Gmail, YouTube, Google Maps* – основаны на системе *BigTable*, которая также является хранилищем «ключ-значение».

Одна из возможностей реализовать хранилище «ключ-значение» – это использование распределенных хэш-таблиц (*DHT*). *DHT* позволяют хранить и осуществлять поиск данных, на основании заданного ключа, среди множества сетевых узлов [16]. Конкретная реализация *DHT* – это протокол, которые регламентирует, как узлы связываются между собой и как они формируют логическую сеть. Важно то, что в каждом узле отсутствует информация о полной топологии всей сети. В каждом узле в таблице маршрутизации хранится только небольшой список *IP*-адресов других узлов. Каждому узлу ставится в соответствие ключ – *id*, который обычно является хэш-значением, вычисляемым на основе *IP*-адреса. Данные с ключом $k$ помещаются в узел, *id* которого ближе к $k$, чем *id* других узлов с точки зрения некоторой функции расстояния $d$. Поиск выполняется «жадным» алгоритмом, путем запросного сообщения от узла к узлу на основе списка узлов, хранимых в каждом узле в таблице маршрутизации. В различных реализациях таблицы *DHT* используются разные функции расстояния. Например, это может быть простое расстояние между двумя числами $x$ и $y$ $d(x, y) = (x - y) \bmod n$ [8] или *XOR*-метрика, определяемая как $d(x, y) = x \oplus y$ [14].

Главное преимущество всех реализаций таблицы *DHT* состоит в том, что ожидаемое число узлов, которое запросное сообщение должно пройти, пока оно достигнет узла назначения, пропорционально $\log(n)$, где $n$ – общее число узлов в сети. Более того, максимальный размер таблицы маршрутизации в любом узле также пропорционален $\log(n)$. Совместно оба свойства делают *DHT* расширяемой. Так вставка новых узлов обеспечивается очень небольшими накладными расходами на общую производительность системы.

Секрет этих двух свойств лежит в структуре таблицы маршрутизации, которая поддерживается в каждом узле. Главная идея реализаций *DHT* – хранить таблицы маршрутизации таким образом, чтобы

*i*-запись (связь) в таблице узла *A* указывала на узел *B*, до которого расстояние от *A* было бы в интервале $2^i < d(A,B) \le 2^{i+1}$. Нестрого говоря, такой способ формирования связей соответствует степенному закону распределения вероятностей длины связей $P \sim d^{-1}$. При добавлении нового узла, *DHT* формирует таблицу маршрутизации нового узла и обновляет таблицы существующих узлов в соответствии с этим распределением. Все узлы вместе со своими таблицами маршрутизации формируют логическую сеть, которая может быть представлена графом *G*(*V, E*), где множество вершин *V* – это множество узлов сети, а множество ребер графа и *E* , соответствует записям в таблицах маршрутизации. Таким образом, любая конкретная реализации регламентирует способ построения и поддержания графа *G*.

Как упоминалось выше, все реализации *DHT* формируют *i-ю* запись в таблице маршрутизации узла *A* явным образом, путем поиска такого узла *B*, что $2^i < d(A,B) \le 2^{i+1}$ , для того чтобы построить граф, в котором «жадный» алгоритм может найти узел с заданным ключом за *log*(*n*) шагов.

В данной статье мы описываем алгоритм, который конструирует граф с подобными свойствами более естественно и более просто без явной поддержки конкретного распределения длины связей. Мы применяем наш алгоритм для приближенного поиска в общем метрическом пространства [12, 13], с незначительной модификацией алгоритма вставки для случая точного поиска в одномерном случае. К сожалению, теоретический анализ алгоритма поиска в общем случае «Метризованного тесного мира» все еще отсутствует. В данной статье в разделе 4 мы представляем такой анализ для одномерного случая. Эмпирическое исследование степенного распределения, независимость от распределения данных, анализ устойчивости и зависимость сложности алгоритма поиска от его параметров мы приводим в разделе 5.

Следует заметить, что ранее в работе [18] был упомянут алгоритм для одномерного случая в качестве основного алгоритма для построения мультиатрибутной структуры данных. В работе [18] ошибочно ссылаются на работу под заголовком «Single-attribute Distributed Metrized Small World Data Structure», где этот алгоритм должен был быть детально описан, однако эта работа по ошибке не была опубликована. В данной мы устраняем данный пробел. С целью согласованности мы представляем этот алгоритм в разделе 2. Мы детально описываем алгоритм добавления и поиска вместе с подробным эмпирическим и теоретическим анализом.

Мы также предлагаем, как полностью разделить концепцию расположения данных и функциональность поиска. Это разделение важно, например, для построения глобально распределенных хранилищ, где данные принадлежат многим участникам, и каждый участник заинтересован аспектами контроля физического хранения и доступа к своим собственным данным. Таким образом, в отличие от *DHT*, вставка новых данных не требует перемещения данных. В этом случае предложенная логическая сеть вместе с алгоритмами поиска и добавления может рассматриваться как структура данных, построенная на множестве данных.

Статья структурирована следующим образом. В разделе 1 описывается структура данных «Метризованный тесный мир» и способ ее отображения на множество сетевых узлов. В разделе 2 описывается алгоритм добавления. В разделе 3 приводятся алгоритмы поиска. В разделе 4 мы показываем, что алгоритм поиска имеет сложность *O*(*log n*). В разделе 5 приведены экспериментальные данные, и дается анализ свойств, предложенной структуры. В «Заключении» мы резюмируем наши результаты.

## 1. Одномерная логическая сеть «Метризованный тесный мир»

Логическая сеть «Метризованный тесный мир» (*MSW*) состоит из множества узлов сети, соединенных физическим каналом, так что каждый узел сети может связываться непосредственно с другим узлом. Каждый узел содержит некоторое количество блоков данных. И узлы, и блоки данных имеют уникальные идентификаторы в контексте всей структуры. Идентификаторы узлов преобразуются в сетевые адреса (например, *DN* в *IP*) внешней системой (такой как *DNS*). Идентификатор блока данных содержит идентификатор узла, в котором этот блок находится, и это обеспечивает мгновенную идентификацию сетевого узла, содержащего блок данных. Это дает возможность создавать на уровне данных логическую структуру, которая, по сути, отображается на физические сетевые узлы. Каждый блок данных *A* хранит изменяющееся множество *N*(*A*) идентификаторов других блоков данных, которые являются связями, составляющие логическую сеть. Логическая сеть может рассматриваться как граф *G*(*V,E*), в которой множество *V* соответствует множеству всех блоков данных, а множество *E* содержит ребра (*A, B*), если блок данных *A* имеет связь с блоком данных *B*.

Все связи – двунаправленные, т.е. каждый блок

данных связан с блоками данных, соединенных с ним. Узлы не напрямую знают друг о друге; идентификаторы других узлов могут быть получены только по связям с блоками данных, расположенными в этих узлах.

Иллюстрация структуры приведена на *рис. 1.* Структура распределена среди трех сетевых узлов, идентифицированными уникальными *URL*-адресами. Блоки данных расположенные в этих узлах также имеют уникальные *URL*-адреса, которые являются *URL*-подадресами узлов. Логическая сеть построена над блоками данных; узлы не имеют прямых связей друг с другом. Алгоритм добавления и поиска блока данных действует исключительно в логической сети, сформированной блоками данных, и связи между ними никак не связанные с физической структурой сети, поддерживающей структуру (идентификаторы блоков данных рассматриваются алгоритмом как атомарные). Выбор узла для размещения нового блока данных или решение по добавлению нового узла полностью не зависят от алгоритма добавления данных, который выполняется после того, как будет размещен новый блок данных. Добавление нового физического сетевого узла не оказывает воздействия на существующую логическую сеть или на распределение блоков данных между физическими узлами; оно просто обеспечивает новое возможное место для размещения данных и расширяет обрабатывающий ресурс для алгоритмов добавления и поиска.

Обращение к логической сети происходит локальным итеративным методом – начиная с произвольно известного блока данных (точки входа) и следуя по связям, содержащимся в посещенных

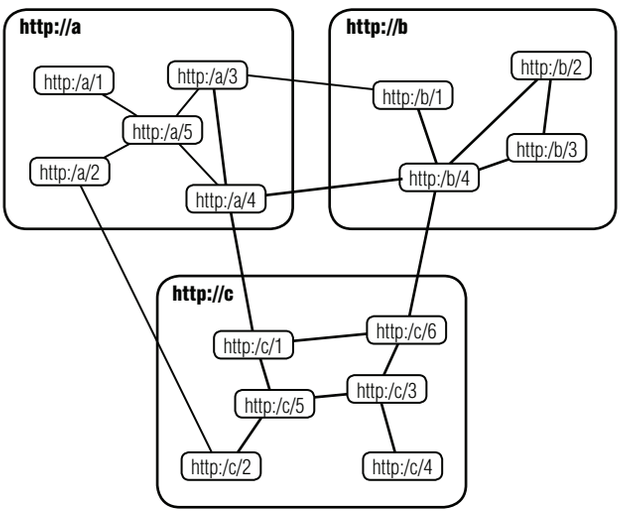


*Рис. 1.* Логическая сеть, построенная на уровне блоков данных

блоках данных. Для нахождения соответствующего блока данных следует выбрать точку входа и следовать через структуру, выбирая связь на каждом шаге.

Для обеспечения эффективности этого процесса должны быть выполнены два требования:

1) Должен иметься относительно короткий путь между точкой входа и искомым блоком данных. Поскольку заранее неизвестны ни точка входа, ни целевой блок данных, в структуре короткий путь должен существовать между любой парой блоков данных.

2) Должен существовать критерий, который на каждом шаге позволяет выбрать алгоритму связь, которая может приблизить процесс поиска к требуемому результату. Желательно также, чтобы в структуре градиент близости соответствовал самому короткому пути к искомому элементу и по возможности отсутствовали локальные минимумы.

Для выполнения первого требования мы разработали алгоритм добавления блока данных, который пошагово строит структуру со свойствами «тесного мира». Графы «тесного мира» [3, 4, 5] обладают важным свойством: они имеют короткий путь между двумя вершинами, хотя большинство вершин не являются непосредственными соседями друг к другу.

Для выполнения второго требования блок данных содержит поисковую информацию. В данной статье в качестве поисковой информации мы считаем множество объектов $U$, которые могут быть биективно отображены на единичный интервал действительных чисел. Например, $U$ может быть множеством всех возможных строк $S = (s_1, ..., s_n)$, где $s_i$ – индекс символа в алфавите $B$, если в качестве отображения мы используем функцию

$$C(S) = \sum_{i=1}^{n} \frac{s_i}{|B|^i}.$$

Для целей описания алгоритмов мы обозначим в качестве $C(D_i)$ образ поисковой информации блока данных $D_i$. Мы использовали простую метрику $M$ между блоками данных, вычисляемую как $M(D_i, D_j) = |C(D_i) - C(D_j)|$. Эта метрика используется как для алгоритма добавления, так и для алгоритма поиска. Такая простая модель поисковой информации позволяет нам осуществлять точный поиск, поскольку на множестве действительных чисел легко задать отношение линейного порядка согласованный с метрикой $M$. На основе отношения линейного порядка алгоритм добавления способен для каждого блока данных найти точных

соседей Вороного. Это – основное преимущество этой модели в отличие от приблизительного поиска в общем метрическом пространстве [1, 12, 13], в которых нахождение точных соседей Вороного – сложная задача.

Несмотря на ограниченный класс решаемых задач, этот подход полезен для хранилищ «ключ-значение» и систем, подобных *CAS* (Адресуемому по содержимому хранилищу) [6], где идентификатор данных однозначно определяется контентом и инвариантен к размещению, например, хэш-значение вычисляемое от контента. С точки зрения структуры *MSW*, хэш-значение, вычисляемое от контента, является поисковой информацией, а контент – непоисковой. В результате процесса поиска хэш-значение от контента будет трансформировано в идентификатор блока данных, который содержит данный контент.

Другие подобные решения – это распределенные хэш-таблицы (*DHT*) [7], например, *Chord* [8]. Эти системы используют подход хэширования [9] для принятия решения, какой физический сетевой узел будет хранить блок данных с определенным ключом. Добавление нового физического сетевого узла требует в среднем перемещение *K/n* блоков данных, где *K* – общее число блоков данных и *n* – число узлов. В противовес, в системе *MSW* добавление нового физического узла не оказывает воздействия на существующие узлы, поскольку размещение блоков данных полностью не зависимо от алгоритмов поиска и добавления. Новый физический узел становится включенным в структуру, когда блок данных, размещаемый в новом узле, добавляется в логическую сеть *MSW* с использованием точки входа в одном из существующих узлов. Другое отличие от систем *DHT* заключается в том, что система *MSW* создает логическую сеть на уровне блоков данных, в то время как логическая сеть систем *DHT* (такая как *Chord*) строится на уровне физических сетевых узлов.

Детальное описание технологических аспектов расширяемой системы базы данных, основанных на структуре *MSW*, приведено в работе [2].

## 2. Алгоритм добавления данных

Цель алгоритма добавления данных – соединить вновь размещенный блок данных со структурой путем расширения его списка связей и соответствующего расширения списка связей блоков данных, выбранных для соединения с новым блоком данных так, чтобы «жадный» алгоритм смог найти любой блок данных, начиная с любого блока данных. Когда имеется отношение порядка *R*, согласованное с метрикой (например, естественный порядок вещественных чисел), мы можем сделать «жадный» поиск гарантированным, путем соединения каждого блока данных с его прямым предшественником $D_{pre}$ и прямым преемником, в отличие от приближенного поиска в общем метрическом пространстве. Для того чтобы сделать алгоритм поиска более быстрым, мы дополнительно соединяем новый блок данных с *m* другими ближайшими блоками данных и при этом не удаляем старые связи с развитием структуры.

Ниже представлен псевдокод алгоритма:

### *Insertion_with_Order* (Вставка_по_порядку)

**Input:** $D_{new}$ – новый блок данных; $D_{ep}$ – точка входа блока данных; *m* – параметр (небольшое натуральное число)

1. Присвоить $D_{cur} = D_{ep}$.
2. Для каждого $D \in N(D_{cur})$ вычислить $D \in N(D_{cur})$.
3. Если $min(M_i < M(D_{cur}, D_{new})$, присвоить $D_{cur} = D_i$, для которого $M_i = min(M_i)$, и перейти к шагу 2.
4. Если $C(D_{cur}) < C(D_{new})$, присвоить $D_{pre} = D_{cur}$, и присвоить $D_{succ}$ равным прямому преемнику $D_{new}$, выбранного из соседей $D_{cur}$.
5. Если $C(D_{cur}) > C(D_{new})$, присвоить $D_{succ} = D_{cur}$ и присвоить $D_{pre}$ равным прямому предшественнику $D_{new}$, выбранного из соседей $D_{cur}$.
6. Соединить $D_{new}$ с $D_{pre}$ и $D_{succ}$, если они существуют.
7. Повторить *m* раз:

   7.1 Если существует $D_{pre}$, присвоить $D'_{pre}$ равным прямому предшественнику $D_{pre}$, выбранного из множества соседей.

   7.2 Если существует $D_{succ}$, присвоить $D'_{succ}$ равным прямыму преемнику $D_{succ}$, выбранного из множества соседей.

   7.3 Если не существует ни одного $D'_{pre}$ и $D'_{succ}$, то прекратить процесс.

   7.4 Если существует только $D'_{pre}$ или $M(D'_{pre}, D_{new}) < M(D'_{succ}, D_{new})$, соединить $D_{new}$ и $D'_{pre}$, и присвоить $D_{pre} = D'_{pre}$.

   7.5 Если существует только $D'_{succ}$ или $M(D'_{succ}, D_{new}) < M(D'_{pre}, D_{new})$, соединить $D_{new}$ и $D'_{succ}$ и присвоить $D_{succ} = D'_{succ}$.

На каждой итерации алгоритм получает соседей текущего блока данных $D_{cur}$, ближайшего к $D_{new}$, и вычисляет расстояние $M_i$ до каждого из них. После этого алгоритм обновляет $D_{cur}$ и выполня-

ет следующую итерацию. Таким образом, мы используем разновидность «жадного» алгоритма для нахождения блока данных, ближайшего к новому блоку данных (после всех итераций ближайшим к блоку данных $D_{new}$ будет находиться в $D_{cur}$). Мы находим, является ли $D_{cur}$ предшественником или преемником $D_{new}$, и соединяем новый блок данных с его прямым предшественником $D_{pre}$ и прямым преемником $D_{succ}$. В конце мы соединяем новый блок данных с $m$ ближайшими блоками данных, находящимися справа от него (блоком данных-преемником) и слева от него (предшествующим блоком данных).

### 3. Алгоритм поиска

«Жадный» алгоритм поиска находит блок данных с поисковой строкой, ближайшей к заданной строке $S_q$, начиная с блока данных $D_{ep}$ – точки входа. Если имеется блок данных с поисковой строкой, идентичной $S_q$, то «жадный» алгоритм возвратит этот блок данных. Для краткости мы будем рассматривать каждый блок данных, как если бы он был той же сущностью, что и доступная к поиску строка, которую он содержит.

***Greedy_Search* (Жадный_поиск)**

**Input:** $S_q$ – ключ; $D_{ep}$ – блок данных в точке входа;

**Return:** блок данных, который является ближайшим к $S_q$.

1. Присвоить $D_{cur} = D_{ep}$.
2. Для каждого $D \in N(D_{cur})$ вычислить $M_i = M(D_i, S_q)$.
3. Если $min(M_i > M(D_{cur}, S_q)$, возвратить $D_{cur}$.
4. Присвоить $D_{cur} = D_i$ для каждого $M_i = min(M_i)$ и перейти к шагу 2.

«Жадный» алгоритм поиска, описанный выше, опирается на тот факт, что каждый блок данных всегда связан со своим прямым предшественником и преемником. Это обеспечивает отсутствие локальных минимумов в структуре. В разделе 4 показано, что сложность алгоритма поиска растет логарифмически с увеличением количества блоков данных в структуре.

Поскольку все блоки данных упорядочены, есть возможность определить операцию, которая осуществляет поиск всех блоков данных из заданного интервала. Мы обозначим прямого предшественника элемента $x$ как $D_{pre}(x)$ и прямого преемника $x$ как $D_{suc}(x)$. Ниже приведен псевдокод диапазонного поиска.

***Range_ Search* (Диапазонный_поиск)**

**Input:** $Lq$ – левая граница; $R_q$ – правая граница; $D_{ep}$ – точка входа;

**Return:** множество блоков данных из интервала $[L_q, R_q]$.

1. Присвоить $D_{cur} = Greedy\ Search(\frac{L_q + R_q}{2}, D_{ep})$
2. $T = \varnothing$; $D'_{cur} = D_{cur}$
3. Пока $D_{cur} \geq L_q$, присвоить $T = T \cup D_{cur}$ и присвоить $D_{cur} = D_{pre}(D_{cur})$;
4. Принять $D_{cur} = D_{suc}(D'_{cur})$.
5. Пока $D_{cur} \leq R_q$, присвоить $T = T \cup D_{cur}$ и присвоить $D_{cur} = D_{suc}(D_{cur})$;
6. Возвратить $T$.

Идея алгоритма поиска в заданном диапазоне очень проста. Мы начинаем из центра интервала, идем влево, пока не достигнем левой границы, и собираем все блоки данных во множество $T$. После этого мы делаем то же самое в противоположном направлении, пока не достигнем правой границы, и возвращаем в качестве результата множество $T$.

### 4. Теоретический анализ алгоритма поиска

**Теорема.** Алгоритм «жадного» поиска имеет сложность $O(\log n)$ в структуре, сформированной алгоритмом построения *Insertion_with_Order*.

**Доказательство**

Предположим, что образы $C(D_i)$ равномерно распределены в интервале $[0,1]$. Наша цель – показать, что каждый раз, когда размер структуры удваивается, среднее число шагов, требующихся для достижения любого блока данных из любого другого блока данных, увеличивается на величину, ограниченную константой, тем самым обеспечивая логарифмический рост сложности поиска. Рассмотрим ситуацию, когда у нас в структуре имеется множество $N$ первых блоков данных. Назовем это множество первым поколением. Предположим, что мы можем достичь любой блок данных из любого другого блока данных максимально за $P$ шагов. Плотность вероятности расстояния $L$ между соседними блоками данных (с точки зрения метрики) падает экспоненциально, так что, если мы проигнорируем интервалы между соседними блоками данных, в $R$ раз большие среднего значения (т.е. больше $R/N$), то мы упускаем только экспоненциально малое число случаев. Тем самым мы игнорируем только редкие случаи очень больших промежутков между соседними блоками данных в первом поколении.

Перевод статьи:
Ponomarenko A.A., Malkov Yu.A., Logvinov A.A., Krylov V.V. An overlay network for distributed exact and range search in one-dimensional space
Business Informatics. 2016. No. 1 (35). P. 26–36.

Затем мы добавляем $N$ блоков данных второго поколения, т.е. удваиваем число блоков данных. Рассмотрим самый большой интервал между соседними блоками данных в первом поколении. Число $k$ блоков данных второго поколения, попадающих в один интервал, подчиняется распределению Пуассона

$$p(k) = \frac{\lambda^k}{k!} e^{-\lambda}$$ для больших значений $N$.

Поскольку средняя длина интервала в первом поколении равна $R/N$, мы имеем ожидаемое значение числа $n_{II}$ блоков данных второго поколения, попадающих в интервал $\lambda = R$, и, следовательно,

$$p(k) = \frac{R^k}{k!} e^{-R}.$$

Если мы предположим, что $n_{II} < M$ , мы ошибемся лишь в малом количестве случаев, если $M$ достаточно велико.

Теперь мы определим верхнюю границу $U$ числа шагов, требующихся для достижения любого блока данных из любого другого блока данных, после добавления блоков данных второго поколения. Рассмотрим наихудший случай, когда и исходный, и целевой блоки данных принадлежат второму поколению. Мы принимаем во внимание только связи между соседними блоками данных первого поколения, соседними блоками данных второго поколения и соседними блоками данных первого и второго поколения, которые входят в подмножество всех связей в структуре, что еще больше увеличивает $U$. Связи между соседними блоками данных первого поколения с точки зрения метрики длиннее, чем связи между соседними блоками данных второго поколения.

Следовательно, первые связи предпочтительнее, чем последние, для достижения искомого блока данных за минимальное число шагов. Для достижения ближайшего блока данных первого поколения из исходного блока данных потребуется $M$ шагов, для достижения блока данных первого поколения, ближайшего к искомому блоку данных, потребуется самое большее $P$ шагов, и для достижения целевого блока данных из ближайшего блока данных первого поколения потребуется в худшем случае $M$ шагов.

Для достижения любого блока данных из любого другого блока данных после добавления блоков данных второго поколения потребуется самое большее $P+2M$ шагов. Таким образом, мы показали, что, когда размеры структуры удваиваются, число шагов алгоритма поиска увеличивается лишь на константу $2M$, а это означает, что мы имеем сложность $O(\log n)$, где $n$ – число блоков данных в структуре.

## 5. Моделирование

Мы использовали структуру *MSW* с одним атрибутом. Моделирование было проведено для различного числа блоков данных в структуре и для различных значений параметра $m$ в алгоритме добавления.

На *рис. 2* показана средняя длина пути (числа шагов), совершенная алгоритмом поиска для достижения блока данных, ближайшего к запросу. с помощью алгоритма *Insertion_with_Order* были сгенерированы сети размером от 1000 до 1000000 элементов для множества ключей являющимися случайными действительными числами из пяти различных распределений: для равномерного в сегменте [0,1], степенного с показателем 0,5; нормального со средним значением 0,0 и стандартным отклонением 1,0; нормально логарифмического со средним значением 4,0 и стандартным отклонением; и точками, распределенными нормально с отклонением 1,0 в зоне 10 центров {-10,0; -7,0; -5,0; 0,0; 1,0; 2,0; 10,0}. Отклонения длины пути от среднего значения для различных распределений показаны на *рис. 3*.

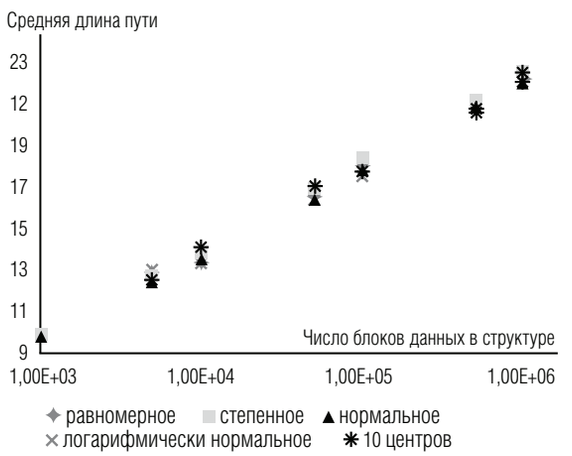


*Рис. 2.* Средняя длина пути для различных распределений

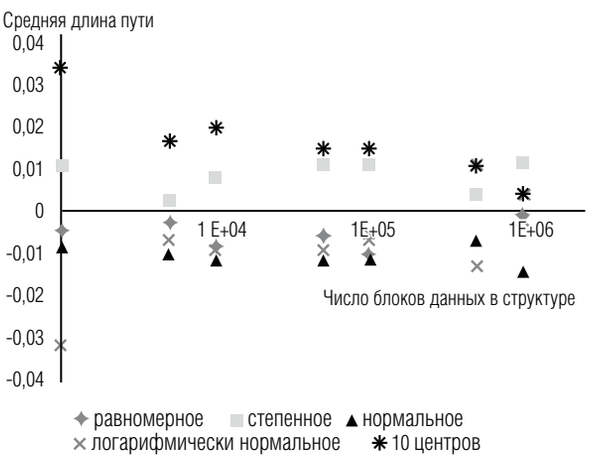


*Рис. 3.* Отклонение среднего пути для различных распределений

Как можно видеть из *Рис. 2* и *Рис. 3*, структура демонстрирует четкую логарифмическую зависимость от числа элементов с очень небольшим отклонением для всех типов распределений. Независимость от типа распределения было ожидаемо, поскольку все алгоритмы структуры *MSW* не используют напрямую значения расстояний между блоками данных и основаны на относительном порядке. Поэтому все остальные эксперименты были проведены на множествах блоков данных, образы которых были равномерно распределены на интервале (0,1).

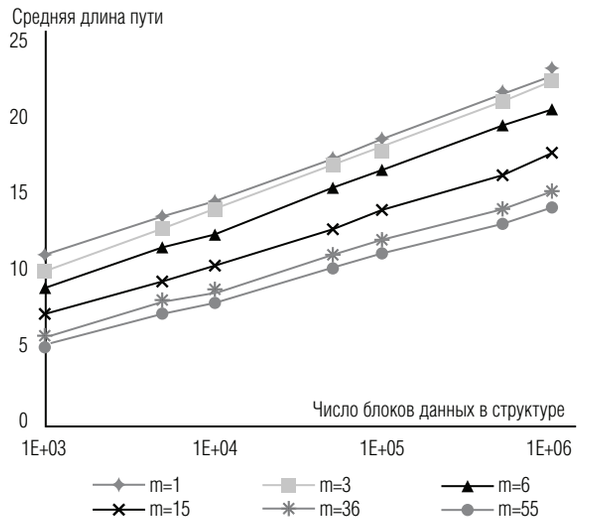


*Рис. 4.* Средняя длина алгоритма поиска

На *рис. 4* и *5* показана зависимость длины пути от параметра *m*. На *рис. 4* представлены результаты для структуры различных размеров, а на *рис. 5* – для шести различных фиксированных размеров: 1 мил-

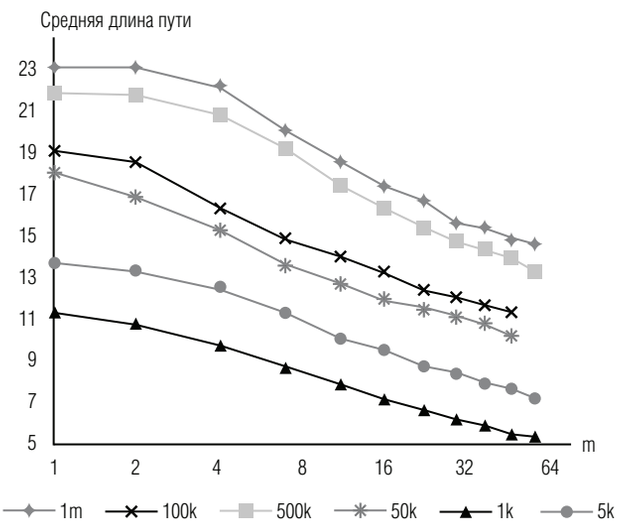


*Рис. 5.* Зависимость длины пути от параметра m для фиксированных размеров структуры – 1k-1m

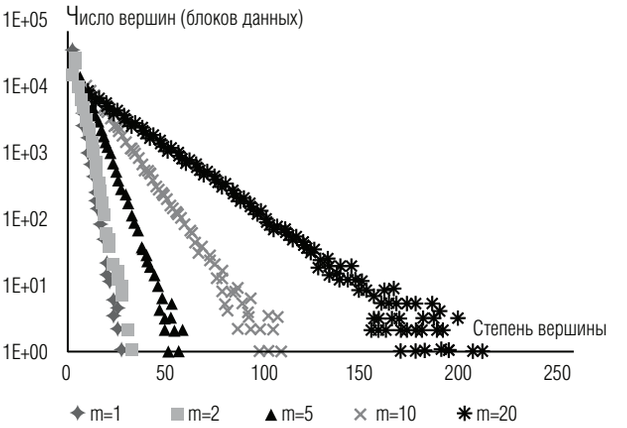


*Рис. 6.* Распределение степеней вершин

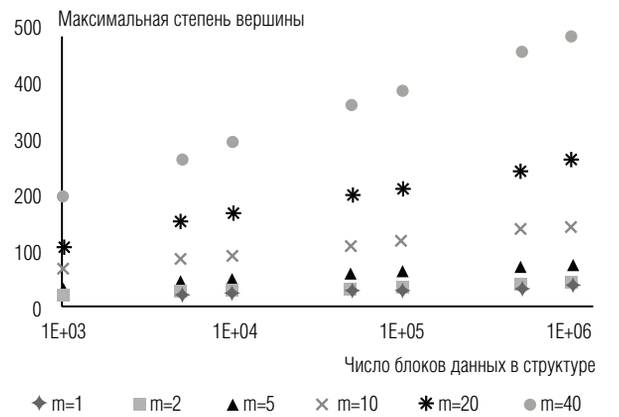


*Рис. 7.* Максимальная степень вершины

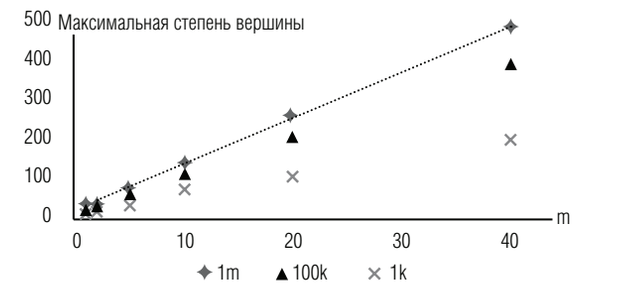


*Рис. 8.* Зависимость максимальной степени вершины от параметра *m* для 1 млн, 100 тыс. и 1 тыс. блоков данных в структуре

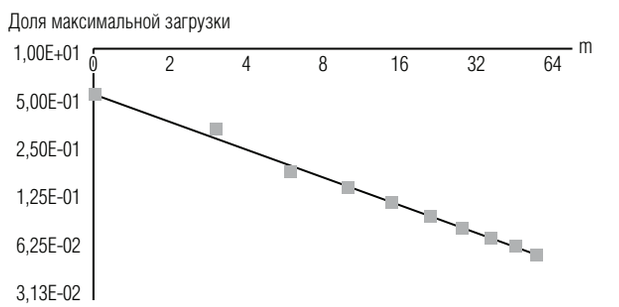


*Рис. 9.* Зависимость доли максимальной загрузки от параметра *m* для 100 тыс. блоков данных в структуре

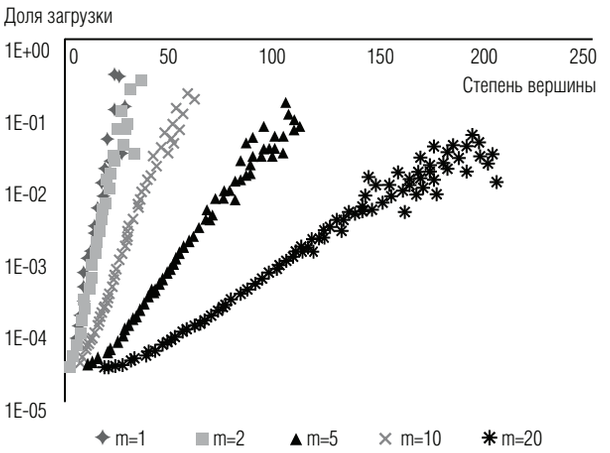


*Рис. 10.* Зависимость загрузки блоков данных от числа связей для 100 тыс. блоков данных в структуре

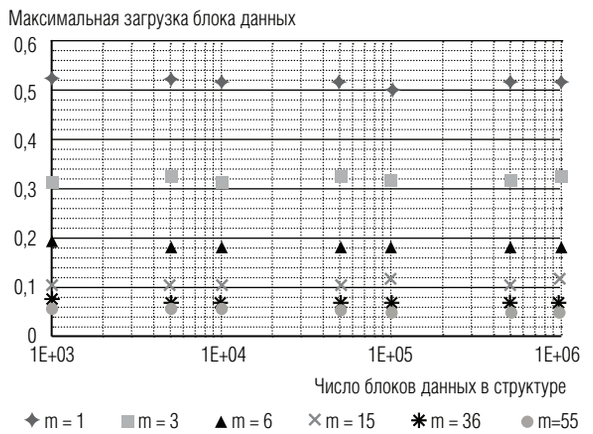


*Рис. 11.* Максимальная загрузка блока данных

лион, 500 тысяч, 100 тысяч, 50 тысяч, 5 тысяч и 1 тысяча блоков данных. Как можно видеть из *рис. 5*, зависимость средней длины пути от переменной *m* близка к обратному логарифмическому закону.

Для того чтобы определить, как распределяется нагрузка на узлы сети, представленные в качестве блоков данных, во время поиска, мы измерили, насколько часто алгоритм поиска посещает конкретный блок данных. Степень загруженности вычислялась, как отношение числа поисков, в которых конкретный блок данных был вовлечен, к общему числу поисков. Как можно видеть из *рис. 10* и *11*, максимальное количество загрузок зависит от размера структуры, и доля загрузок конкретного блока данных экспоненциально зависит от степени вершины (числа связей с другими блоками данных.

Зависимость доли максимальной загрузки от параметра *m* представлена на *рис. 9*. Из него не трудно видеть, что дополнительные связи увеличивают разнообразие путей поиска, тем самым снижая нагрузку на блоки данных с более значительным количеством связей, что помогает избежать проблем узкого места. Исследование степеней вершин представлено на *рис. 6*, *7* и *8*. Легко заметить, что максимальная степень вершины имеет логарифмическую зависимость от размера структуры (*рис. 8*) и линейную зависимость от параметра *m*. На *рис. 6* показано, что распределение степеней вершины следует экспоненциальному закону.

Кроме того, мы исследовали свойство стабильности структуры, т.е. способность сохранять работоспособность при выпадении узлов. На *рис. 12* показана доля числа узлов (в процентах), которые должны быть удалены (случайным образом с равномерным распределением) для разделения сети на две компоненты связности. Как видно, доля удаленных блоков данных имеет степенную зависимость от общего числа блоков данных в структуре. Значение степени в этом степенном законе также подчиняется степенному закону, причем показатель найден эмпирическим путем равным -1,24 (*см. рис. 13*).

Так, мы имеет зависимость $\frac{r}{n} \sim n^{c \cdot m^{-1.24}}$, где $C$ – некоторая константа ~ 1,7 и $r$ – ожидаемое число удаленных блоков данных, для которых сеть распадается.

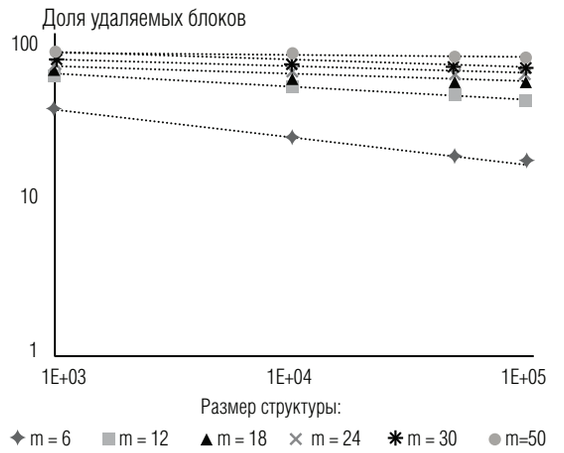


*Рис. 12.* Средняя процентная доля числа случайным образом удаляемых узлов с равномерным распределением для разделения сети на две компоненты связности

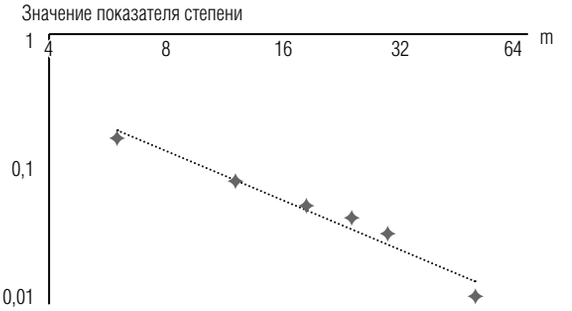


*Рис. 13.* Значение показателя степени в законе зависимости числа удаленных блоков данных от размера структуры

## Заключение

В данной статье мы представили вариант структуры данных «Метризованного тесного мира», который позволяет осуществлять точный поиск блоков данных. Мы описали алгоритмы добавления данных и поиска данных и продемонстрировали, что структура имеет вычислительную сложностью поиска $O\left(\frac{\log n}{m^{\alpha}}\right)$, где найденный эмпирическим путем показатель равен $\alpha \sim 0.11$.

Мы представили экспериментальные данные, которые демонстрируют свойства расширяемости и децентрализации структуры. Наш результат может быть кратко описан следующим образом:

✦ Логическая сеть на уровне блоков данных имеет сложность поиска $O\left(\frac{\log n}{m^{\alpha}}\right)$ при использовании простого «жадного» алгоритма поиска.

✦ Структура не зависит от распределения данных.

✦ Инкрементный алгоритм добавления нового блока данных изменяет только небольшое количество блоков данных.

✦ Короткие пути для «жадного» алгоритма поиска формируются без поддержания явным образом какого-либо распределения длин связей.

✦ Физическое размещение блоков данных не зависит от содержимого данных и функциональности поиска.

✦ Динамическая вставка новых физических сетевых узлов не требует в перемещении блоков данных из существующих узлов.

## Благодарности

Работа выполнена в Национальном исследовательском университете «Высшая школа экономики» при поддержке гранта *RSF* 14-41-00039. ■

## Литература

1. Albert R., Barabasi A.L. Statistical mechanics of complex networks // Reviews of Modern Physics. 2002. Vol. 74. No. 1. P. 47–97.
2. Amaral L.A.N., Scala A., Barthelemy M., Stanley H.E. Classes of small-world networks // Proceedings of the National Academy of Sciences (USA). 2000. Vol. 97 (21). P. 11149–11152.
3. Looking up data in P2P systems / H.Balakrishnan [et al.] // Communications of the ACM. 2003. Vol. 46. No. 2. P. 43–48.
4. Bondi A.B. Characteristics of scalability and their impact on performance // Proceedings of the 2nd International Workshop on Software and Performance (WOSP 2000), September 17–20, 2000, Ottawa, Canada. 2000. P. 195–203.
5. Cattell R. Scalable SQL and NoSQL data stores // ACM SIGMOD Record. 2011. Vol. 39 (4). P. 12–27.
6. Dean J., Ghemawat S. Mapreduce: Simplified data processing on large clusters // Communications of the ACM. 2008. Vol. 51. No. 1. P. 107–113.
7. Harizopoulos S., Abadi D.J., Madden S., Stonebraker M. OLTP through the looking glass, and what we found there // Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data (SIGMOD 2008), June 9–12, 2008, Vancouver, Canada. 2008. P. 981–992.
8. Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the World Wide Web / D.Karger [et al.] // Proceedings of the 29th Annual ACM Symposium on Theory of Computing, ACM, May 4–6, 1997, El Paso, Texas, USA. 1997. P. 654–663.
9. Krylov V., Logvinov A., Ponomarenko A., Ponomarev D. Active database architecture for XML documents // Proceedings of the 21st International Conference on Computer Applications in Industry and Engineering (CAINE 2008), November 12–14, 2008, Honolulu, Hawaii, USA. 2008. P. 244–249.
10. Krylov V., Logvinov A., Ponomarenko A., Ponomarev D. Metrized small world properties data structure // Proceedings of the 17th International Conference on Software Engineering and Data Engineering (SEDE 2008), June 30 – July 2, 2008, Los Angeles, California, USA. 2008. P. 203–208.
11. Logvinov A., Ponomarenko A., Krylov V., Malkov Y. Metrized small world approach for nearest neighbor search // Proceedings of the 2010 Spring/Summer Young Researchers' Colloquium on Software Engineering (SYRCoSE 2010), June 1–2, 2010, Nizhny Novgorod, Russia. 2020. Vol. 4. P. 151–156.
12. Malkov Y., Ponomarenko A., Logvinov A., Krylov V. Approximate nearest neighbor algorithm based on navigable small world graphs // Information Systems. 2014. No. 45. P. 61–68.
13. Malkov Y., Ponomarenko A., Logvinov A., Krylov V. Scalable distributed algorithm for approximate nearest neighbor search problem in high dimensional general metric spaces // Lecture Notes in Computer Sciene –

Similarity Search and Applications. 2012. Vol. 7404. P. 132–147.
14. Maymounkov P., Mazieres D. Kademlia: A peer-to-peer information system based on the XOR metric // Peer-to-Peer Systems. 2002. Vol. 2429. P. 53–65.
15. Solving big data challenges for enterprise application performance management / T.Rabl [et al.] // Proceedings of the VLDB Endowment. 2012. Vol. 5. No. 12. P. 1724–1735.
16. Chord: A scalable peer-to-peer lookup service for internet applications / I.Stoica [et al.] // ACM SIGCOMM Computer Communication Review. 2001. Vol. 31. No. 4. P. 149–160.
17. Tarkoma S. Overlay networks: Toward information networking. Boca Raton, FL: CRC Press, 2010. 237 p.
18. Opportunistic use of content addressable storage for distributed file systems / N.Tolia [et al.] // Proceedings of the 2003 USENIX Annual Technical Conference (USENIX 2003), General Track, June 9–14, 2003, San Antonio, Texas, USA. 2003. Vol. 3. P. 127–140.
19. Watts D.J. Small worlds: The dynamic of networks between ordered and randomness. New Jersey: Princeton University Press, 1999. 168 p.