ПРИМЕНЕНИЕ (N, k) ПОРОГОВЫХ СХЕМ ОБЕСПЕЧЕНИЯ ДОСТУПНОСТИ INTERNET-СЕРВНСОВ

М.А. Хасин

Московский физико-технический институт (Государственный Университет)

Abstract

M.A. Khassine. Using (N, к) threshold scheme to provide availability of the Internet-services. In the article the method of the distributed data storing is considered. The method is based on the model when any ftle is represented as a (N, k) threshold scheme. The algorithms of splitting and assembling offiles are based on finite field fields math (Galois field GF). The file is represented as a sequence of K-dimension vectors above GF(2") field, and each part of the file in the threshold scheme is a sequence of projections of these vectors on some selected for the given part K-dimension vector above GF. In the article the implementation of the method for n=16 is adduced and the method application is researched to provide availability ofthe Internet-service systems.

Введение

На фоне бурного развития Internet-технологий, все более актуальной становится задача построения системы, которая бы обеспечивала возможность хранения огромных объемов информации и в то же время гарантировала стабильное время доступа к информационному ресурсу, а также включала в себя механизмы обеспечения безотказного доступа к информации.

Большинство существующих систем, предоставляющих различные Internet-сервисы, как правило, управляют различными информационными ресурсами, которые распределены между различными компьютерами сети (далее эти компьютеры, принадлежащие системе, будем называть серверами). При построении таких систем одной из главных проблем является обеспечение стабильного времени удовлетворения клиентских запросов в условиях загруженной сети. Во-первых, из-за высокой внешней загруженности сети часть серверов системы может быть просто недоступна. Во-вторых, из-за неравномерного доступа пользователей к информации, в некоторый момент времени может сложиться ситуация, при которой очень много клиентских запросов, будут направлены к одному и тому же информационному ресурсу, что приведет к существенной загрузке сети, вокруг серверов, содержащих интересующий ресурс. Это приводит к тому, что некоторые сервера системы становятся недоступными, хотя к другим доступ свободен.

Перечисленные проблемы создают ряд неудобств пользователям, которые фактически, не могут вовремя получить необходимую информацию. Наглядными примерами проблем, с которыми сталкиваются пользователи, может служить сайт посвященный олимпийским играм в Сиднее, который во время проведения олимпийских игр, одновременно посещали несколько миллионов пользователей, что сильно сказывалось на скорости получения данных каждым из пользователей, а во время некоторых соревнований, сервер был недоступен из-за перегрузки.

Среди существующих технологий, обеспечивающих доступность Internet-ресурсов, следует отметить технологию SLB (Server Load Balancing) [1], задача которой обеспечить масштабируемость серверов, предоставляющих различные Internet-сервисы (Web,DNS,FTP). Суть его заключается в том, что несколько тесно связанных серверов объединяются в один виртуальный сервер и клиентские запросы распределяются между ними. Такой подход предполагает, что клиенты всегда обращаются к 1P-адресу

виртуального сервера, а функция SLB выбирает реальный сервер для обслуживания запроса клиента.

Такая технология, однако, имеет ряд недостатков. Во-первых, она предполагает зеркалирование данных каждого сервера, включенного в группу. Во-вторых, технология обеспечивает масштабируемость только по скорости обработки запросов и не обеспечивает масштабируемость по скорости передачи данных пользователю. Если каналы связи из-за загруженности сети не позволяют передавать объем данных, необходимый для удовлетворения всех клиентских запросов, то даже если виртуальный сервер найдет все необходимые данные, он не сможет в необходимое время передать их клиенту.

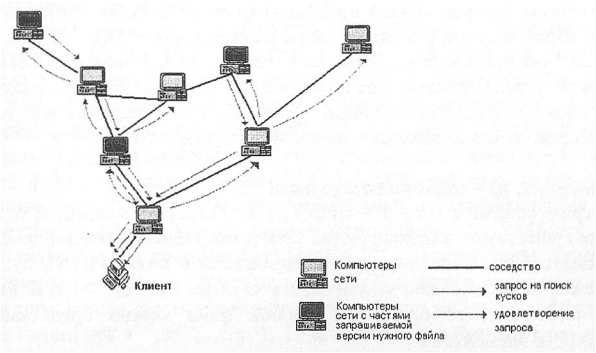
Кроме вышеописанной технологии SLB, в существующих системах часто применяется технология "зеркала". Создаются несколько идентичных серверов, дублирующих друг друга. Для доступа к информации, пользователю достаточно подсоединится к "ближайшему" с точки зрения доступности серверу. Недостатками данной технологии является необходимость многократного дублирования хранилищ данных, что требует существенных ресурсов для построения больших систем, и потребность пользователю самостоятельно определять "ближайший" к себе сервер.

В данной статье рассматривается метод распределенного хранения данных, который может быть использован для обеспечения доступности информационной системы в условиях неравномерной загруженности сети и неравномерного распределения в ней информации. Метод основан, на представлении файла в виде *(N, к)* - пороговой схемы, которая позволяет представить файл в виде *n* частей *(n < N),* таким образом, что пользователю достаточно иметь доступ к любым к из них, чтобы собрать весь файл целиком. *(N, к)* - пороговые схемы были разработаны независимо Эди Шамиром [2] и Бобом Блэкли [3]. Задачей этих схем являлось обеспечение безопасного хранения секретной информации. Никакие *k*-\ из существующих частей не представляют абсолютно никакой информации (в смысле теории информации Шеннона) относительно исходных данных. Рассматриваемая ниже *(N, к)* - пороговая схема отличается от алгебраической схемы Шамира и геометрической схемы Блэкли. Ее задача обеспечить минимальное увеличение размера частей, необходимых для сборки файла по сравнению с размерам исходного файла. Кроме этого число *N* должно быть достаточно велико, чтобы обеспечивать доступность ресурса при интенсивном его использовании.

Модель распределенного представления данных

Для чего хранить файл в виде (N, к)-пороговой схемы

Каждый файл предлагается хранить в виде некоторого набора частей, таким образом, из любых *к* частей можно полностью собрать файл. Размер каждой части ~ *Mk* от размера хранящегося файла. Количество частей файла - *n,* содержащихся в системе может меняться в зависимости от ее конфигурации, от количества компьютеров в ней, интенсивности работы с данным файлом и т.д. Причем, при изменении (увеличении) числа *n,* уже существующие части никак не изменяются, то есть новые части строятся таким образом, чтобы по-прежнему выполнялось пороговой схемы.



Для того, чтобы получить файл, пользователю необходимо получить любые к частей из существующих в данный момент в системе, из которых он сможет восстановить исходный файл. Общий объем полученной информации будет равен объему файла, тем самым общая нагрузка на сеть не будет увеличена.

На рисунке показано удовлетворение пользовательского запроса на чтение файла при $k=3$. Для удовлетворения запроса пользователь будет получать k "ближайших" с точки зрения загрузки в данный момент сети и пропускной способности каналов связи, что обеспечивает минимальное время удовлетворения запросов. Запросы на поиск кусков в системе служат для мониторинга интенсивности использования данного файла. Для этого каждый из серверов хранит историю запросов прошедших через него.

Такая модель позволяет динамически контролировать избыточность данных в системе и поддерживать ее для каждого файла на уровне, необходимом для равномерной загрузки сети и стабильного времени доступа к нему при заданной потребности пользователей. То есть при росте пользовательских запросов к данному файлу в некоторой области сети, модель представления данных позволяет системе создавать новые части данного файла, размещая их как раз на компьютерах, находящихся в той области сети, где спрос на файл возрос, тем самым разгружая ту часть сети, где спрос на файл невысок и увеличивая скорость доступа к файлу. Если потребность в некотором файле будет высока по всей сети, то части этого файла создадуться системой в достаточном количестве по всей сети, тем самым, распределяя нагрузку в ней. И наоборот, если спрос на файл будет уменьшаться, лишние (неиспользуемые) его части будут постепенно удаляться системой.

Предлагаемый метод будет позволять во-первых, динамически контролировать избыточность данных в системе на уровне конкретного файла, в зависимости от интенсивности его использования в данный момент, во-вторых, избыточные данные будут не статически располагаться на конкретных серверах, а динамически изменять свое расположение во времени, для того чтобы находится максимально близко с точки зрения скорости каналов к месту их использования. Таким образом, для того чтобы получить данные, пользователю будет достаточно получить их "ближайшие" копии. "Близость" одного сервера к другому будет определяться скоростью выполнения операции ping. При увеличении интенсивности работы с конкретным данными в некоторой области сети, в ней будут создаваться дополнительные их копии так, чтобы максимально разгрузить каналы связи.

Число k является фиксированным для каждого конкретного файла и зависит от его размера. При использовании протокола TCP/IP [3], который де факто является основным в существующих сетях, скорость передачи данных максимальна, когда их

размер соответствует размеру пакета протокола (message transfer unit), что составляет ~ 1 Kb. Таким образом, размер каждой части должен быть ~ 1Kb. То есть для файла размера *S* число *к* определяется по формуле $к = SZS_v$, где S_p - размер пакета протокола TCP/IP. Средний размер Internet ресурсов ~ 5-7 Kb. В следующих разделах будет показано, что сложность алгоритма сборки файла пропорциональна к и размеру файла, поэтому при исследовании производительности мы будем считать к = 5.

Описание (N, к) - пороговой схемы

Рассмотрим конечное поле P = GF(A) (поле Галуа, состоящее из *N* элементов). Конечные поля существуют для количества элементов равного степени простого числа. [5, 6] (Как вводятся операции сложения и умножения в полях $GF(A^r)$, будет описано ниже). Все элементы поля можно занумеровать числами от 0 до *N-l.* Так как файл это последовательность бит, то при $N = 2"$ любой файл можно представить в виде последовательности элементов P.

Рассмотрим теперь £-мерное пространство векторов L над полем P. Каждый элемент этого пространства - это к элементов поля P. Теперь каждый файл можно представить в виде последовательности векторов из L. (Если количество бит в файле не кратно kn, то дополняем его нулевыми битами до ближайшего числа, делящегося на kn). Пусть есть файл f, который представляет собой последовательность векторов $I_1, 1_2, \ldots, I_m$ из L. И пусть есть набор E из n векторов из L: $E = \{e_ь e_2, \ldots, e_{,,}\}$. Причем этот набор такой, что любые к из них образуют базис в L. Тогда каждому вектору e* из E можно поставить в соответствие часть:

$(I_1, e0, (1_2, eO, \cdot\cdot, (lm, Єї)$, где (Ij, eO — скалярное произведение векторов Ij и ej. Скалярное произведение любых двух векторов из L есть элемент из P. Поэтому размер такого куска равен *n* бит. Размер же исходного файла равен *nrkn* бит. Т. е. размер каждой из полученных частей равен mn, то есть *Mk* от размера самого файла.

Пусть у нас есть теперь произвольные k частей из построенного набора. Обозначим вектора, которым соответствуют части, через $S_1, S_2, \ldots$ st, а сами части через $E_1, E_2, \ldots E_k$. По условию набора E, эти вектора образуют базис в L. Поэтому матрица A размером kx к, строки которой - вектора $S_1, ε_2, \ldots, S_k$, имеет обратную, которую мы обозначим через A'". Тогда Al, - столбец, состоящий из i-тых элементов всех кусков. Обозначим такой столбец через Sj. Т. е. для любого г'с $[1, m\rangle$. $I_i = A^{-1}S_i$. Таким образом, мы можем из этих частей восстановить исходный файл. К каждой части должна быть приложена информация о порождающем ее векторе и исходная длина файла, для того, чтобы отбросить потом добавленные нулевые байты.

Следует отметить, что при таком способе представления данных, для получения фрагмента файла I_i нет необходимости прибегать к сборке всего файла. Мы можем собрать только необходимый фрагмент: $\mathbf{l_i} = A^{\wedge'} S_i$. Размер $\mathbf{l_i}$ составляет kn бит, что при *k~5* и о = 8,16,32 составляет несколько байт. Таким образом, существует возможность неполного восстановления файла, обусловленная запросом чтения/записи конкретного участка файла (диапазона байт). Это может оказаться существенной для асинхронной работы различных процессов с файлами.

Алгоритм построения набора векторов, любые к из которого образуют базис, достаточно прост. Для каждого ненулевого элемента р поля P строим вектор $(1, p, p^2 \ldots p^к)$. Таким образом, мы можем построить *N-I* вектор. Любые *к* из них образуют базис — детерминант образуемой ими матрицы — определитель Вандермонда, который равен нулю только, когда некоторые из знаменателей прогрессий совпадают. Указанным способом мы легко строим набор, из *N*-1 векторов, любые к из которых образуют базис. Для того, чтобы описать вектор, по которому построена часть, достаточно задать знаменатель прогрессии, который занимает n бит.

Реализации модели

В данной секции рассматривается вариант реализации алгоритмов сборки/разборки в GF(A'). *(N* = 2", где *n*= 16, в этом случае количество возможных частей файла в системе ограничено числом 65535) на процессоре Celeron 450. Задача состоит в том, чтобы добиться результатов производительности алгоритма сборки файлов порядка скорости передачи данных в той среде, в которой функционирует данная модель. В Internet максимальная скорость передачи данных ~ 1.5 МБ/сек, кроме того сборка файла не должна отнимать более 50% времени процессора (во время сборки система должна позволять принимать данные и оставлять время процессору на другие процессы), поэтому скорость сборки должна быть ~ 3 - 5 МБ/сек.

Стандартное представление поля GF(A' = 2") — многочлены степени не выше n-1, коэффициенты которых — элементы поля GF(2) остатков от деления на 2 (то есть 0 или 1). Для того, чтобы задать такой многочлен, нужно задать n коэффициентов (0 или 1 — то есть битов) при x^{n-1}, x^{n-2}, ..., x, 1. То есть каждый η-битный элемент задает такой многочлен. Сложение и умножение таких многочленов происходит по остатку при делении на произвольный неразложимый многочлен степени n над полем GF(2). Выбор такого многочлена *p(x)* обусловлен исключительно удобством реализации алгоритма Эвклида для вычисления остатка от деления на него. Таким образом, сложение многочленов - это просто операция XOR над соответствующими n-битными элементами. Умножение можно разложить на два этапа: последовательность сдвигов и XOR-ОВ (фактически обычное умножение в котором, сложение заменено операцией XOR) и поиск остатка от деления получившегося после первого этапа многочлена (в общем случае степени *2n-2)* на многочлен p(x). Второй этап также реализуется последовательностью сдвигов и XOR. Деление реализовывается как операцию, обратная умножению.

Реализация такого алгоритма для n = 16 на процессоре Celeron 450 при к = 5 показывает производительность 0.8 МБ/сек. что является достаточно низким показателем для промышленного использования модели (реализация допускает лишь ограниченное применение в Internet), поэтому автор предлагает другую реализацию операций умножения и деления, основанную на алгоритмах pre-calculated lookup tables [7].

Используем, тот факт, что в *GF(N)* существует генератор *p,* то есть все элементы являются степенью *pcGF(N).* Таким образом, GF(AO={0,p,p^2,...,$p^{N\sim2}$,$p^{N"'}$ =1}. В GF(2^{16}) генератором является число x+1. Вычислим все пары *(a,i)*, такие что $a=p^x$. Сохраним эти пары в две таблицы: Iog—отсортированную по a, и alog — отсортированную по *i.* То есть : log ^>'] = *i,* alog [г] =*p'.* Размер каждой из двух таблиц равен 128 КБ. Поэтому имеем для любых элементов *a, b* из GF(2^{16}):

$ab = \text{alog}[(\log[a] + \log[b]) \bmod (2^{16} - 1)]$ и $a/b = \text{alog}[(\log[a] - \log[b]) \bmod (2^{16} - 1)]$,
причем,
$(\log[a] + \log[b]) \bmod (2^{16} - 1) = \log[a] + \log[b]$, если $\log[a] + \log[b] < 2^{16} - 1$,
и
$(\log[a] + \log[b]) \bmod (2^{16} - 1) = \log[a] + \log[b] - (2^{16} - 1)$, если $\log[a] + \log[b] >= 2^{16} - 1$,
а также
$(\log[a] - \log[b]) \bmod (2^{16} - 1) = \log[a] - \log[b]$, если $\log[a] - \log[b] >= 0$,
и
$(\log[a] - \log[b]) \bmod (2^{16} - 1) = \log[a] - \log[b] + (2^{16} - 1)$, если $\log[a] - \log[b] < 0$.

Поэтому операция mod(2 -1) заменяется на операцию сравнения и сложения (вычитания). Таким образом, умножение сводится к сложению, а деление к вычитанию. Такая реализация на процессоре Celeron 450 и k = 5 дает скорость сборки ~ 10 МБ/сек, что достаточно для работы в Internet. Результаты можно улучшить путем использования MMX технологии для одновременного вычисления нескольких арифметических операций. Предварительные оценки показали, что возможно

увеличение скорости работы от 2 до 7 раз (особенно при использовании систем с большим вторичным cache процессора — более 256K).

Анализ производительности модели

Проведем теперь сравнительный анализ скорости получения данных в стандартных условиях в сети Intemet и в предлагаемой модели, при условии, что сеть существенно загружена.

Рассмотрим сначала ситуацию со стандартной скоростью. Обозначим через V скорость передачи данных в сети. Пусть U - скорость канала между клиентским компьютером и сервером, через который он осуществляет соединение с Internet (см. рис. 2). Будем рассматривать ситуацию, когда каждый канал в сети используется одновременно N пользователями, это означает, скорость канала в сети понижается до WN. Будем считать, что U " WN, поэтому скорость получения данных V_3 в нагруженной сети 6yflerV, = WN.

Исследуем теперь нашу модель в тех же условиях (см. рис. 3). Скорость канала в сети также понижается до WN, однако k частей файла одновременно поступают на хост клиентского компьютера. Скорость получения данных на хост, таким образом, будет V_x = кУЛЧ. Так как мы предполагаем, что U " WN, то скорость получения данных определяется скоростью получения данных хостом и скоростью сборки файла V_c.

Итак, скорость равна:

$V'_э = 1/(1/V_x+1/V_c) = 1/(N/kV+1/V_c) = kVV_c / (kV+NV_c)$.

Пусть $V_c = CtV$, тогда V_3 =akV/(k+aN)=kWN=kV>, так как aN"k. То есть при существенной загрузке сети, в предлагаемой модели скорость получения данных в k раз

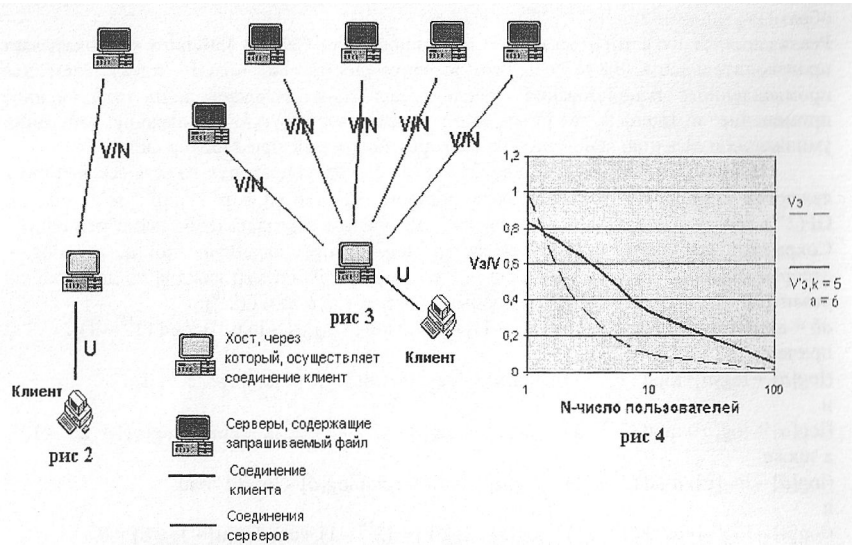


рис 2

рис 3

рис 4

выше, чем в стандартных условиях. На рис. 4 приведены графики зависимости производительностей двух систем от загрузки сети.

В предлагаемой реализации k = 5 и α = 6. Таким образом, в случае существенной загрузки скорость получения данных будет в 5 раз выше, чем в обычных условиях.

Выводы

Предложена оригинальная модель распределенного хранения данных, позволяющая динамически контролировать степень избыточности информации, в зависимости от потребностей пользователей. Данные представляются в виде (yV *,k)* - пороговой схемы, построенной на теории конечных полей Галуа - GF(A0- Степень избыточности данных в рамках модели ограничена числом *N* - мощностью поля GF. Рассмотрена реализация модели при $N=2^{16}$, Вычисления в GF выполняются на основе алгоритмов pre-calculated lookup table. Исследована эффективность использования данной реализации.

Модель представляет практический интерес для применения в управлении большими информационными ресурсами в сети Internet. Она позволяет динамически равномерно распределять нагрузку передачи данных между компьютерами сети, обеспечивать надежность работы системы и стабильное время доступа к информации.

Литература

1. *H. Kameda, J. Li, C. Kim, Y. Zhang.* Optimal Load Balancing in Distributed Computer Systems (Telecommunication Networks and Computer Systems). — Berlin: Springer Verlag. 1996. — 251 p.
2. A. Shamir. How to share a secret. - Communications of the ACM // vol. 24. 1979. — pp. 612 - 613.
3. G. Blakley. Safeguarding cryptographic keys. ~ Proceeding of AFIPS // vol. 48. 1979. — pp. 313-317.
4. *W. RichardStevens.* TCP/IP Illustrated vol. 1-3. — Addison: Wesley Pub Co. 1994. — 2078 p.
5. *Б. Ван дер Варден.* Алгебра. — М.: Наука. 1979. — 648 с.
6. *А. Курош.* Курс высшей алгебры. — М.: Наука. 1975. — 431 с.
7. *E. Win, A. Bosselaers, S. Vanderberghe, P. Gersem, J. Vandewalle.* A Fast Software Implementation for Arithmetic Operations in GF(2"). / Katholieke Universiteit Leuven. 1997. — 12 p.