Санкт-Петербургский Государственный Университет
Математико-механический факультет
Кафедра системного программирования

Архитектура промышленной децентрализованной файлообменной сети

Курсовая работа студента 444 группы
Булычева Антона Дмитриевича

Научный руководитель Щитинин Д.А.

Санкт-Петербург
2014

Оглавление

Введение

Если раньше вычислительная система могла состоять из лишь единственного компьютера, то к настоящему времени ситуация коренным образом изменилась. Случилось это по нескольким причинам:

- удешевление компьютеров
- возросшие требования к вычислительным системам: увеличился объем данных, и старые технологии уже не могли обеспечить нужного быстродействия
- появление и распространение стека протоколов TCP/IP, облегчающего взаимодействие между парой компьютеров

Все это привело к необходимости масштабирования вычислительной системы на несколько компьютеров. И теперь вопрос передачи файлов между ними занимает ключевое место.

Файлообенной сеть — это компьютерная сеть для совместного использования файлов, основанная на равноправии участников обмена данными. Участники обмена называются узлами файлообменной сети. Файлобменные сети, в которых ровно два узла, не представляют технической сложности, поэтому в данной курсовой работе будут рассматриваться случаи, когда их количество превышает два.

Виды сетей

Различают следующие виды сетей: централизованные, децентрализованные и частично децентрализованные сети.

В **централизованной** файлообменной сети вся служебная информация хранится на специально выделенном сервере. Можно выделить особый вид централизованной сети — с внешним хранилищем. Каждый участник, желающий опубликовать файл, загружает его на специально выделенный сервер, после чего любой другой участник может забрать его оттуда. Существует также вариант сети, использующий распределенное внешнее хранилище для обеспечения отказоустойчивости.

Децентрализованные файлообменные сети — это сети, в которых отсутствуют вспомогательные сервера.

Частично децентрализованные файлообменные сети — сети, состоящие из нескольких вспомогательных серверов, периодически синхронизирующих данные между собой.

Применение

Можно выделить следующие сферы применения: промышленное и непромышленное.

Непромышленным использованием является обмен файлами между людьми. Подобные системы отличаются большим числом узлов (может достигать от 100 тысяч до нескольких миллионов) . В данной сфере нет жестких требований на время публикации и поиска файла. В таких сетях допустимо то, чтобы опубликованный файл был временно недоступен.

При **промышленном** использовании обмен файлами, как правило, является вспомогательной задачей.

Рассмотрим пример поискового сервиса в интернете. Он состоит из нескольких частей, ключевыми из которых являются подготовка данных и осуществление поиска по ним. Данные индексируются на одной группе узлов, поиск же осуществляется на другой. Между этими группами узлов нужно осуществлять быструю и надежную доставку, что напрямую влияет на качество и стабильность сервиса.

У промышленных сетей, в отличие от непромышленных, нет большого количества узлов (от нескольких штук до нескольких тысяч), но к ним предъявляют следующие требования:

- высокая скорость публикации файлов
- высокая скорость поиска файлов
- высокая доступность — опубликованный файл остается доступным до тех пор, пока он будет содержаться хотя бы на одном из доступных узлов или пока не будет удален
- получение уведомлений о публикации новых файлов

Сейчас промышленные файлообменные сети, как правило, являются централизованными. Главным недостатком такого подхода является его ненадежность. Настройка же такой сети очень проста: достаточно сконфигурировать один сервер и сообщить его адрес узлам файлообменной сети.

Если же требуется обеспечить надежность, то используют распределенное внешнее хранилище. Но при таком подходе нужно использовать выделенный кластер машин, что приводит к сложной настройке, трудностям администрирования и высокой стоимости решения.

Гибридные системы очень специфичны и также требуют выделения специальных серверов.

Существующие децентрализованные файлообменные сети не подходят для промышленного использования, так как не обеспечивают высокой доступности данных, и они не позволяют отслеживать публикации новых файлов.

Существование же промышленной децентрализованной сети помогло бы разработчикам и системным администраторам упростить процесс создания и эксплуатации больших распределенных систем.

Постановка задачи

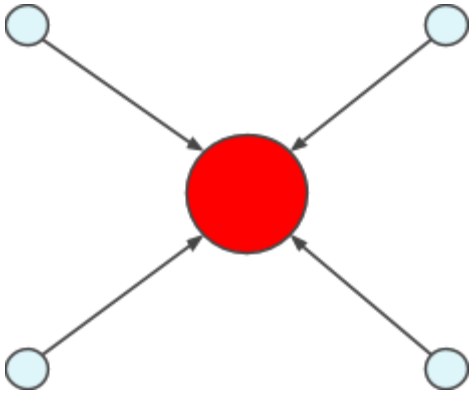
В данной работе решаются следующие задачи:

- обзор существующих файлообменных сетей
- разработка прототипа промышленной децентрализованной файлообменной сети, которая будет иметь следующие свойства:
    - высокая скорость публикации файлов
    - высокая скорость поиска файлов
    - высокая доступность
    - возможность получения уведомлений о публикации новых файлов

Обзор существующих решений

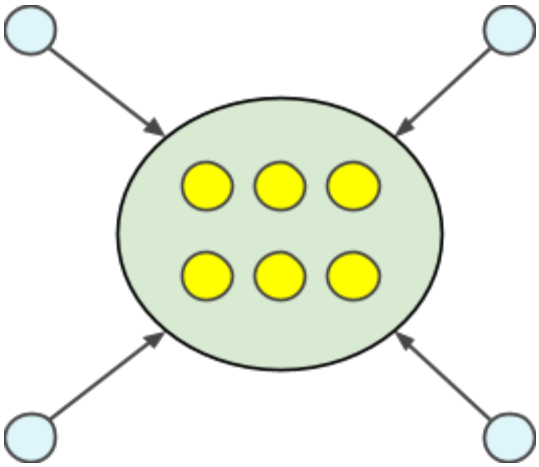
Файлообменная сеть с внешних хранилищем

Одним из способов организовать файлообменную сеть может служить система, использующая для хранения файлов некоторое постоянное внешнее хранилище. В качестве хранилища может выступать файловая система компьютера с высокой производительностью. Когда пользователь хочет опубликовать файл, он загружает его в это хранилище. Одними из первых протоколов, реализующих подобный подход, были FTP[1], NFS[2], HTTP[3].



Если разместивший файл узел по какой-либо причине перестанет быть доступным, то файл останется доступным для других узлов, что является важным преимуществом. Также эти сети легки в настройке и эксплуатации. Но есть у них и недостаток: хранилищем является один компьютер, и в случае его выхода из строя файлообменная сеть перестанет функционировать.

Данную проблему могут решить *распределенные файловые хранилища*, такие как HDFS[4], GlusterFS[5] и им подобные. Их особенность заключается в том, что файлы хранятся не на одном компьютере, а на целом кластере. Хранение информации организовано с помощью дублирования на нескольких узлах, поэтому при возникновении проблем даже с несколькими из них, информация все равно останется доступной.

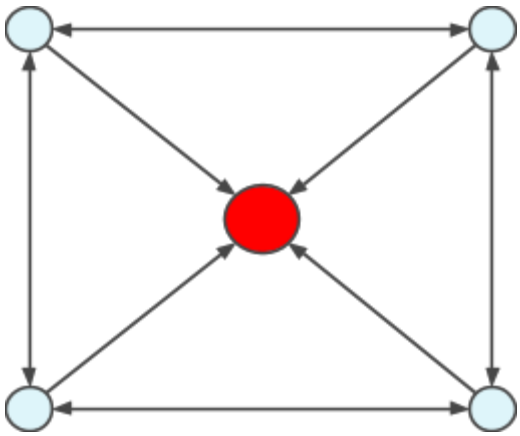


При данном подходе, получая высокую надежность и доступность, система имеет такие недостатки, как высокая стоимость оборудования, сложность настройки и эксплуатации системы.

В промышленных системах для передачи данных чаще всего используют файлообменные сети с внешним хранилищем, так как они предсказуемы и позволяют каждом узлу легко отслеживать публикацию новых файлов.

Централизованные файлообменные сети

В централизованных файлообменных сетях выделяют сервер, который хранит как метаинформацию о файле (размер файла, дату создания и прочее), так и адреса узлов, на которых этот файл можно найти.



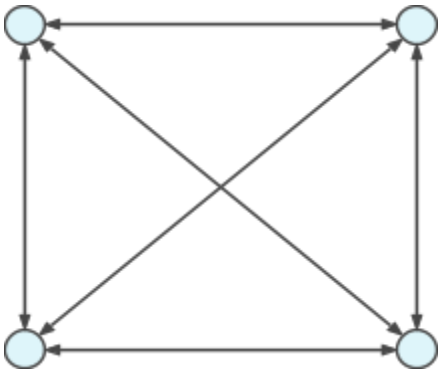
Ярким примером такой сети является Napster[6] — файлообменная сеть, действовавшая с 1999 года по 2001 год. Сервис позволял легко обмениваться музыкальными файлами с другими людьми, что привело к обвинениям в нарушении авторских прав со стороны музыкальной отрасли. Сервис был остановлен по решению суда. Примерами также еще являются DC[7] и BitTorrent[8] (если не используется расширение с DHT[9]).

К достоинствам централизованного подхода можно отнести сравнительную простоту программирования такой сети и небольшой объём служебной информации, передаваемой на серверы. Кроме того, с такой сетью легко работать, и настройка не занимает много времени.

Недостатком всех централизованных систем является наличие единой точки отказа.

Децентрализованные сети

Децентрализованные файлообменные сети функционируют без выделенных серверов. В таких сетях объём передаваемой служебной информации больше, но зато их надёжность гораздо выше. К децентрализованным сетям относится, например, Kad[10].



Большинство децентрализованных сетей в своей реализации используют для поиска узлов, содержащих нужную информацию, *распределенную хеш-таблицу*. Для того, чтобы говорить о распределенной хеш-таблице, надо сначала пояснить понятие простой хеш-таблицы.

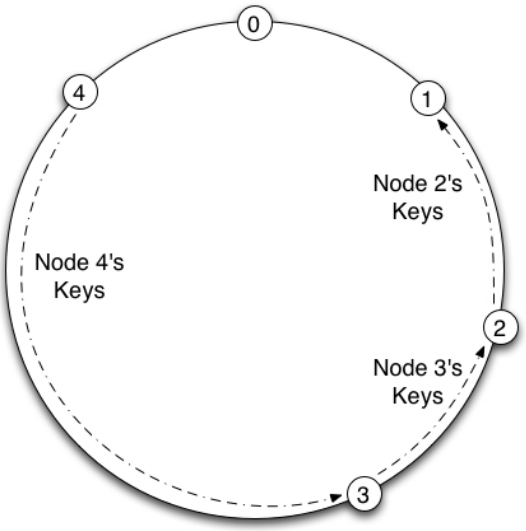
Хеш-таблица — это структура данных, позволяющая хранить пары ключ-значение и выполнять три операции: добавление новой пары, поиск и удаление пары по ключу.

Распределённая хеш-таблица отличается от вышеописанной тем, что таблица хранится не на одном узле, а распределена между узлами системы — таким образом, каждый узел хранит лишь ее часть.

Задача распределения ключей по узлам решается следующим образом. Пусть все ключи принадлежат некоторому множеству K, например, множеству 160-битных чисел (это длина значения SHA-1 хеша). Пусть также задана функция δ: $K \times K \to \mathbb{R}$, определяющая абстрактное понятие расстояния между парой ключей. Каждому узлу a присваивается уникальный идентификатор ID_a, выбранный из множества K. Тогда ключом k владеет узел a с наименьшим значением $\delta(k, ID_a)$. Также существуют варианты, когда ключом k владеет

не один, а несколько узлов с наименьшими значениями функции — такой подход помогает избежать потери данных, если узел перестает быть доступным.

Одной из первых реализаций распределенной хеш-таблицы был Chord DHT[11]. Chord DHT рассматривает ключи как точки на окружности, а $\delta(k_1, k_2)$ — это расстояние, пройденное по часовой стрелке по окружности от ключа k_1 к k_2. Таким образом, круг пространства ключей разделён на смежные сегменты, чьи концы являются идентификаторами узлов. Если ID_a и ID_b следуют друг за другом по часовой стрелке соответственно, то узел b содержит все ключи, которые находятся между ID_a и ID_b.



При непромышленном использовании распределенной хеш-таблицы количество узлов может достигать огромного числа, поэтому отслеживание входов и выходов узлов из системы не представляется возможным. Следовательно, каждый узел хранит лишь некоторое подмножество всех узлов, зависящее от реализации. Обычно мощность этого подмножества равна числу порядка log N, где N — количество всего узлов в системе. Также от реализации зависит алгоритм поиска узлов, ответственных за ключ.

В децентрализованных файлообменных сетях распределенные хеш-таблицы применяются следующим образом.

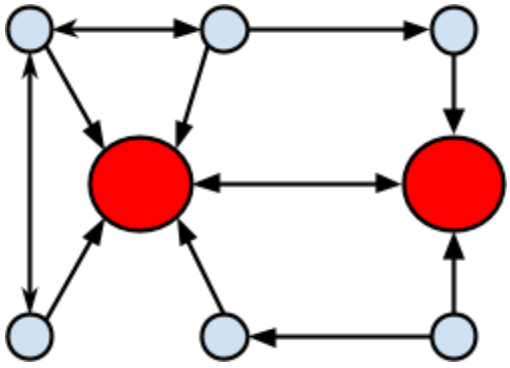
Если узел с адресом A хочет опубликовать файл, он вычисляет хеш-функцию от имени файла — пусть она равна k — и записывает в распределенную хеш-таблицу пару (k, A).

Для того, чтобы данные не потерялись из-за входа и выхода узлов из сети, узел периодически перезаписывает эту пару. Когда узел хочет узнать, где находится некоторый файл, он вычисляет хеш-функцию от имени файла и ищет в распределенной хеш-таблице пару с этим ключом. Значение этой пары будет адресом узла, на котором находится этот файл.

Все реализации данного подхода создавались для непромышленного использования. Для промышленного использования необходима возможность уведомления о публикации новых файлов, которую не позволяют данные реализации.

Частично децентрализованные сети

Частично децентрализованные файлообменные сети встречаются довольно редко. Они были попыткой решить проблему отказоустойчивости централизованного подхода, используя вместо одного служебных серверов несколько. Эти сервера периодически синхронизируют информацию между собой. Каждый участник обращается лишь к одному из серверов, тем самым снижая нагрузку. Если же один из серверов выйдет из строя, участники начнут обращаться к другому, и сеть продолжит функционировать.
Недостатком сети является то, что из-за наличия нескольких вспомогательных серверов возрастает стоимость такого решения.



К частично децентрализованным файлообменным сетям относится OpenNap[12].

Предлагаемое решение: архитектура и детали реализации

Промышленные системы, как было сказано ранее, используют для решения своих задач централизованные файлообменные сети, чаще всего с внешним хранилищем. Децентрализованные сети не подходят для этих целей из-за того, что создавались для непромышленного использования, и имеют следующие ограничения:

- отсутствие высокой доступности: возможная недоступность опубликованного файла, даже если он содержится на одном из доступных узлов
- невозможность получения уведомления о публикациях новых файлов

Далее в работе будет предложена архитектура решения, удовлетворяющая следующим требованиям:

- высокая скорость публикации файлов
- высокая скорость поиска файлов
- высокая доступность
- возможность получения уведомлений о публикации новых файлов

Архитектуру можно разбить на четыре уровня:

1. кластерный
2. передачи данных и метаданных
3. поиска и адресации
4. пользовательский интерфейс

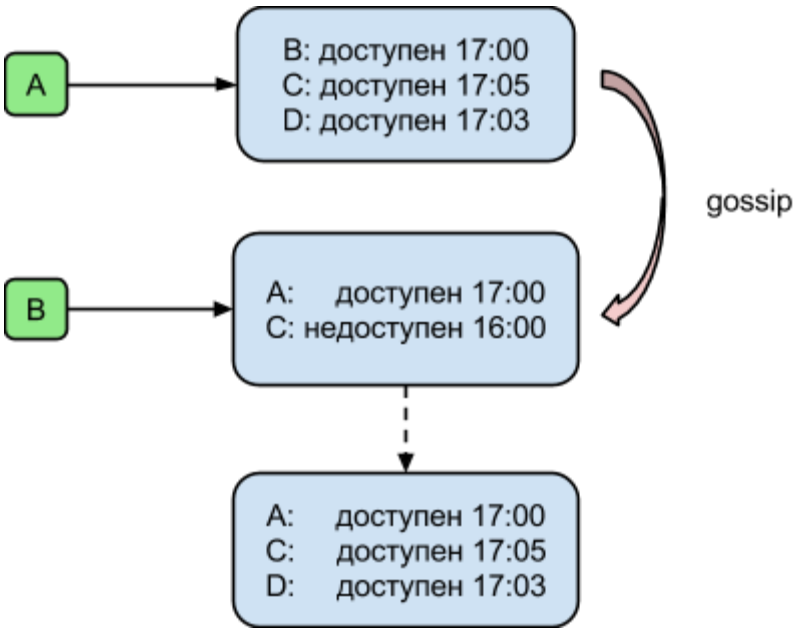
Кластерный уровень

На данном уровне решаются задачи взаимодействия и организации узлов:

- выделение уникального ключа для узла
- определение активных в данный момент узлов
- передача различной информации всем активным узлам сети

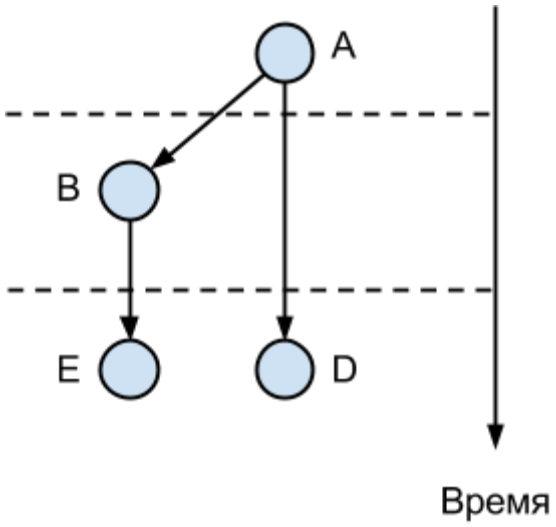
В качестве уникального ключа берется 160-битное число. Выделить такой уникальный ключ можно как SHA-1 хеш от адреса узла.

Каждый узел хранит список известных ему узлов. Кроме того, он также хранит информацию об их статусе: доступность и время ее последней проверки. Этой информацией узлы обмениваются с помощью *gossip*-протокола[13].



Идею обменом с помощью *gossip*-протокола можно проиллюстрировать следующим образом.

Пусть имеется множество узлов, обозначенных латинскими буквами, и узел *A* хочет оповестить всех о том, что узел *C* доступен. Для этого он выбирает случайно один из доступных ему узлов — пусть это будет узел *B* — и сообщает ему эту информацию. Теперь уже два узла, *A* и *B*, случайно выбирают узлы, которым передадут информацию, — пусть это будут *D* и *E*, соответственно. Таким образом, в силу случайного выбора узлов, информация распространится до всех доступных узлов.



Сила такого подхода основывается на надежности передачи информации: даже если бы узел *E* не получил по каким-либо причинам сообщения от *B*, то он наверняка бы скоро получил информацию от какого-нибудь другого узла.

Выражая эту идею более формально, *gossip*-протокол обладает свойствами:

- участники взаимодействуют попарно
- есть ограничение на размер передаваемой информации при каждом взаимодействии
- не требует надежности коммуникации

- выбор участников для взаимодействия производится случайно

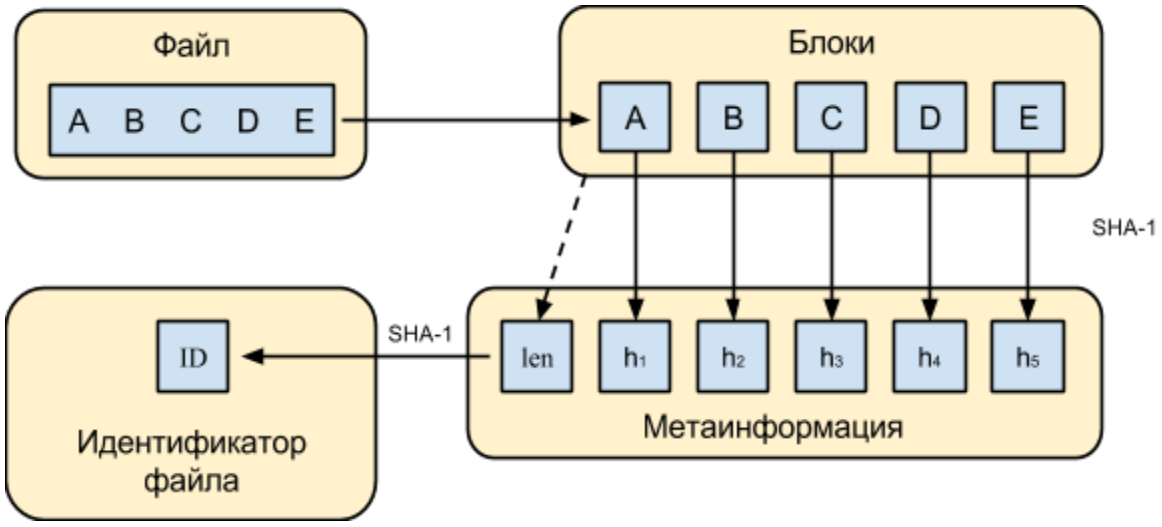
Каждый узел с заданной частотой посылает gossip-сообщение с имеющейся информацией об известных ему узлах. Помимо gossip-сообщений, эту информацию каждый узел получает следующим образом. Пусть узел *A* хочет узнать, доступны ли известные ему узлы. Для этого он выбирает некоторый набор узлов из своего полного списка и посылает им сообщение *ping*. Каждый узел, получив такое сообщение, должен ответить отправителю *pong*. Если после отправки сообщения *ping* узел *A* не дождется сообщения *pong*, он повторит запрос еще несколько раз. Если же интересующий нас узел не ответил на какое-то пороговое число запросов *ping*, узел *A*, отмечает его как недоступный. Если же какой-то узел числился недоступным в списке узла *A*, но ответил *pong* на запрос, то его статус меняется на “доступен”. Как только *A* узнает о смене статуса какого-либо узла, он спешит поделиться этой информацией и сразу же посылает *gossip*-сообщение.

Для функционирования подобного подхода одним из требований является синхронность времени на узлах. Данное требование в наше время является легко осуществимым.

Уровень хранения и передачи данных и метаданных

Данный уровень отвечает за обмен информацией о файле о файле.

Если файл находится на нескольких узлах, то его можно загружать с них одновременно. При этом нагрузка распределится по этим узлам и, возможно, уменьшится итоговое время загрузки. Для обеспечения такой возможности файлы разбиваются на блоки, и его последующая загрузка осуществляется поблочно. Каждый блок имеет свой идентификатор, вычисляемый как SHA-1 хеш-функция от его содержимого. С каждым файлом связана метаинформация — длина файла и список блоков, из которых он состоит. Также всем файлам присваивается идентификатор, равный значению SHA-1 хеш-функции от метаданных.



Каждый узел хранит список опубликованных файлов сети в виде файлового дерева. При создании, изменении или удалении файла изменяется локальное файловое дерево. Файловые деревья различных узлов синхронизируются с помощью *gossip* протокола, реализованного на кластерном уровне.

API этого уровня содержит следующие методы:

- получение локального файлового дерева
- получения метаинформации о файле по его идентификатору
- получение содержимого блока по его идентификатору

Подобную функциональность просто реализовать в стиле REST:
REST (Representational state transfer) — это стиль архитектуры программного обеспечения для распределенных систем, таких как World Wide Web, который, как правило, используется для построения веб-служб. Термин REST был введен в 2000 году Роем Филдингом, одним из авторов HTTP-протокола. Системы, поддерживающие REST, называются RESTful-системами. REST базируется на понятии ресурса, который однозначно определяется идентификатором. Наиболее часто REST интерфейс реализуют посредством HTTP-сервера. В таком случае в качестве идентификатора выступает URL. Каждый ресурс поддерживает определенный набор действий. Для HTTP реализации это могут быть GET (получить), PUT (добавить, заменить), POST (добавить, изменить, удалить), DELETE (удалить)[14].

Таким образом, в терминах REST API запросы этого уровня будут звучать так:

- GET */fs* — возвращает локальное файловое дерево узла
- GET */files/$id/metadata* — возвращает метаданные файла
- GET */blocks/$id/data* — возвращает содержимое блока

Уровень поиска и адресации

Данный уровень отвечает за поиск узла, на котором находится информация.

В качестве реализации можно взять распределенную хеш-таблицу, понятие которой описывалось ранее.

Ключами хеш-таблицы являются идентификаторы файлов и блоков, значениями — адреса узлов, содержащие их. Так как ключи вычисляются как *SHA-1* хеш, то множество всех ключей — множество всех 160-битных чисел.

В качестве функции расстояния $\delta(k_1, k_2)$ берется функция, аналогичная *Chord DHT*: если расположить ключи на кольце, то расстоянием будет длина пути, пройденного по часовой стрелке по окружности от k_1 до k_2. Но в отличие от непромышленных реализаций распределенной хеш-таблицы, класстерный уровень позволяет отслеживать всех участников системы. Благодаря этому каждый узел может при необходимости переозаписывать пары ключ-значения при выходе и входе узлов. Таким образом, хеш-таблица не будет терять данные.

Работа файлообменной сети выглядит так.

Если участник с адресом A хочет опубликовать файл, он поступает следующим образом:

- делит файл на блоки, вычисляет их идентификаторы ID_k, формирует метаинформацию о файле, вычисляет идентификатор файла ID
- сохраняет метаинформацию
- для каждого блока с идентификатором ID_k добавляет в распределенную хеш-таблицу пару (ID_k, A)
- добавляет в распределенную хеш-таблицу пару (ID, A)
- добавляет файл в локальное файловое дерево и с помощью *gossip* оповещает об изменении

Если участник с адресом A хочет получить файл с идентификатором ID, он поступает следующим образом:

- получает из хеш-таблицы по ключу *ID* значения, которые являются адресами узлов
- обращается к этим узлам за метаинформацией о файле, используя API уровня хранения и передачи данных и метаданных
- сохраняет метаинформацию
- добавляет в хеш-таблицу пару (ID, A)
- для каждого блока с идентификатором ID_k:
    - получает из хеш-таблицы по ключу ID_k значения, которые являются адресами узлов
    - обращается к этим узлам за содержимым блока, используя API уровня хранения и передачи данных и метаданных
    - сохраняет блок
    - добавляет в распределенную хеш-таблицу пару (ID_k, A)

Для функционирования этого уровня каждый узел должен реализовать следующий интерфейс взаимодействия с ним:

- добавить пару ключ-значение
- получить по ключу все значения, сохраненные на данном узле

Этот интерфейс можно реализовать также в стиле REST. Тогда запросы будут иметь следующий вид:

- PUT */dht/$key*

  {$value}
- GET */dht/$key*

Пользовательский интерфейс

Так как список файлов на узлах хранится в виде дерева, интерфейс доступа к локальному файловому дереву можно реализовать в виде файловой системы. Для большинства ОС известны следующие способы реализации:

- реализация модуля ядра ОС
- реализация интерфейсов FUSE[15]

Модуль ядра ОС — это объект, содержащий код, расширяющий функциональность ОС. Для загрузки модуля в ядро требуются соответствующие привилегии. Подобный подход не является кроссплатформенным, поэтому для каждой операционной системы придется реализовывать свой модуль.

FUSE (Filesystem in Userspace) — модуль для ядер UNIX-подобных операционных систем, позволяющий пользователям без привилегий создавать их собственные файловые системы. Код файловой системы запускается в режиме пользователя, в то время как FUSE является мостом для интерфейсов ядра. В данном подходе требуется реализовать интерфейсы FUSE, которые не будут зависеть от платформы.

FUSE уступает в производительности модулю ядра, так как является лишь интерфейсом взаимодействия с виртуальной файловой системой, в то время как модуль ядра может взаимодействовать с ней напрямую. Но несмотря на это, он позволяет более простым способом реализовать файловую систему, так как обладает следующими свойствами:

- является кроссплатформенным и не требует изменения кода ядра ОС
- исполняется в пользовательском режиме и является безопасным для ОС
- позволяет легко отлаживать программы

Поэтому в работе было отдано предпочтение FUSE.

Используемые технологии

Для реализации файлообменной сети были использованы следующие технологии:

- Java[16] — объектно-ориентированный язык программирования, компилируемый в Java байт-код. Выбор обусловлен кроссплатформенностью виртуальной машины (JVM), а также наличием большого числа открытых библиотек
- Scala[17] — функциональный язык программирования, компилируемый в Java байт-код. Обладает более выразительными средствами, что позволяет существенно уменьшить объем кода и ускорить процесс разработки приложений
- Akka[18] — библиотека, реализующая Actor model. Позволяет просто и безопасно писать многопоточные приложения
- Jetty[19] — библиотека, реализующая HTTP-сервер
- FUSE[20] — модуль для ядер UNIX-подобных ОС, позволяющий создавать файловые системы в пользовательском режиме
- fuse-jna[21] — Java библиотека для работы с FUSE

Результаты

В рамках данной курсовой работы были решены следующие задачи:

- обзор существующих архитектур файлообменной сети
- описание и реализация прототипа децентрализованной файлообменной сети для промышленного использования, обладающего свойствами:
    - высокая скорость публикации файлов
    - высока скорость поиска файлов
    - высокая доступность
    - возможность получения уведомлений о публикации новых файлов

Исходный код доступен по адресу https://github.com/abulychev/dfs

Дальнейшее развитие

Дальнейшим развитием системы является вывод ее из прототипа, а именно:

- оптимизация
- покрытие тестами
- средства мониторинга
- документация
- введение в эксплуатацию в промышленных сервисах

Список литературы

1. File Transfer Protocol RFC — http://www.ietf.org/rfc/rfc959.txt
2. Network File System RFC — http://www.ietf.org/rfc/rfc3530.txt
3. HyperText Transfer Protocol RFC — http://www.ietf.org/rfc/rfc2616
4. Hadoop Distributed File System — http://hadoop.apache.org/docs/r1.2.1/hdfs_design.html
5. GlusterFS — http://www.gluster.org/
6. Napster — http://en.wikipedia.org/wiki/Napster
7. Direct Connect — http://en.wikipedia.org/wiki/Direct_Connect_(protocol)
8. BitTorrent — http://www.bittorrent.org/beps/bep_0000.html
9. BitTorrent Enhancement Proposal: DHT — http://www.bittorrent.org/beps/bep_0005.html
10. Kad — http://en.wikipedia.org/wiki/Kad_network
11. Chord DHT — http://en.wikipedia.org/wiki/Chord_(peer-to-peer)
12. OpenNap — http://opennap.sourceforge.net/
13. Gossip — http://en.wikipedia.org/wiki/Gossip_protocol
14. REST — http://habrahabr.ru/post/38730/
15. FUSE — http://fuse.sourceforge.net/
16. Java — http://www.oracle.com/technetwork/java/index.html?ssSourceSiteId=opn
17. Scala — http://www.scala-lang.org/
18. Akka — http://akka.io/
19. Jetty — http://www.eclipse.org/jetty/
20. fuse-jna — https://github.com/EtiennePerot/fuse-jna