УДК 004.056

Математическая модель эффективных алгоритмов поиска и размещения данных в децентрализованной распределенной файловой системе

А.Г.Сметанин, А.Г.Тормасов

Введение

В последние годы быстрое развитие получили технологии организации распределенной обработки информации и высокопроизводительных вычислений. Одним из классов таких технологий являются облачные технологии хранения данных - это модель обеспечения повсеместного и удобного сетевого доступа по требованию к общему пулу конфигурируемых вычислительных ресурсов (например, сетям передачи данных, серверам, устройствам хранения данных), которые могут быть оперативно предоставлены и освобождены с минимальными эксплуатационными затратами и/или обращениями к провайдеру.

Ключевые особенности, которыми должны обладать такие технологии:

- Децентрализация: форма системы коллективных узлов без координации.
- Масштабируемость: система будет одинаково эффективно функционировать при тысячах или миллионах узлов.
- Отказоустойчивость: система будет одинаково надежна (в некотором смысле) с узлами постоянно включающимися, выключающимися и выдающими ошибки.
- Мобильность доступа: система должна быть доступна с тем же качеством из любой географической точки мира
- Расширяемость: новые вычислительные узлы с легкостью добавляются в систему.
- Безопасность: авторизация доступа, контроль доступа, контроль целостности, защита от несанкционированных вторжений извне;
- Изоляция: данные пользователя должны быть изолированы от всех других данных.

К облачным технологиям хранения данных относят различные системы, использующие разнообразные алгоритмы поиска и размещения данных:

1. облачные хранилища (Amazon S3, EMC Atmos, OceanStore)
2. распределенные файловые системы (GFS, Lustre, Ceph, HDFS)
3. децентрализованные распределенные файловые системы(FreeNet, Tahoe-LAFS)

4. p2p системы обмена файлами (BitTorrent, Gnutella)

В данной работе авторами рассматриваются p2p децентрализованные файловые системы, в которых, алгоритм поиска и размещения данных реализован с помощью DHT(англ. Distributed Hash Table — «распределённая хеш-таблица»). В дальнейшем под термином децентрализованная распределенная файловая система будет подразумеваться децентрализованная распределенная файловая система в классе p2p DHT.

DHT — это класс децентрализованных распределённых систем, которые обеспечивают поисковый сервис, похожий по принципу работы на хеш-таблицу, и имеют структуру ассоциативного массива: (ключ, значение), хранящиеся в DHT, а каждый участвующий узел может рационально искать значение, ассоциированное с данным именем. Ответственность за поддержку связи между именем и значением распределяется между узлами, таким образом, изменение набора участников является причиной минимального количества разрывов. Это позволяет легко масштабировать DHT и постоянно отслеживать добавление/удаление узлов и ошибки в их работе. Примеры DHT систем: Chord [1], Pastry[2], CAN [3], Tapestry[4].

Однако у DHT систем есть недостатки, связанные с тем, что данные, которые размещаются в системе, жестко привязаны к узлам системы:

- Chord [1] присваивает идентификаторы узлам и блокам данных из одного одномерного пространства идентификаторов. Узел, который отвечает за хранения блок данных, является узлом, идентификатор которого наиболее близок к идентификатору блока данных
- Pastry [2] присваивает каждому узлу случайный идентификатор, который определяет позицию узла во множестве идентификаторов с круговой топологией. Для поиска блока данных по его идентификатору поисковый запрос отсылается узлу с идентификатором наиближайшим к идентификатору блока данных.
- CAN [3] использует n-мерное пространство декартовых координат для осуществления DHT абстракции. Координатное пространство разбивается на гиперпрямоугольники. Гиперпрямоугольники называют зонами. Каждый узел в системе отвечает за зону, и узел определяет границы своей зоны. Блок данных отображается в точку

в данном пространстве координат, и этот блок данных хранится в узле, зона которого содержит координаты точки.

- Tapestry [4] осуществляет поиск блока данных по его идентификатору, сравнивая определенные биты идентификатора с определенными битами идентификатора узла, поисковый запрос отсылается тому узлу, битовое совпадение с которым наилучшее.

В случае добавления новых узлов к DHT требуется перераспределение размещенных данных по узлам системы. Кроме того миграция данных между двумя узлами осложнена по той же причине.

Лишенная этих недостатков распределенная система рассматривалась в работе[6].Предлагаемая система имеет три основных свойства. Во-первых, данные единицы образуют сеть типа "малого мира", которая гарантирует, что любой блок данных доступен из любого другого блока данных за небольшое количество шагов. Это дает возможность быстро найти любой блок данных, начиная с любого другого блок данных. Во-вторых, все блоки данных упорядочены и задана метрика (расстояние) между двумя блоками данных. Это позволяет построить систему таким образом, что простой жадный алгоритм поиска может быть использован для поиска любого блока данных. В-третьих, блоки данных не привязаны к конкретным узлам, а топология системы основана на графе блоков данных, а не на графе узлов, как например, в случае DHT(Chord) [1].

В работе [6] не рассматриваются вопросы обеспечения отказоустойчивости системы, в том случае если узлы системы склонны выключаться на время, что вполне реально при практическом применении.

В данной работе рассматривается способ обеспечения отказоустойчивости системы за счет построения двухуровневой сети малого мира. Первый уровень состоит из групп узлов, и образуют сеть аналогичную [6], нижележащий второй уровень состоит из блоков данных, которые также образуют сеть типа [6]. В работе проводится математические исследование и моделирование алгоритмов поиска и размещения данных в построенной распределенной файловой системе.

Моделирование поиска и размещения данных в распределенной файловой системе

Для моделирования соединенных по сети Internet компьютеров с хранимыми на них данными используется граф, в вершинах которого находятся отдельные компьютеры системы, а ребра графа представляют собой каналы коммуникации между компьютерами системы. Для создания более реалистичной модели считается, что компьютеры могут, как появляться и присоединяться к распределенной системе, так и пропадать, причем пропадать компьютеры, могут не только штатно (например, компьютер выключен для проведения технического обслуживания), но и в случае возникновения сбоев в компьютере или каналах связи между ними. В данной работе рассматривается построение системы масштаба 10^5 географически удаленных компьютеров. Для эффективной коммуникации такого числа компьютеров, весьма важным являются эффективная топология, обеспечивающая быструю связь между любыми двумя компьютерами системы. Одним из вариантов такой топологии системы может быть модель “малого мира” [5].

“Малый мир” это тип связного графа, в котором, каждая вершина достижима из любой другой за малое количество шагов порядка $\log(N)$, где N – число вершин графа, при этом любая вершина не обязана быть соседом другой напрямую. Существует множество работ описывающих свойства ”малого мира”, авторы рассматривали работу [6] в которой описаны эффективные алгоритмы поиска данных в “малом мире”. За основу взята данная модель, так как она обладает весьма важными свойствами:

- Короткий путь от одной вершины графа к любой другой, пропорционален $\log(N)$, где N – число вершин графа [6].
- Надежность сети образуемой данным графом. То есть, если например, удалить какую-либо вершину из такого графа, тем не менее, в большинстве случаев граф останется связанным, а длина пути из одной вершины до любой другой значительно не изменится.

Модель “малого мира” предполагает, что вершины графа не будут удаляться или добавляться слишком часто. Тем не менее, компьютеры (участники распределенной системы) могут появляться и удалятся гораздо чаще, чем предполагается в модели[6]. Для того чтобы сгладить данное явление, компьютеры объединяются в небольшие группы, а уже группы

объединяются в граф “малого мира”. Группы компьютеров хранят информацию о данных распределенной файловой системы с помощью так называемой (n, k) – схемы описанной в работе [7]. (n, k) – схема это схема избыточного кодирования, которая позволяет разбить исходную информацию на n частей и через продолжительное время восстановить из любых k частей исходную информацию. Таким образом, можно разбить любую информацию на n частей, затем эти части разложить по компьютерам группы для последующего хранения. Возникающие вопросы репликации данных и синхронизации между компьютерами группы в данном методе хранения выходят за рамки этой работы. Для упрощения будем считать, что компьютеры группы хранят идентичные копии информации.

Рассмотрим n компьютеров объединенных в одну группу. Считая что компьютер работоспособен (доступен) с вероятностью p, можно оценить работоспособность группы, как вероятность того что хотя бы один из компьютеров группы включен:

$$P_g = 1-(1-p)^n$$

Например, пусть $p = 0.5$, уже при $n \geq 5$, вероятность $P_g \geq 0.96875$ и близка к 1, что и позволяет объединить группы в “малый мир” и воспользоваться его свойствами.

Авторами, предложена математическая модель описывающая алгоритмы поиска и размещения данных в описанной распределенной системе. Модель описывает систему, которая состоит из двухуровневой сети малого мира: первый уровень состоит из групп узлов, и образуют сеть аналогичную [6], нижележащий второй уровень состоит из блоков данных, которые также образуют сеть типа [6]. Маршрутизация между двумя блоками данных, не лежащих в одной группе, строится с помощью маршрутизации на уровне групп узлов.

Модель системы состоит из следующих частей:

1. модели одного узла: близкие по географической метрике узлы системы объединяются в группы узлов.

2. модели группы узлов: группы узлов образуют так называемую “small world” структуру [6], когда каждая группа связана лишь с небольшим количеством других групп.
3. модели (n; k) схемы [7] преобразования данных для обеспечения сохранности данных.
4. модели кэширования данных: миграции и дупликации данных внутри распределенной системы для более эффективной работы с точки зрения расхода ресурсов.

Математическая модель поиска и размещения информации

В нашей модели данные хранятся в виде пар "ключ" и соответствующее "ключу" значение. Зададим множество уникальных ключей K. Пары ключ значения в распределенной системе хранятся на серверах системы, а сами сервера системы объединяются в группы. Для простоты будем считать, что потери на коммуникацию между серверами внутри группы незначительны в сравнении с потерями на коммуникацию между группами. Тогда будем считать, что ключи лежат внутри групп серверов, пропуская уровень серверов в нашей модели.

Определение 2.1 Множество $GUID$ – множество уникальных целых чисел из числового отрезка $[2^{127}, 2^{128}-1]$.

Алгоритм генерации уникальных 128 битных идентификаторов уже известен и описан в работе [8] (версия 4). Можно оценить для алгоритма [8] (версия 4) вероятность того, что два сгенерированных идентификатора совпадут, с учетом “парадокса дней рождений” [9], приближенной формулой:

$p(n) \approx 1 - \exp(-\frac{n^2}{2x})$, где n – число сгенерированных идентификаторов, $x = 2^{122}$, где 122 – число бит в алгоритме генерируемых случайным образом. Например, при $n = 2^{36} \approx 6.9*10^{10}$, $p \approx 4*10^{-16}$. Хотя уникальность каждого отдельного идентификатора не гарантируется, будем считать, что генерируемые идентификаторы уникальны. Для упрощения считается, что существует функция **gen_guid()**, которая возвращает уникальный ранее не использовавшийся идентификатор из множества $GUID$.

Определение 2.2 Ключом k системы называется кортеж $(id, data, gid)$, где $id \in GUID, data \in B, B -$ множество слов из алфавита $\{0,1\}, gid \in GUID$.

Будем в дальнейшем обозначать для ключа k, что $k.id$ - это идентификатор ключа, $k.data$ - данные ключа, $k.gid$ - идентификатор группы, к которой относится данный ключ.

В качестве данных ключа $k.data$ могут выступать любые бинарные данные. Например, содержимое каждого файла распределенной файловой системы может храниться в поле $k.data$ ключа ассоциированного с файлом.

Определение 2.3 Группа ключей g системы называется кортеж $(id \in GUID, keys \subset K)$.

Будем в дальнейшем обозначать для группы g, что $g.id$ - это идентификатор группы.

Зададим два множества - множество уникальных ключей K и множество групп G. Будем считать, что множество групп G состоит из групп ключей, так что никакие две группы не имеют общих ключей. Далее введем неориентированный граф ключей системы - $G_k(V_{k,}E_k), V_k \subset K$ и неориентированный граф групп системы - $G_g(V_{g,}E_g), V_g \subset G$.

Введем функцию M :

$M : GUID \times GUID \to Z^+, Z^+$ – множество целых положительных чисел

$$M(x, y) = |x - y|$$

, где $x - y$ означает арифметическое вычитание целых чисел, $|x - y|$ означает взятие модуля целого числа.

На множестве групп определим функцию M_g – метрика близости групп по их идентификаторам:

$$M_g : V_g \times V_g \to Z^+$$

$$M_g(g_x, g_y) = \left| g_x.id - g_y.id \right|, \text{ где } g_x, g_y \in V_g$$

На множестве ключей определим функцию M_k – метрика близости ключей по их идентификаторам:

$$M_k : V_k \times V_k \to Z^+$$

$$M_k(k_x, k_y) = \left| k_x.id - k_y.id \right|, \text{ где } k_x, k_y \in V_k.$$

Лемма 2.1 Функции $M_g(g_x, g_y), M_k(k_x, k_y)$ удовлетворяют аксиомам метрики:

1. $M_g(g_x, g_y) = 0 \Leftrightarrow g_x = g_y$ и $M_k(k_x, k_y) = 0 \Leftrightarrow k_x = k_y$
2. $M_g(g_x, g_y) = M_g(g_y, g_x)$ и $M_k(k_x, k_y) = M_k(k_y, k_x)$
3. $M_g(g_x, g_z) \le M_g(g_x, g_y) + M_g(g_y, g_z)$ и
   $M_k(k_x, k_z) \le M_k(k_x, k_y) + M_k(k_y, k_z)$

Доказательство. Доказательство следует из свойств арифметического вычитания и модуля целых чисел. ■

Для ключей и групп из графов $G_k(V_{k,}E_k), G_g(V_{g,}E_g)$ определим функции **neighbours**, **successor**, **predecessor**, **nearest** следующим образом:

$neighbours(x) = \Gamma^+(x)$, где $\Gamma^+(x)$ –множество смежности вершины x.

$$successor(x, id) = z \in \Gamma^+(x) : M(z.id, id) = \min(M(y.id, id)),$$
$$\forall y \in \Gamma^+(x) : y.id > id$$

$$predecessor(x, id) = z \in \Gamma^+(x) : M(z.id, id) = \min(M(y.id, id)),$$
$$\forall y \in \Gamma^+(x) : y.id < id$$

$$nearest(x, id) = z \in \Gamma^+(x) : M(z.id, id) = \min(M(y.id, id)),$$
$$\forall y \in \Gamma^+(x)$$

Приведем описание алгоритмов добавления и поиска ключа/группы в систему в виде псевдокода.

Алгоритм поиска группы по ее идентификатору.

Поиск группы реализован как последовательное приближение к группе с идентификатором равному искомому. Для этого на каждом шаге алгоритма из соседей текущей группы выбирается сосед, с идентификатором ближе по метрике к искомому идентификатору и осуществляется переход в эту соседнюю группу. Алгоритм итеративно повторяется до достижения цели.

group_nearest_search(curr_g, id):

#curr_g - любая группа уже находящаяся в графе групп

#id - идентификатор группы, по которому требуется найти группу

1. d = M(curr_g.id, id)

2. if d = 0 return curr_g

3. nearest_g = nearest(curr_g, id)

4. if M(nearest_g.id, id) < d : curr_g = nearest_g; goto 1;

5. return nearest_g

Алгоритм добавления новой группы

Для добавления новой группы в граф достаточно найти группу с идентификатором ближайшим к идентификатору новой группы, затем у этой ближайшей группы найти соседа который является predecessor или successor для новой группы, затем связать эти две группы с новой:

group_append(curr_g, new_g):

#curr_g - любая группа уже находящаяся в графе групп

#new_g - новая группа, которую необходимо добавить в граф групп

1. new_g.id = gen_guid()

2. nearest_g = group_nearest_search(curr_g, new_g.id)

3. if nearest_g.id = new_g.id return ALREADY_EXISTS

4. if nearest_g.id < new_g.id: neighbour = successor(nearest_g, new_g.id)

5. else: neighbour = predeccessor(nearest_g, new_g.id)

6. mutually connect new_g and neighbour

7. mutually connect new_g and nearest_g

8. return SUCCESS

Алгоритм поиска ключа по его идентификатору

По аналогии с группами строится граф ключей. Ключи связаны между собой близкими по метрике идентификаторами, и для поиска достаточно переходить от одного ключа к другому с идентификатором более близким к искомому, пока не будет найден искомый или наиближайший. Отличием является то, что следующий в пути поиска ключ может находиться в другой группе и эту группу надо найти в графе групп по ее идентификатору:

key_nearest_search(curr_g, id):

#curr_g - любая группа уже находящаяся в графе групп

#id - идентификатор ключа, по которому требуется найти ключ

1. curr_k = curr_g.keys[0]

2. d = M(curr_k.id, id)

2. if d = 0 return curr_k

3. nearest_k = nearest(curr_k, id)

4. if M(nearest_k.id, id) >= d: return nearest_k

5. if nearest_k not in curr_g:

6. curr_g = group_search(curr_g, nearest_k.gid)

7. return key_nearest_search(curr_g, id)

8. else: curr_k = nearest_k; goto 2

Алгоритм добавления нового ключа

Для добавления нового ключа в систему достаточно найти ключ с идентификатором ближайшим к идентификатору нового ключа, затем у этой ближайшего ключа найти соседа который является predecessor или successor для нового ключа и связать эти две ключа с новым:

key_append(curr_g, new_k):

#curr_g - любая группа уже находящаяся в графе групп

#new_k - новый ключ, который необходимо добавить в граф ключей

1. new_k.id = gen_guid()

2. nearest_k = key_nearest_search(curr_g, new_k.id)

3. if nearest_k.id = new_k.id return ALREADY_EXISTS

4. if nearest_k.id < new_k.id: neighbour = successor(nearest_k, new_k.id)

5. else: neighbour = predeccessor(nearest_k, new_k.id)

6. mutually connect new_k and neighbour

7. mutually connect new_k and nearest_k

8. return SUCCESS

Далее докажем, что вышеприведенные алгоритмы корректно и эффективно работают.

Определение 2.4 Предыдущим ключом p по отношению к ключу k, называется такой ключ p, что $p.id < k.id : p \in \Gamma^+(k)$.

Последующим ключом s по отношению к ключу k, называется такой ключ s, что $s.id > k.id : s \in \Gamma^+(k)$.

Предыдущей группой p по отношению к группе g, называется такая группа p, что $p.id < g.id : p \in \Gamma^+(g)$.

Последующей группой s по отношению к группе g, называется такая группа s, что $s.id > g.id : s \in \Gamma^+(g)$.

Определение 2.5 Множество $KeyIds = \bigcup_{k \in V_k} k.id$.

Множество $GroupIds = \bigcup_{g \in V_g} g.id$.

Определение 2.6 Максимальным по идентификатору ключ из множества $V_k \subset G_k(V_k, E_k)$ называется ключ $k_{\max}$, такой что: $k_{\max}.id = \max_{id \in KeyIds} id$.

Минимальным по идентификатору ключ из множества $V_k \subset G_k(V_k, E_k)$ называется ключ $k_{\min}$, такой что: $k_{\min}.id = \min_{id \in KeyIds} id$.

Максимальной по идентификатору группой из множества $V_g \subset G_g(V_g, E_g)$ называется группа $g_{\max}$, такая что: $g_{\max}.id = \max_{id \in GroupIds} id$.

Минимальной по идентификатору группой из множества $V_g \subset G_g(V_g, E_g)$ называется группа $g_{\min}$, такая что: $g_{\min}.id = \min_{id \in GroupIds} id$.

Лемма 2.2

1. $\forall k : k \in V_k \setminus \{k_{\min}, k_{\max}\} \subset G_k(V_k, E_k)$ существуют последующий и предыдущий ключи по отношению к ключу k.

2. У ключа $k_{\min} \in V_k \subset G_k(V_k, E_k)$ существует последующий ключ по отношению к ключу $k_{\min}$.

3. У ключа $k_{\max} \in V_k \subset G_k(V_k, E_k)$ существует предыдущий ключ по отношению к ключу $k_{\max}$.

Доказательство. Следует из построения графа $G_k(V_k, E_k)$, непосредственно из процедуры **key_append**. Для вставки нового ключа процедура **key_append** проходит по вершинам графа в поисках места, где новый ключ станет соседом двух вершин или одной вершины. В случае

двух вершин эти вершины являются предыдущим и последующим ключами по отношению к новому ключу. Случай нахождения одной вершины-соседа - это случай когда вставляемый ключ будет являться $k_{\min}$ либо $k_{\max}$.

Лемма 2.3

1. $\forall g : g \in V_g \setminus \{g_{\min}, g_{\max}\} \subset G_g(V_g, E_g)$ существуют последующая и предыдущая группы по отношению к группе g.

2. У группы $g_{\min} \in V_g \subset G_g(V_g, E_g)$ существует последующая группа по отношению к группе $g_{\min}$.

3. У группы $g_{\max} \in V_g \subset G_g(V_g, E_g)$ существует предыдущая группа по отношению к группе $g_{\max}$.

Доказательство. Следует из построения графа $G_g(V_g, E_g)$, непосредственно из процедуры **group_append**. Для вставки новой группы процедура **group_append** проходит по вершинам графа в поисках места, где новая группа станет соседом двух вершин или одной вершины. В случае двух вершин эти вершины являются предыдущим и последующим группами по отношению к новой группе. Случай нахождения одной вершины-соседа - это случай когда вставляемая группа будет являться $g_{\min}$ либо $g_{\max}$.

Определение 2.7 $D(k_x, k_y)$ – функция расстояния между ключами системы определенная на множестве ключей, и равна числу межгрупповых переходов в процедуре **key_nearest_search**, то есть числу вызовов **group_search** в **key_nearest_search**.

Таким образом,

$D(k_x, k_y) = 0$, если $k_x = k_y$.

$D(k_x, k_y) = 0$, если $k_x.gid = k_y.gid$, то есть принадлежат одной группе.

Теорема 1 $\forall k_x \neq k_y : k_x, k_y \in G_k(V_{k,} E_k)$ верно, что

1. существует путь от ключа k_x до ключа k_y.
2. путь от ключа k_x до ключа k_y конечен.
3. если идентификаторы ключей и групп, которые добавлялись последовательно в систему, равномерно распределены на множестве $GUID$, то длина пути - $D(k_x, k_y) \sim O(\log(|V_g|))$.

Доказательство.

Докажем пункт 1. Для любого ключа $k \neq k_y$ существует ключ $x \in \Gamma^+(k)$ который ближе по метрике M_k к k_y. Действительно, в качестве такого ключа x для ключа k мы можем выбрать его предыдущий ключ или последующий ключ на основе леммы 2.2. Таким образом, можно построить путь от любого ключа $k \neq k_y$ к ключу k_y, в том числе путь от k_x до k_y. Если $k_x.id < k_y.id$, то мы можем построить путь:

$k_x, k_i, k_{i+1}, ..., k_N, k_y$ такой, что:

$k_x.id < k_i.id < k_{i+1}.id < ... < k_N.id < k_y.id$

Иначе $k_x.id > k_y.id$, и мы можем построить путь:

$k_x, k_i, k_{i+1}, ..., k_N, k_y$ такой, что:

$k_x.id > k_i.id > k_{i+1}.id > ... > k_N.id > k_y.id$

Докажем пункт 2. Из доказательства пункта 1, построенный путь будет конечен, так как последовательность идентификаторов ключей проходимых в построенном пути конечна. Действительно, на отрезке $[k_x.id, k_y.id]$ найдется лишь конечное число идентификаторов ключей, так как сами идентификаторы уникальны и являются целыми положительными числами. В построенном пути не один из ключей не встречается дважды.

Докажем пункт 3. Вычисление длины путь от k_x до k_y упрощается, если рассматривать межгрупповой путь от группы к которой принадлежит

k_x до группы к которой принадлежит k_y. К такому упрощению можно перейти из определения функции $D(k_x, k_y)$, а именно из того факта что путь между ключами одной группы равен 0. Следовательно, длина пути $D(k_x, k_y)$ равна длине межгруппового пути от группы, к которой принадлежит k_x до группы, к которой принадлежит k_y. Граф групп $G_g(V_g, E_g)$ аналогичен структуре описанной в работе [6], причем выполняются те же самые предположения относительно распределения входных данных. В работе [6] доказывается, что длина пути от одной вершины структуры до другой $\sim O(\log(N))$, где N – число элементов структуры. Для аналогичного графа $G_g(V_g, E_g)$ справедлива такая же оценка длины пути. Следовательно, $D(k_x, k_y) \sim O(\log(|V_g|))$. ■

Результаты имитационного моделирования

Авторами разработан комплекс программ, реализующий имитационную модель системы, были проведены эксперименты по измерению длины пути поисковых запросов в системе.

Целью имитационного моделирования являлось подтверждение полученных математических оценок длины пути поискового запроса. Для этого проводились измерения длины пути поискового запроса в зависимости от числа групп.

Алгоритм проведения испытания следующий:

1. Пусть число групп = 2, число ключей 1024.
2. Создается заданное количество групп, которые объединяются в сеть “малого мира”. Идентификаторы групп генерируются с помощью функции **gen_guid()**. Считаем, что качество используемого генератора случайных чисел приемлемо для эксперимента.
3. В каждой группе размещается заранее заданное количество ключей (для каждой группы одинаковое число ключей), которые объединяются в сеть “малого мира”. Число ключей размещаемых в каждой группе равно (число ключей)/(число групп).
   Идентификаторы генерируются с помощью функции **gen_guid().**

4. Далее случайным образом выбираются два различных ключа в системе. С первого ключа запускается поиск второго ключа по его идентификатору. Записывается длина пути запроса.
5. Повторяем с шага 3. 100 раз для различных пар ключей.
6. Усредняем по числу запросов длину пути запроса.
7. Если число групп не превышает число ключей, тогда удваиваем число групп, и переходим к шагу 2. Если же число групп превышает число ключей, тогда измерения закончены.

На графике (рисунок 1) приведены результаты измерения длины поискового запроса в зависимости от числа групп в системе. Видно, что длина поискового запроса (длина пути от одного ключа к другому) в среднем логарифмически зависит от числа групп в системе.

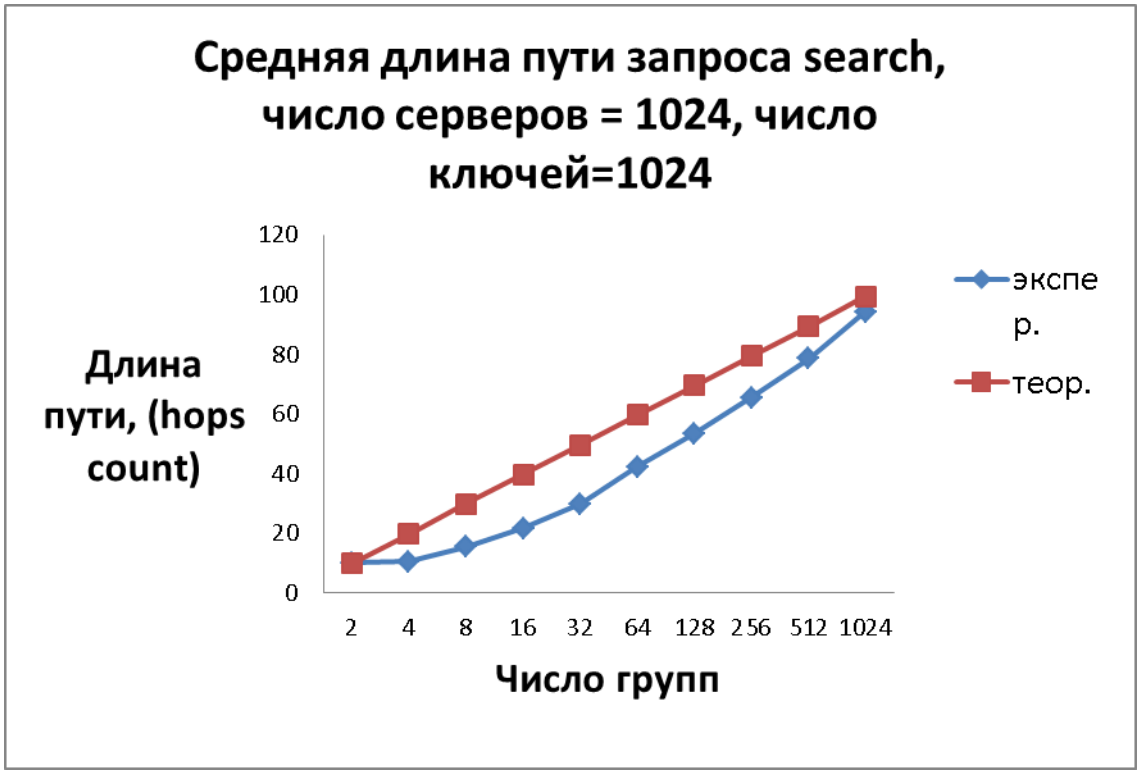

Рисунок 1. Зависимость длины пути запроса search от числа групп в системе

Далее мы добавили в имитационную модель уровень серверов хранящих ключи с данными, которые объединялись в группы серверов. На графике (рисунок 2) приведены результаты измерения длины поискового запроса в зависимости от числа серверов в системе при фиксированном числе групп. Видно, что длина поискового запроса (то есть путь от одного

ключа к другому) в среднем логарифмически зависит от числа серверов в системе.

Рисунок 2. Зависимость длины пути запроса search от числа серверов в системе

Заключение

Авторы предложили модель распределенной системы, построенную как двухуровневая сеть малого мира, первый уровень состоит из групп узлов и образует сеть аналогичную [6], нижележащий второй уровень состоит из блоков данных, которые также образуют сеть типа [6]. Предложены алгоритмы поиска и добавления данных, а также поиска и добавления узлов системы.

На основе предложенной модели и алгоритмов была построена математическая модель системы. Рассматривая данную математическую модель системы, доказана теорема о работоспособности и эффективности данной системы, а именно была получена математическая оценка длины поиска в данной системе.

Разработан комплекс программ, реализующий имитационную модель системы, были проведены эксперименты по измерению длины поискового

запроса в системе. Установлено, что экспериментальные результаты соответствуют полученной оценке длины поискового запроса.

Список литературы

[1] I. Stoica, R. Morris, D. Karger, M.F. Kaashoek, H. Balakrishnan “Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications”, Proc. ACM SIGCOMM, August 2001.

[2] Owstron, A., Druschel, P. Pastry: Scalable, distributed object location and routing for large-s cale peer-to-peer systems. In Proceedings of the 18th IFIP/ACM International Conference on Distributed Systems Platforms (Middleware 2001) (Nov. 2001).

[3] Ratnasamy, S., Francis, P., Handley, M., Karp, R., Shenker, S. A scalable contentaddressable network. In Proc. ACM SIGCOMM (San Diego, CA, August 2001), pp. 161–172.

[4] Hildrum, K., Kubatowicz, J. D., Rao, S., Zhao, B. Y. Distributed Object Location in a Dynamic Network. In Proc. 14th ACM Symp. on Parallel Algorithms and Architectures (Aug. 2002).

[5] Watts, Duncan J.; Strogatz, Steven H. (June 1998). "Collective dynamics of ’smallworld’ networks". Nature 393 (6684): 440–442

[6] V. Krylov, A. Logvinov, A. Ponomarenko, D.Ponomarev Metrized Small World Properties Data Structure, Proc. Software Engineering and Data Engineering (SEDE 2008).

[7] Тормасов А.Г., Хасин М.А., Пахомов Ю.И. Обеспечение отказоустойчивости в распределенных средах // Программирование – 2001. - № 5. - с. 26-34.

[8] RFC 4122, A Universally Unique IDentifier (UUID) URN Namespace, http://www.ietf.org/rfc/rfc4122.txt

[9] M. Klamkin and D. Newman, Extensions of the Birthday Surprise, (1967), Journal of Combinatorial Theory 3, 279–282.