УДК 004.75

А.В. Сергеев

Технологии поиска данных в одноранговых сетях, построенных на основе распределенных хеш-таблиц

Проведен анализ архитектуры одноранговых сетей, построенных на основе распределенных хэш-таблиц, а также введены метрики для сравнения алгоритмов поиска с целью определения оптимального.

Ключевые слова: децентрализованные сети, одноранговые сети, распределенная хеш-таблица.

Здійснено аналіз архітектури одноранговых мереж, побудованих на основі розподілених хеш-таблиць, та введено метрики для порівняння алгоритмів пошуку з метою визначення оптимального.

Ключові слова: децентралізовані мережі, однорангові мережі, розподілена хеш-таблця.

Введение. Интенсивное увеличение объемов интернет-трафика и количества пользователей как в глобальных, так и в локальных сетях привело к выявлению недостатков классической клиент-серверной архитектуры, основной причиной которых есть *узкое место* (*bottleneck*) таких сетей – наличие единственного центрального сервера. Это приводит к вопросам, связанным со слабой масштабируемостью (способностью системы справляться с увеличением нагрузки путем добавления новых элементов) и отказоустойчивостью (способностью сохранять работоспособность при отказе одного или нескольких компонентов сети). Решением этих проблем есть использование децентрализованных сетей, функционирование которых не требует привязки к одному постоянному серверу.

Исходя из того, что каждый компьютер может быть как сервером, так и клиентом, использование такой модели приводит к тому, что любой новый элемент сети может быть присоединен без перестройки существующей архитектуры. Отсутствие единого центрального элемента значительно повышает надежность сети путем того, что при отказе ее части сохраняется работоспособность системы до тех пор, пока функционирует хотя бы один элемент.

Децентрализованные сети широко применяются в файлообменных (*mTorrent*, *Gnutella*, *EDonkey*), мультимедийных (*SopCast*, *AceStream*), финансовых (*Bitcoin*, *Litecoin*) и других сервисах. Теоретические подходы построения децентрализованных сетей и алгоритмы организации поиска в них рассматривались в работах *Alireza Naghizadeh* (2014), *Tahereh Yourdkhani* (2014), *Ciprian Dobre* (2011), *Florin Pop* (2011), *Moni Naor* (2003), *Udi Wieder* (2003), *Gennadiy Porev* (2010) и других.

Постановка проблемы

Основной практической задачей в децентрализованных сетях есть поиск информации пользователем. При этом необходимо обеспечивать выполнение двух основных требований: релевантности, т.е. меры соответствия результатов поиска ожиданиям пользователя, и оптимизации затрат ресурсов. Таким образом, появляется проблема выбора алгоритма поиска, соответствующего необходимым критериям эффективности. Учитывая их дифференциацию в зависимости от выполняемых задач, необходимо выделить ряд характеристик для проведения сравнительного анализа. Цель этой статьи – исследование признаков эффективности децентрализованных сетей и определение алгоритмов, оптимальных для организации поиска по набору критериев.

Для выполнения поставленной задачи необходимо определить основные характеристики децентрализованных сетей, влияющих на эффективный поиск данных и организации обмена ими.

Применение распределенных хеш-таблиц для построения одноранговых сетей

Для децентрализованных (одноранговых – *P2P*) сетей характерно то, что каждый узел равноправный по отношению к другим. В такой сети любой элемент есть как клиентом, так и сервером. К преимуществам такой системы можно отнести:

• масштабируемость – независимость от количества узлов в сети;

• отказоустойчивость – сеть продолжит работать при отказе отдельных узлов, что есть прямым следствием отсутствия центрального сервера.

Кроме приведенных примеров в одноранговых системах, можно отнести одну из популярных систем мгновенной отправки сообщений *Skype* версии 2012 года.

Для повышения эффективности *P*2*P* можно использовать распределенные хэш-таблицы [1] (*Distributed Hash Table – DHT*). Хеш-таблица – это структура данных, которая реализует интерфейс ассоциативного массива и позволяет хранить пары ключ–значение и выполнять следующие операции: добавление новой пары к массиву, поиск элемента по ключу и удаления элемента из массива. Преимущество такой структуры данных то, что каждая операция выполняется за время не более чем $O(N)$, где N-размер хеш-таблицы. Распределенная хеш-таблица (*DHT*) представляет собой класс систем, реализующих сервис, и работает подобно хеш-таблице. Таким образом, каждый узел в сети может рационально искать значение, ассоциированное с данным ключом. Обычно ключом выступает некоторая хэш-функция, т.е. функция, преобразующая массив произвольной длины в битовую последовательность фиксированной длины по определенному алгоритму, зависящая от данных. Наиболее известная реализация *DHT – BitTorrent*.

Каждый элемент сети имеет свой идентификатор в некотором бинарном ключевом пространстве, определяемый случайным образом. Обычно его длина составляет 128 или 160 бит.

Хранение файлов в сети происходит следующим образом. Сначала каждому элементу сети ставится в соответствие некоторый уникальный бинарный идентификатор. Каждый файл, который распределяется, также имеет свой идентификатор, причем он есть значением хеш-функции от названия файла. Обычно, для хеширования используется алгоритм *SHA*1, который заключается в функции сжатия. На вход этой функции подается блок текста длиной в 512 байт, а также результат прошлого шага алгоритма. На выходе получаются значения всех хеш-блоков до этого момента. Искомой хеш-функцией есть исходное значение последнего блока.

С учетом организации аппаратной структуры *P*2*P* эффективность ее функционирования определяется взаимным расположением элементов по определенной заранее некой метрикой *близости*. Таким образом, оптимизация ее работы может быть достигнута путем применения различных алгоритмов поиска. На сегодня существует несколько систем с индивидуальными алгоритмами поиска, использование которых обоснованно теоретически или практически, поэтому их характеристики будут составлять базу этого исследования.

Анализ характеристик одноранговых систем, созданных на основе *DHT*

•*DistHash*

Оверлейная сеть – иерархическая структура, состоящая из трех уровней [1]. На первом находятся так называемые агенты (*agents*), которые связаны с *RAgents* (хранят не только локальные, но и метаданные), на втором уровне и вместе со своими агентами, в свою очередь, объединяются в кластер на высшем уровне. *RAgents* – (супер-агенты, супер-узлы) выбираются из множества агентов случайным образом, осуществляя управление определенным их набором Модальность сбора элементов в кластер определяется пространственным положением каждого элемента и набором определенных метрик.

Топология сети базируется на предположении, что уже существует несколько кластеров, из которых новый агент может получить информацию (сетевой адрес и порт подключения) о списке существующих *RAgents*. После этого элемент пытается найти *ближайшего RAgent*, к которому он может подключиться. В этом случае таким *RAgent* будет тот, который выбран не только по критериям сети и пространственному положению, но и с учетом количества агентов в нескольких ближайших *RAgents*. Агент подсоединяется к ближайшему *RAgent*, у которого будет наименьшее количество уже присоединенных агентов.

Необходимо оптимально распределить количество агентов и *RAgents*. При слишком большом количестве агентов, в сравнении с количеством *RAgents*, каталог метаданных станет слишком большим и, соответственно, операции на нем будут занимать больше времени. С другой стороны, если количество агентов мало, то количество запросов к системе с целью доступа к данным будет выше. Для решения этой проблемы количество *RAgents* увеличивается и уменьшается в соответствии с ростом или уменьшением количества агентов. Данное распределение происходит на уровне кластера. Когда количество агентов, присоединенных к *RAgent*, превышает некоторое лимит, новый *RAgent* выбирается из списка агентов с помощью алгоритма голосования [3] и кластер разделяется на два. Оставшиеся агенты распределяются между двумя *RAgents* (вместе с метаданными). В противном случае (малое количество агентов) кластер уничтожается при слиянии его агентов с агентами другого кластера.

Каждый агент обслуживает копии нескольких объектов данных. Для согласования между копиями данных, каждый объект имеет *владельца* – агента, который имеет право его изменять. Каталог метаданных имеет структуру хеш-таблицы, т.е. представляет собой ассоциативную структуру, где каждому ключу ставится в соответствие какое-то значение (данные, в нашем случае). Значение имеет структуру связного списка. Каждый объект имеет уникальный идентификатор, каждое значение имеет ссылки на связный список идентификаторов агентов, причем первый в списке идентификатор принадлежит *хозяину* объекта. Внутри копии объекты также хранятся в виде хеш-таблицы, где ключом есть идентификатор, а значением объект. Чтобы найти соответствующую информацию, запрос поступит сначала в *RAgent* и только потом к агенту, у которого находится необходимый объект.

Алгоритм поиска данных состоит из нескольких шагов. Сначала *RAgent*, с которым соединен агент, что ищет данные (клиент), направляет запрос с определенным критерием поиска. После этого каждый *RAgent* предоставляет ответ со списком объектов, удовлетворяющих условиям поиска. На следующем этапе изначальный *RAgent* соединяет полученные списки со своим списком таким образом, чтобы в конечном списке не было объектов с одинаковыми идентификаторами. После этого полученный список отправляется к клиенту.

Сложность алгоритма определяется по формуле:

$$S = RM(4P + LogL),$$

где R – количество *RAgents*, M – количество ключей в каталоге метаданных, P – количество объектов, подходящих под критерии поиска, L – количество объектов, обслуживаемых агентом.

- ***Chord***

Каждый элемент системы хранит таблицу, в которой находятся *IP* адреса элементов, находящихся на «расстоянии» 2^n, где n – натуральное от его *ID* [4]. При попытке найти данные с ключом k элемент направляет запрос к элементу со своей таблицы, у которого *ID* наибольший, но не превышает k. Построение таблицы на основе степени двойки обеспечивает то, что элемент может всегда направить запрос по крайней мере половине от оставшихся *ID*, близких k, При этом количество запросов стремится к $O(LogN)$, где N – количество элементов сети.

Основной акцент в построении системы ставится на надежность и корректность поиска. Алгоритм обеспечивает корректный поиск, используя список потомков: каждый элемент сохраняет *IP* – адрес некоторого количества последующих элементов сразу после того как запрос попал в их ключевое пространство. Это позволяет запросу инкрементально двигаться по пространству ключей, даже если многие элементы из таблицы уже исчезли или вышли из строя. Единственным случаем, когда алгоритм не может гарантированно найти потомка – это тот, в котором все потомки исчезнут одновременно, до того как элемент успеет обновить свой список.

Добавление элемента в систему начинается с запроса этого элемента к некоторому элементу сети с просьбой вычислить его *ID*. Это нужно для того, чтобы новый элемент корректно

присоединился к системе и позволил родительским элементам обновить их списки потомков. Параллельно с этим, все элементы системы обновляют свои таблицы на фоновом уровне. Новый элемент должен узнать: будут ли данные с ним ассоциированы, а связь потомков обеспечит то, что ключи могут быть взяты у потомков нового элемента.

- ***CAN***

Для реализации *DHT* система использует многомерную (d – количество измерений) декартову систему координат [5]. Пространство разбивается на d – мерные прямоугольники, называемые *зонами.* Каждый элемент системы отвечает за определенную зону и, наоборот, каждый участник однозначно определяется в пределах своей зоны. Ключ привязан к точке пространства и хранится у элемента, который обслуживает зону, где находится эта точка. Каждый элемент сохраняет таблицу маршрутизации с координатами всех соседей элемента. Элементы считаются соседними, если их зоны разделяют $d-1$ мерную гиперплоскость.

Операция поиска реализована пересылкой запроса по пути, который представляет собой наиболее близкую к прямой линии траекторию в координатном пространстве, связывающей искателя с элементом, хранящим искомый ключ. После получения запроса, элемент отправляет его участнику, ближайшему к элементу, хранящему ключ, и разрывает с ним связь. Каждый элемент обслуживает $O(d)$ состояний, сложность поиска составляет $O(dN^{\frac{1}{d}})$, что близко к $O(N^{\frac{1}{d}})$, так как обычно $d << N$ [6].

Чтобы присоединиться к сети, новый элемент сначала выбирает случайную точку P в координатном пространстве и просит уже существующий элемент найти участника n, который отвечает за зону, где находится эта точка. Элемент n разбивает свою зону на две и перекладывает ответственность за одну из полученных зон на нового участника. Новый элемент может легко инициализировать свою таблицу маршрутизации, так как все его соседи, кроме самого элемента n, есть и соседями n. После присоединения, элемент представляет информацию о себе своим соседям, для того чтобы они обновили свои таблицы. При выходе элемента из сети его зона передается одному из его соседей.

- ***Pastry***

Данная система подробно описана в работе [7]. Каждый элемент сети n поддерживает некоторый набор из L (L-набор) элементов системы, который состоит из $\frac{|L|}{2}$, близких к n, но не превышающих его и $\frac{|L|}{2}$ близких, однако меньших, чем n элементов. Корректность всей системы зависит только от корректности данного набора.

Для оптимизации алгоритма поиска система поддерживает таблицу маршрутизации указателей к другим элементам, распределенным в ключевом пространстве. Процесс поиска выглядит следующим образом. Если искомый ключ покрывается L-набором n, то запрос пересылается к нему. Как правило, это не выполняется до тех пор, пока запрос не войдет в зону ключевого пространства, достаточно близкую к ключу. В этом случае запрос приходит к элементу из таблицы маршрутизации, у которого общий с ключом бинарный префикс длиннее, чем у n. В случае когда такой элемента найти не удается, запрос переходит к элементу, у которого общий префикс с ключом не короче, чем у n, а ID арифметически ближе к ключу [5]. Этот элемент точно находится в L-наборе n, кроме случая, когда запрос уже дошел до элемента, у которого ID арифметически ближайший к ключу. Если таблица маршрутизации корректна, то сложность алгоритма поиска будет рассчитываться, как $Log2^b N$, где N– количество участников в сети, b – алгоритмический параметр, который обычно равен четырем.

При присоединении к сети новый элемент должен создать свою таблицу маршрутизации, а также уведомить других участников о своем существовании. Предполагается, что новому элементу сначала известно о ближайшем элементе в системе, согласно метрике, которая уже есть ее частью. Такой элемент может быть определен автоматически, например использованием *IP*-

мультикаст, или может быть получен администратором сети с помощью внешних каналов. Выход элементов из системы может происходить без предупреждения. Для замены участника его сосед в ключевом пространстве связывается с элементом, у которого самый большой *ID* со стороны исчезнувшего элемента, и запрашивает его *L*-набор. Этот набор частично перекрывает набор текущего элемента, но в нем также хранятся участники, там не представленные. Среди этих элементов выбирается один (предварительно убедившись, что он еще в сети) и присоединяется к *L*-набору. Таким образом, процедура гарантирует, что каждый элемент может восстановить свой *L*-набор, кроме случая, когда одновременно исчезли из сети не менее $\frac{|L|}{2}$ элементов со смежными *ID*. Но благодаря разнообразию (географической, ресурсной и т.д.) этих элементов такая вероятность достаточно низкая, даже для небольших значений *L*. Данная система применяется в таких сервисах как *Past* и *Scribe*.

- ***Kademlia***

Алгоритм в данной системе [8] состоит в использовании рекурсии, с помощью которой элемент находит необходимое значение в сети за приемлемое время и с задействованием рационального количества запросов к другим элементам сети. Алгоритм базируется на расчете *расстояния* между узлами в сети, который определяется как результат операции *XOR* между уникальными 160-битными идентификаторами элементов сети. Эффективность алгоритма заключается в том, что каждый элемент сети должен обмениваться информацией только с $O(LogN)$ участников, где *N* – общее количество элементов сети, т.е. при изменении их количества не нужно будет перестраивать всю сеть.

Преимущества алгоритма – следствие использования именно упомянутой метрики. *XOR* есть симметричной операцией и позволяет участникам сети получать запросы от элемента, который находится у них в таблице маршрутизации. В отличие от других алгоритмов, это позволяет получать полезную маршрутную информацию сразу от запросов. Более того, асимметрия приводит к усложнению изменения таблицы маршрутизации.

В алгоритме предполагается, что элементы составляют листья бинарного дерева, в котором позиция каждого элемента определяется как кратчайший уникальный префикс его идентификатора. Для каждого элемента происходит разбиение бинарного дерева на набор последовательных низших поддеревьев, которые не включают в себя данный элемент. Высшее поддерево состоит из половины бинарного дерева, в котором нет элемента. Следующее поддерево состоит из половины остатка дерева, не включая элемент и т.д.

Алгоритм поиска представлено следующей процедурой. После нахождения хэш-функции от файла, проводится операция их распределения по следующему алгоритму: рассчитывается операция *XOR* между идентификатором каждого элемента сети и хэш-функцией каждого файла. Соответствующий файл будет привязано к тому элементу, для которого результат вышеупомянутой функции будет минимальным. Таким образом, алгоритм поиска файла в сети сводится к нахождению элемента, к которому он привязан. Поиск файла произвольным элементом сети представляет собой рекурсивную процедуру, которая базируется на расчете вышеупомянутой метрики, между ключом файла (который известен) и идентификаторами соседних к элементу участников. Поиск продолжается до тех пор, пока либо информация не будет найдена, либо не закончатся участники сети. Сложность алгоритма является логарифмической, т.е. увеличение количества элементов сети вдвое приводит к увеличению шагов алгоритма на один.

Эффективность этого алгоритма проверено эмпирически через использование в таких системах как *eMule*, *I2P*, *RevConnect* и др.

Определение оптимального алгоритма поиска

Для определения оптимального алгоритма поиска необходимо провести сравнительный анализ по ряду критериев, т.е. решить многокритериальную задачу. Она может быть формализована следующим образом:

$$Z = f(x_1, ..., x_n) \to \min,$$

где $x_1, \ldots, x_n$ – характеристики алгоритма.

Очевидно, что совокупность критериев не может быть полной, но можно выделить группу критических признаков. Прежде всего, это количество шагов алгоритма, т.е. количество запросов, необходимых для нахождения нужной информации; расстояние между элементами, т.е. некоторая численная метрика, определяющая относительное расположение узлов в сети; практическая реализация системы. Исходя из этих метрик и будем оценивать эффективность алгоритмов.

Для сравнения сложности различных алгоритмов, обоснуем отношение предпочтения. Так, очевидно, что логарифмическая функция $O(LogN)$ описывает меньшее количество шагов, чем полиномиальная $O(N^{\frac{1}{d}})$. Функция $RM(4P + LogL)$ зависит от количества объектов данных (L) так, если RM относительно мало, то сложность алгоритма описывается логарифмической зависимостью. В противном случае следует учитывать все компоненты. Таким образом, эта функция может описывать широкий диапазон сложности. Примем, что

$$RM(4P + LogL) \prec O(LogN).$$

Таблица 1. Сравнительная характеристика алгоритмов по основным критериям

	Количество шагов алгоритма	Операция нахождения *расстояния* между элементами	Практическая реализация
DistHash	$RM(4P + LogL)$	Арифметическая разность	–
Chord	$O(LogN)$	Арифметическая разность	–
CAN	$O(N^{\frac{1}{d}})$	Арифметическая разность	–
Pastry	$O(LogN)$	Длина одинаковой последовательности префиксных битов	+
Kademlia	$O(LogN)$	*XOR*	+

Таким образом, отношение предпочтения выглядит следующим образом:

$$RM(4P + LogL) \prec O(N^{\frac{1}{d}}) \prec O(LogN).$$

Для введения количественных оценок проранжируем значения критериев для алгоритмов, исходя из того, что:

1. $x_1 \to \min$, где x_1 – количество шагов алгоритма, т.е. чем меньше шагов будет задействовано, тем быстрее будет найдена информация.

2. x_2 x_2 – операция нахождения расстояния между элементами. Этот критерий субъективен, так как только в алгоритме *Kademlia* процедура симметрична, что дает некоторую информацию о расстоянии. Поэтому предположим, что вид отношение предпочтения:

$$x_{2(DistHash)} \approx x_{2(Chord)} \approx x_{2(CAN)} \approx x_{2(Pastry)} \prec x_{2(Kademlia)}.$$

3. x_3 – практическая реализация алгоритма, результат представлен булевым значением: 1 – в случае существующий реализации и 2 – в случае отсутствующей.

Так как функция $Z \to \min$, то чем меньше значение метрики параметра, тем лучше. Нужно принимать во внимание тот факт, что критерии не равнозначны и поэтому необходимо проранжировать параметры по важности (метод простого ранжирования [9]). Эти значения будут коэффициентами (весами) при оценке критериев.

Таблица 2. Ранжирование критериев по важности

x_1	x_2	x_3
3	2	1

Дадим оценку каждого критерия для каждой системы:

Таблица 3. Сравнение алгоритмов по каждому из критериев

	x_1	x_2	x_3
DistHash	3	2	2
Chord	1	2	2
CAN	2	2	2
Pastry	1	2	1
Kademlia	1	1	1

Теперь можно оценить эффективность каждого алгоритма по формуле:

$$Z = \sum_{i=1}^{3} x_i k_i,$$

где x_i – определенный критерий, а k_i – соответствующий показатель веса.

После расчета функции для каждой системы получаем: $Z_{DistHash} = 15$; $Z_{Chord} = 9$; $Z_{CAN} = 12$; $Z_{Pastry} = 8$; $Z_{Kademlia} = 6$;. Таким образом, функция Z достигает минимума для алгоритма *Kademlia*, который и есть оптимальным при сравнении по этим критериям.

Заключение. Рассмотрено понятие децентрализованных сетей, выделены преимущества таких сетей перед клиент-серверной архитектурой. Сделан обзор и проведено сравнение систем и

алгоритмов поиска данных в этих системах, созданных на основе распределенных хеш-таблиц.

Описан ряд систем, построенных на основе распределенных хеш-таблиц, для которых разработаны алгоритмы поиска данных. Следует отметить, что все рассмотренные алгоритмы достаточно эффективны, но при этом *Chord*, *Pastry* и *Kademlia* имеют минимальную логарифмическую сложность. Преимуществом *Kademlia* есть симметричная метрика *XOR* для нахождения *расстояния* между узлами сети, которая значительно упрощает построение таблицы маршрутизации и способна предоставлять некоторую информацию, что выгодно отличает данную метрику от метрик, применяемых в других системах.

Таким образом, несмотря на то что для решения большинства существующих проблем клиент-серверной архитектуры можно использовать любую из систем, проанализированных в статье, сравнение по нескольким критериям указывает на то, что оптимальной системой является *Kademlia*.

1. *Порєв Г.В.* Методи та засоби побудови інформаційних технологій на основі територіально розосереджених сервіс-орієнтованих однорангових мереж: Дис. … д-ра техн. наук. – К., 2013. – 397 с.
2. *Florin Pop*, *Valentin Cristea.* DistHash: A robust P2P DHT-based system for replicated objects Ciprian Dobre. – Bucharest, Romania, May 26–29, 2009. – **1**. – P. 453-460.
3. *Hardekopf B.*, *Kwiat K.*, *Upadhyaya S.* A Decentralized Voting Algorithm for Increasing Dependability in Distributed Systems. – 5th World MultiConference on Systemic // Cybernetics and Informatics (SCI2001), 2001. – P. 3–5.
4. *Chord*: A scalable peer-to-peer lookup protocol for internet applications / I. Stoica, R. Morris, D. Karger et al. // IEEE/ACM Transactions on Networking. – 2003. – **11**, N 1. – P. 17–32.
5. *A scalable* content-addressable network / S. Ratnasamy, P. Francis, M. Handley et al. // Proc. of ACM SIGCOMM. – San Diego, CA (Aug. 2001). P. 162–168.
6. *Looking* up data in p2p systems / H. Balakrishnan, M. Kaashoek, D. Karger et al. // Comm. ACM 46,2 (Feb. 2003).
7. *Rowstron A.*, *Druschel P.* Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems // Proc. of the 18th IFIP/ACM Int'l Conf. on Distributed Systems Platforms (Nov. 2001). – P. 3–13.
8. *Maymounkov P.*, *Mazieres D.* Kademlia: A Peer-to-Peer Information System Based on the XOR Metric // IPTPS 2002, 7-8 March 2002. – P. 2–9.
9. *Гнатієнко Г.М.*, *Снитюк В.Є.* Експертні технології прийняття рішень. – К.: Маклаут, 2008. – 444 с.

Поступила 14.12.2015

E-mail: *a.serhieiev@gmail.com*

UDC 004.75

A.V. Serhieiev

The Technologies of the Data Retrieval in Peer-to-Peer Networks Constructed on the Basis of the Distributed Hash Tables

Keywords: decentralized networks, ad hoc networks, distributed hash table.

Introduction. The necessity of the large data sets processing, increasing the scalability and fault tolerance lead to the development of peer-to-peer networks, that require, unlike client-server architecture, the use of complex algorithms for data search. Thus, it is necessary to determine a peer-to-peer network with the optimal search algorithm according to the certain criteria.

Purpose. The purpose of this paper is to analyze the constructed peer-to-peer networks based on distributed hash tables and to determine the optimal network according to some metrics.

Methods. To determine the optimal system we have introduced the set of metrics, which correspond to the set of criteria's which are critical to the systems comparison. The set of criteria's is as follows: the number of steps in the search algorithm, the operation of calculating the «distance» between the elements of the network, and practical implementation of the system. Next, the criteria are ranked by importance and to each criterion there is a certain coefficient (weight) which is assigned so that the more important criterion strongly influences on the result than the less important. The linear convolution has been calculated for each system and the best system is the one which has the minimal result.

Results. According to the result of calculations, the optimal search criteria is proved by Kademlia system, primarily due to the unique metric of distance between elements of the network.

Conclusion. Analysis and comparison of peer-to-peer networks constructed on the basis of distributed hash tables, shows that, despite the fact that each of them is able to solve the problem of scalability and fault tolerance, the optimum system is Kademlia.

●