Оптимизированная загрузка файлов произвольного доступа

А. А. Царьков

***Аннотация*—Файлы, доступные для загрузки по HTTP, составляют значительное количество общедоступной информации. Для получения доступа к ней пользователи вынуждены загружать файл целиком, что зачастую является неэффективным способом получения данных. Например, когда данные файла необходимо использовать незамедлительно или необходимо получение доступа лишь к части файла, обычная загрузка файла приводит к нежелательным простоям и расходованию ресурсов сети.**

В статье изложен способ оптимизированной загрузки файла по запросу, состоящий в загрузке именно тех частей файла, чтение которых было инициировано клиентским ПО. Для соблюдения баланса между скоростью исполнения очередного запроса чтения и суммарным уровнем сопутствующих накладных расходов, предлагается эффективный способ загрузки, на порядок снижающий максимальное время ожидания исполнения запроса чтения при среднем уровне накладных расходов 10%.

Изложен механизм выявления последовательных запросов чтения и предсказания будущих обращений к фрагментам файла, снижающий уровень накладных расходов в 2-10 раз для частичного чтения файла. Для этого предложена вероятностная модель определения размера фрагментов, доступ к которым вероятнее всего будет запрошен последовательно. Она основана на двухпараметрическом распределении Вейбулла и позволяет гибко настраивать ее параметры для эффективного применения с разными паттернами доступа.

***Ключевые слова*—readahead, определение последовательного чтения, оптимизированная загрузка файла, упреждающая загрузка.**

I. Введение

Сегодня все реже данные передаются клиенту с целью хранения и все чаще с целью использования «здесь и сейчас». Во многих ситуациях обычная загрузка файла бывает неэффективна с точки зрения доступа к его данным, что приводит к распространению сетевых протоколов доставки данных с целью их незамедлительного использования [1], [2]. Однако именно HTTP наиболее распространен на сегодняшний день.

Общая особенность загрузчиков общего назначения — последовательная загрузка и сохранение фрагментов файла. Такой подход позволяет минимизировать накладные расходы на отправку запросов к серверу, однако, можно привести условия, при которых такой способ загрузки не является эффективным.

Частичный доступ. Если пользователю необходима лишь часть данных файла, много меньшая его фактического размера, то на загрузку невостребованной части файла впустую расходуются ресурсы сети.

Произвольный доступ. При загрузке файла произвольного доступа загруженные блоки данных остаются недоступны в общем случае до окончания загрузки файла целиком, так как для их интерпретации могут быть необходимы данные из конца файла.

Имеют место ситуации, в которых приложение, инициирующее чтение файла не последовательными фрагментами, ожидает загрузки файла целиком, хотя могло бы получить к нему доступ без задержек, если скорость загрузки превосходит среднюю скорость чтения загружаемого файла приложением.

В настоящее время из-за экспоненциального роста количества хранимой информации, опережающего развитие технологий предоставления доступа к хранилищам, велик разрыв между производительностью оперативной и внешней памяти. С тем чтобы нивелировать этот разрыв, применяются различные приемы кэширования и упреждающего чтения [3]-[6]. В то время как обычное кэширование выполняется по факту обращения к некоторой области внешней памяти, упреждающее чтение может происходить асинхронно, его задача — поместить фрагмент внешней памяти в кэш до инициации самого обращения к нему, сокращая простой внешней памяти. Чтобы определить, к какому фрагменту вероятнее всего будет инициировано следующее обращение, используются методы предсказания, которые можно разделить на два класса: эвристические методы и методы поиска ассоциативных правил. В то время как задача поиска ассоциативных правил имеет ограниченную область применения, эвристические методы предсказания применимы почти всегда.

В статье изложен способ загрузки файла по запросу по HTTP, состоящий в загрузке именно тех частей файла, чтение которых было инициировано клиентскими приложениями. Для соблюдения баланса между скоростью исполнения очередного запроса чтения и суммарным уровнем сопутствующих накладных расходов, предлагается эффективный способ загрузки, а также механизм выявления последовательных запросов чтения и предсказания будущих обращений к фрагментам файла.

II. Определения

Исходная последовательность запросов чтения файла $\{r_n\}$ инициируется пользовательским приложением. Каждый запрос в последовательности характеризуется отступом внутри файла p_i байт, начиная с которого ОС следует считать и поместить в предоставленный буфер

Статья получена 29 мая 2018.
А. А. Царьков (e-mail: alex.alex.tsarkov@gmail.com)

фрагмент файла размером s_i байт.

$$\{r_n\}_{n=1}^{N} = \{(p_1, s_1), (p_2, s_2), (p_3, s_3), \ldots, (p_N, s_N)\}$$

Если последовательность запросов чтения содержит обращения к фрагментам файла, покрывающим весь файл, то такая последовательность называется полной последовательностью запросов чтения. Если последовательность запросов содержит обращения к фрагментам, лишь частично покрывающим файл, то такая последовательность запросов чтения называется частичной.

То, какую последовательность запросов чтения инициирует приложение, определяет его паттерн доступа к файлу. В случае, если приложение последовательно запрашивает фрагменты файла, расположенные друг за другом, можно говорить о преобладании последовательного паттерна доступа. В случае, если запросы к последовательным фрагментам чередуются с запросами к фрагментам файла, расположенным в произвольном порядке, можно сделать вывод о преобладании частично-последовательного паттерна доступа. Если же выявить закономерность последовательности запросов к фрагментам файла без дополнительных знаний не представляется возможным, или инициируется доступ к фрагментам, расположенным в произвольных частях файла, то преобладает произвольный паттерн доступа.

Запросы чтения файлов последовательного доступа обычно имеют последовательный или частично-последовательный паттерн, а запросы чтения файлов произвольного доступа — частично-последовательный или произвольный паттерн. Однако, четко выявить соответствующий паттерн доступа возможно только постфактум, когда приложение завершило чтение файла. До завершения чтения можно лишь строить предположения о типе паттерна. Частично-последовательный паттерн доступа является характерным, например, для файлов-контейнеров (мультимедиа, архивы, установщики), которые чаще всего загружаются из сети.

Принимая одновременно запросы чтения от всех пользовательских приложений, операционная система принимает решение об агрегировании активных запросов в новую последовательность запросов чтения $\{r_m\}$ [4]. Результирующие последовательности, генерируемые операционной системой, называются агрегированными последовательностями запросов чтения.

Если текущий запрос чтения фрагмента файла определяет, какой фрагмент будет загружен, то такой способ загрузки называется загрузкой по запросу. В случае, если загрузка по запросу инициирует сетевые запросы получения фрагментов файла, размер каждого из которых кратен фиксированному числу, этот способ загрузки называется равномерной загрузкой по запросу.

Для того чтобы сравнивать разные способы загрузки, определим показатель эффективности загрузки. Функция $U(\{r_n\})$ называется показателем недоступности последовательности запросов чтения $\{r_n\}$ и определяется следующим образом:

$$U(\{r_n\}_{n=1}^{N}) = \sum_{n=1}^{N} t_e(r_n)\, s_n.$$

Считается, что запросы чтения файла поступают непрерывно, без пауз между запросами, а выполнение запросов не занимает времени при условии локальной доступности запрашиваемого фрагмента. Каждому запросу $r=(p,s)$ к фрагменту файла, имеющему отступ p байт и размер s байт, ставится в соответствие время исполнения $t_e(r)$, прошедшее с момента инициации первого запроса к файлу до момента исполнения запроса r.

Для любой последовательности запросов доступа метод загрузки по запросу определяет последовательность $\{R_m\}$ сетевых запросов загрузки фрагментов файла. Функция $U(\{R_m\})$ называется показателем локальной недоступности файла:

$$U(\{R_m\}_{m=1}^{M}) = \sum_{m=1}^{M} t_e(R_m)\, s_m.$$

Пример. Пользовательское приложение инициирует произвольную последовательность запросов чтения файла $\{r_n\}$, в то время как файл последовательно загружается из сети. Диаграмма времени исполнения этих запросов представлена на рисунке 1.

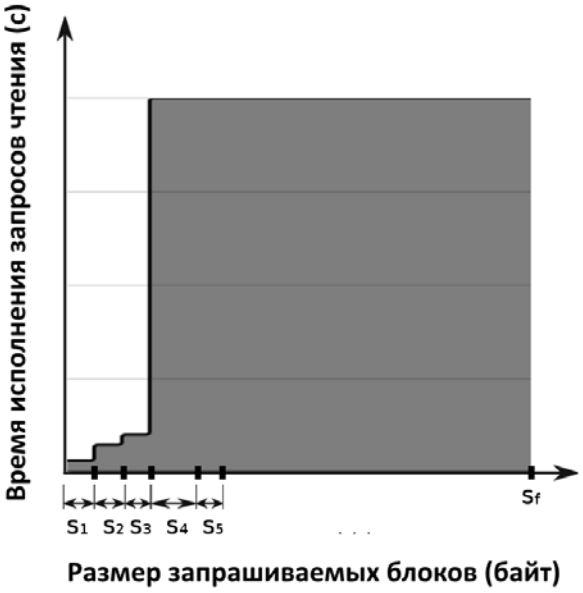

Рис. 1. Время исполнения $t_e(r_i)$ запросов $\{r_n\}$ с произвольным паттерном доступа.

Площадь под диаграммой соответствует значению показателя недоступности $U\{r_n\}$. Обращение r_4 к незагруженному концу файла обуславливает рост показателя недоступности. Время исполнения этого запроса можно оценить по формуле

$$t_e = t_r + \frac{s_f}{v_d},$$

где t_r — время ответа сервера (с), s_f — размер файла (байт), v_d — средняя скорость загрузки (байт/с).

III. Оптимизированная равномерная загрузка по запросу

Цель этого раздела состоит в описании

оптимизированного метода равномерной загрузки файла по запросу, при котором:

◦ На вход поступает URL файла и агрегированные последовательности запросов чтения (например, через FUSE API);

◦ Файл загружается по HTTP и сохраняется на диске;

◦ Для любой полной последовательности запросов чтения показатель локальной недоступности файла является минимальным из всех методов равномерной загрузки по запросу.

Пусть X — множество всех полных последовательностей запросов чтения $\{r_n\}$ файла размера s_f, Y — множество всех непустых последовательностей равномерных сетевых запросов загрузки $\{R_m\}$. Каждый метод равномерной загрузки по запросу определяется отображением $f: X \to Y$, а множество всех таких методов определяется множеством $F = \{f \mid f: X \to Y\}$. Задачу оптимизации можно сформулировать так:

Найти $f: X \to Y$ такую, что

$$\forall \{r_n\}_{n=1}^{N} \in X \qquad U(\{R_m\}) \to \min_{f \in F}, \qquad f(\{r_n\}) = \{R_m\}_{m=1}^{M} \in Y.$$

Размер файла s_f можно считать кратным размеру блока s_b, не ограничивая общности (формально, файл можно дополнить нулями до кратности s_b). Минимизируем показатель локальной недоступности файла:

$$U(\{R_m\}) = \sum_{m=1}^{M} t_e(R_m)\, s_b = \sum_{m=1}^{M} \left(t_e(R_{m-1}) + t_r + \frac{s_b}{v_d} \right) s_b =$$

$$= \frac{M}{2} \left(t_r + \frac{s_b}{v_d} + M \left(t_r + \frac{s_b}{v_d} \right) \right) s_b = \{M = \frac{s_f}{s_b}\} =$$

$$= \frac{s_f}{2} \left(t_r + \frac{s_b}{v_d} + \frac{s_f\, t_r}{s_b} + \frac{s_f}{v_d} \right)$$

Выбор метода равномерной загрузки определяет отображение f и последовательность $\{R_m\}$. Характеристикой метода является размер блока s_b и количество запросов $M = s_f / s_b$. Тогда минимум $U(\{R_m\})$ легко найти, продифференцировав по s_b:

$$\frac{dU(\{R_m\})}{ds_b} = \frac{s_f}{2} \frac{s_b^2 - s_f\, v_d\, t_r}{v_d\, s_b^2} = 0 \Leftrightarrow s_b = \sqrt{s_f\, v_d\, t_r} \quad (1)$$

Нетрудно убедиться, что найденное значение s_b минимизирует показатель локальной недоступности файла. Эффективность найденного метода состоит в том, что при нем достигается баланс между временем ожидания исполнения очередного запроса чтения и общим уровнем накладных расходов.

Пример. Чтение ZIP-архива размера *100 МБ* во время загрузки при скорости передачи данных *1 Мб/с* и времени отклика сервера *1 с*. Оптимальный размер блока — *3620 КБ*, накладные расходы загрузки по запросу — *7.3% (59 секунд)*. На рисунке 2 видно, что оптимизированная загрузка позволила снизить показатель недоступности по сравнению с последовательной загрузкой.

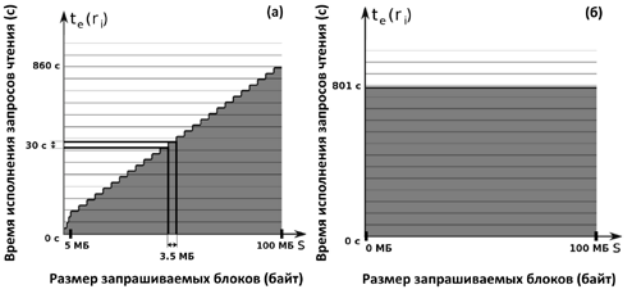

Рис. 2. Время исполнения запросов чтения при оптимизированной загрузке по запросу (а) и при последовательной загрузке (б).

IV. Упреждающая загрузка

Время ожидания исполнения запросов чтения при загрузке по запросу можно уменьшить, отказавшись от загрузки еще не запрошенных фрагментов файлов. Но тогда последовательности из большого количества запросов к небольшим фрагментам файла становятся причиной резкого увеличения накладных расходов на частую отправку запросов к серверу. Меньшее количество запросов приводят к чтению загруженных фрагментов с диска, и большее — к инициации загрузки нового фрагмента. Цель этого раздела состоит в описании метода загрузки по запросу, при котором для частичных последовательностей запросов чтения, имеющих частично-последовательный паттерн, накладные расходы ниже, чем при использовании равномерной загрузки по запросу.

Идея метода состоит в том, чтобы оценивать вероятность обращения к каждому фрагменту файла в будущем и принимать решение об упреждающей загрузке на основании этой оценки.

Пусть s — случайная величина, выражающая точный размер фрагмента файла, последовательное чтение которого будет инициировано после текущего запрошенного в r_i фрагмента (p_i, s_i). Не ограничивая общности можно считать s непрерывной случайной величиной.

$$s \in [0;\ +\infty)$$

Пусть $P(s_{ra})$ – вероятность того, что после упреждающего чтения s_{ra} байт, следующих за текущим запрошенным фрагментом, более s_{ra} байт будут действительно запрошены.

$$P(s_{ra}) = P(s > s_{ra})$$

Для того чтобы смоделировать частично-последовательный паттерн доступа, $P(s_{ra})$ должна отражать его эмпирические закономерности. Основное предположение о характере частично-последовательного паттерна состоит в том, что если последовательно прочитано n байт, то скорее всего следующие n байт также будут прочитаны. Для этого $P(s_{ra})$ должна задавать распределение случайной величины, обладающее следующими свойствами:

1. Если текущий запрос является произвольным, математическое ожидание s равно нулю;
2. Если текущий запрос является последовательным, математическое ожидание s равно размеру запрошенного перед ним

фрагмента.

К сожалению, в общем случае невозможно точно классифицировать текущий запрос. Отнесение его к произвольному или последовательному типу возможно только с некоторой вероятностью. В таком случае свойства *s* должны быть переформулированы так:

1. Если текущий запрос *вероятнее всего* является произвольным, математическое ожидание *s близко* к нулю;
2. Если текущий запрос *вероятнее всего* является последовательным, математическое ожидание *s близко* к размеру запрошенного перед ним фрагмента.

Для того чтобы состоятельно определить условия близости рассматриваемых величин для широкого класса всевозможных последовательностей доступа, необходимо проведение дополнительных испытаний и исследования. В этой статье предлагается лишь общий подход к моделированию частично-последовательного паттерна доступа, и предлагаемые значения величин выбраны для наглядности и демонстрации работоспособности модели.

Функция $P(s_{ra})$ должна обладать свойствами:

1. Для произвольных запросов $P(s_{ra})$ имеет максимум в нуле и экспоненциально убывает с ростом s_{ra}, стремясь к нулю;
2. Для последовательных запросов $P(s_{ra})$ имеет максимум в нуле, медленно убывает до точки математического ожидания, а после нее убывает экспоненциально.

Свойство 1 объясняется тем, что в случае произвольного запроса вероятность обращения к следующим байтам мала. При этом возможно, что текущий запрос на деле является началом новой последовательной серии запросов, в следствие чего вероятность обращения к нескольким ближайшим байтам выше, чем вероятность обращения к ближайшему фрагменту большего размера. Свойство 2 объясняется тем, что в случае последовательного запроса следующие байты практически наверняка будут запрошены в будущем. Это свойство продиктовано основным предположением о характере частично-последовательного паттерна доступа.

Через $P(s_{ra})$ выразим функцию распределения $F_s(s_{ra})$ случайной величины *s*.

$$F_s(s_{ra}) = P(s \le s_{ra}) = 1 - P(s_{ra})$$

Для классификации текущего запроса предлагается использовать информацию о размере последовательно прочитанных фрагментов файла, а также учитывать размеры фрагмента, запрошенного в предыдущем обращении, и текущего запрашиваемого фрагмента. Для текущего запроса $r_i = (p_i, s_i)$ определен s_c — размер фрагмента, последовательно прочитанного (одним или несколькими запросами) до p_i.

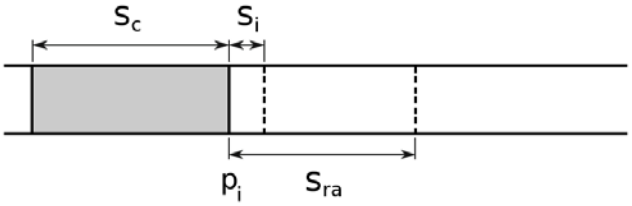

Рис. 3. Запрос $r_i = (p_i, s_i)$ к фрагменту, следующему за последовательно считанными s_c байт.

Пусть s_{min} и s_{max} — минимальный и максимальный размер запрашиваемого фрагмента соответственно. Например, система ввода-вывода Linux преобразует небольшие запросы в запросы к большим фрагментам, минимальный размер которых составляет размер страницы кэша (обычно 4 КБ) и разбивает большие запросы на запросы меньшего размера, определенного максимальным пороговым значением. Кроме того, Linux выполняет асинхронное упреждающее чтение, экспоненциально увеличивая размер каждого следующего считываемого фрагмента до порогового значения, если в исходной последовательности запросов выявлен последовательный паттерн. Поэтому s_{min} и s_{max} можно выбрать равными соответствующим значениям для данной файловой системы. Так, принимая на вход агрегированные запросы чтения, можно судить об исходной последовательности следующим образом:

1. Если для текущего запроса $s_i = s_{min}$, вероятнее всего он является произвольным;
2. Если для текущего запроса $s_{min} < s_i < s_{max}$, вероятно, что последовательный паттерн начинает преобладать в исходных запросах;
3. Если для текущего запроса $s_i = s_{max}$, вероятно устойчивое преобладание последовательного паттерна.

Для случайной величины *s* предлагается использовать двухпараметрическое распределение Вейбулла $W(k, \lambda)$ [7], [8] с плотностью вероятности $f_s(s_{ra})$, функцией распределения вероятности $F_s(s_{ra})$ и математическим ожиданием $M[s]$. Хотя обычно оно используется для моделирования времени наработки до отказа, свойства таких случайных величин аналогичны описанным для *s*. Обоснованием тому может служить тот факт, что *s*, вообще говоря, определяет также и время, в течении которого будут инициированы и исполнены новые последовательные запросы до того, как весь последовательный фрагмент данных не будет прочитан. В такой трактовке конец чтения последовательного фрагмента эквивалентен отказу.

$$s \in [0;+\infty), \quad k > 0, \quad \lambda > 0,$$

$$f_s(s_{ra}) = \frac{k}{\lambda}\left(\frac{s_{ra}}{\lambda}\right)^{k-1} e^{-\left(\frac{s_{ra}}{\lambda}\right)^k},$$

$$F_s(s_{ra}) = 1 - e^{-\left(\frac{s_{ra}}{\lambda}\right)^k},$$

$$M[s] = \lambda\, \Gamma\left(1 + \frac{1}{k}\right).$$

Коэффициент масштаба *k* в таком случае

определяется характером выявленного паттерна: при произвольном паттерне доступа $k < 1$, при последовательном — остальные значениях k.

Коэффициент формы λ определяется размером фрагмента, доступ которому вероятнее всего будет последовательно запрошен после текущего запроса: при произвольном паттерне доступа $\lambda < 1$, и значения, пропорциональные s_c, при последовательном.

Математическое ожидание случайной величины s определяет размер фрагмента, упреждающая загрузка которого эффективно снижает накладные расходы, сохраняя приемлемый уровень производительности ввода-вывода. Чтобы сократить задержки выполнения запросов чтения, в качестве максимального размера загружаемого фрагмента предлагается выбирать оптимальный размер блока s_b, определенный в (1); в качестве минимального – размер фрагмента, загрузка которого занимает столько же времени, сколько и одно обращение к серверу.

Параметры распределения предлагается определить как $\lambda = \lambda(s_i, s_c, s_{min}, s_{max})$, $k = k(s_i, s_{min}, s_{max})$:

$$\lambda = \begin{cases} \varepsilon, & \text{если} \quad s_i = s_{min}, \\ \left(\frac{s_i}{s_{max}}\right)^5 s_c, & \text{если} \quad s_i > s_{min}, \end{cases}$$

$$k = \frac{1.5\, s_i + 0.5\, s_{max} - 2\, s_{min}}{s_{max} - s_{min}},$$

где $0 < \varepsilon < 1$. Это не единственный способ выражения параметров распределения. Эмпирически можно построить выражения, использование которых более эффективно для большего класса последовательностей чтения.

Пример 1. Три серии последовательных запросов чтения файла (размер *1 ГБ*, скорость *10 Мб/с*, время ответа сервера *0.5 с*, минимальный размер блока *640 КБ*, максимальный размер блока *25904 КБ*, общий размер запрошенных фрагментов *220 МБ*). При выявлении последовательного паттерна доступа выполняется упреждающая загрузка фрагментов, часть которых остается невостребованной, если серия последовательных запросов завершается раньше. В этом примере чтение *19 МБ* из загруженных *239 МБ* не было инициировано, что соответствует *15* секундам, затраченным на накладные расходы. При использовании равномерной загрузки по запросу с оптимальным размером блока *25904 КБ* невостребованными остаются *58.2 МБ*, загружаемые лишние *46.6* секунд. Таким образом, в этом примере уровень накладных расходов при загрузке по запросу с определением частично-последовательного паттерна доступа составляет *7.9%*, при использовании оптимизированной загрузки — *24.3%*, при использовании загрузчиков общего назначения — *328%*. На рисунке 4 фрагменты, запрошенные последовательно, объединены.

Рис. 4. Три серии последовательных запросов чтения.

Пример 2. Чередование последовательных и произвольных запросов чтения файла (размер *1 ГБ*, скорость *10 Мб/с*, время ответа сервера *0.5 с*, минимальный размер блока *640 КБ*, максимальный размер блока *25904 КБ*, общий размер запрошенных фрагментов *65 МБ*). В этом примере после каждой последовательной серии запросов чтения следует запрос к произвольному фрагменту файла. Накладные расходы: *14.6% (9.6* секунд, *9.5 МБ)*. Накладные расходы при использовании равномерной загрузки по запросу с оптимальным размером блока *25904 КБ*: *172.4% (142* секунды, *177.1 МБ).*

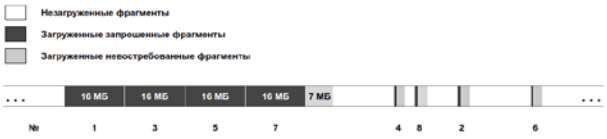

Рис. 5. Чередование последовательных серий запросов чтения с произвольными запросами.

Пример 3. Произвольная последовательность запросов чтения файла (размер *1 ГБ*, скорость *10 Мб/с*, время ответа сервера *0.5 с*, минимальный размер блока *640 КБ*, максимальный размер блока *25904 КБ*, общий размер запрошенных фрагментов *300 МБ*). В этом примере размер последовательно запрашиваемых фрагментов варьируется от *4 КБ* до *1 МБ*, запрашиваемые фрагменты распределены по всему файлу случайным образом. На рисунке 6 для наглядности показан отрезок файла, включающий лишь часть всех запрошенных фрагментов. Накладные расходы: *107%*. Накладные расходы при использовании как оптимизированной загрузки: *241%*.

Рис. 6. Произвольная последовательность чтения файла.

V. Заключение

Оптимизированный метод загрузки по запросу продемонстрировал снижение максимального времени исполнения запроса чтения до значения времени загрузки одного блока. Уровень накладных расходов сетевых ресурсов составил порядка 10%. Реализованный алгоритм выявления частично-последовательных паттернов доступа позволил сократить уровень накладных расходов частичной загрузки файла в среднем в 2-10 раз по сравнению с использованием равномерной загрузки по запросу.

Преимущество представленного подхода также состоит в том, что для его реализации выдвигается единственное требование к серверу – поддержка запросов *Range*.

Перспективным направлением дальнейших исследований является определение параметров вероятностной модели, которые позволят добиться наиболее эффективного определения частично-последовательных паттернов чтения для более широкого спектра файлов и последовательностей чтения. Для этого необходимо проведение испытаний на большой выборке целевых файлов, сбор статистики и модификация параметров вероятностного распределения.

Библиография

[1] Javvin Technologies, RTP, Network Protocols Handbook, 2005, ISBN 9780974094526.

[2] Parmar H., Ed. And Thornburgh M., Ed., Adobe Systems Incorporated, Adobe's Real Time Messaging Protocol, 2012.

[3] Gill Binny S., Modha Dharmendra S. SARC: Sequential Prefetching in Adaptive Replacement Cache.

[4] Wu, F., Xi, H., Li, J., & Zou, N. (2007). Linux readahead: less tricks for more. In . Proceedings of the Linux Symposium, 2, 273–284.

[5] Shriver, E., Small, C., & Smith, K. A. (1999). Why does file system prefetching work? In Pro-ceedings of the Annual Technical Conference on 1999 USENIX Annual Technical Conference, (pp. 71-84).

[6] Patterson III Russel. H. (1997). Informed Prefetching and Caching. PhD thesis, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA.

[7] Weibull, W. (1951), "A statistical distribution function of wide applicability", J. Appl. Mech.-Trans. ASME T. 18 (3): 293–297.

[8] Engineering statistics handbook. National Institute of Standards and Technology (2008).

[9] Bach, M. J., The Design of the UNIX Operating System, Englewood Cliffs, NJ: Prentice-Hall, 1986.

[10] Butt, A. R., Gniady, C., & Hu, Y. C. (2007). The Performance Impact of Kernel Prefetching on Buffer Cache Replacement Algorithms. IEEE Transactions on Computers, 56(7), 889–908. doi:10.1109/TC.2007.1029

[11] Kroeger, T. M. and Long D. D. E., Design and implementation of a predictive file prefetching algorithm, USENIX Annual Technical Conference, pp. 105-118, 2001.

[12] Liang, S., Jiang, S., & Zhang, X. (2007). STEP: Sequentiality and Thrashing Detection Based Prefetching to Improve Performance of Networked Storage Servers. 27th International Conference on Distributed Computing Systems (ICDCS'07), (p. 64).

[13] Megiddo, N. and Modha D. S., ARC: A self-tuning, low overhead replacement cache, Proc. FAST Conf., pp. 115-130, 2003.

[14] Megiddo, N. and Modha D. S., Outperforming LRU with an adaptive replacement cache algorithm, IEEE Computer, vol. 37, no. 4, pp. 58-65, 2004.

[15] Sagias, Nikos C. & Karagiannidis, George K. (2005), "Gaussian class multivariate Weibull distributions: theory and applications in fading channels", Institute of Electrical and Electronics Engineers. Transactions on Information Theory T. 51 (10): 3608–3619, ISSN 0018-9448, doi:10.1109/TIT.2005.855598

[16] Wu, F.: Sequential file prefetching in Linux. In: Wiseman, Y., Jiang, S. (eds.) Advanced Operating Systems and Kernel Applications: Techniques and Technologies, ch. 11, pp. 217–236. IGI Global (2009)

Optimized Random Access File Downloading

A. Tsarkov

***Abstract*—Significant amount of publicly availiable information is files available for download via HTTP. To access it, users are forced to download the entire file, which is often an inefficient way to retrieve data. For example, when you need to use file data immediately or you need to access only a portion of the file, normal file downloading results in undesirable overhead and network resource consumption.**

In this article a method for optimizing the on-demand download of a file is described. It is based on downloading of exactly those parts of the file that are currenly being read by a client software. To maintain a balance between the speed of execution of the next read request and the total level of associated overhead, an effective download method is proposed. It reduces the maximum waiting time of the read request execution of by an order of magnitude with an average overhead rate of 10%.

The mechanism of sequential read requests detection and predicting future calls to fragments of the file is described, which reduces the level of overhead costs by 2-10 times for partial file reading. To this end, a probabilistic model for determining the size of fragments which are most likely to be read sequentially is proposed. It is based on the two-parameter Weibull distribution and allows flexible configuration of its parameters for effective use with different access patterns.

***Keywords*—optimized file downloading, preemptive downloading, readahead, sequential read detection.**