УДК 004.75

Задача оптимизации размещения данных в распределённых системах

А. В. Жипа

Кафедра информационных технологий
Российский университет дружбы народов
ул. Миклухо-Маклая, д. 6, Москва, Россия, 117198

Эффективность работы распределённых вычислительных систем основывается на способе распределения потоков вычислительных задач и данных относительно ограниченного количества вычислительных ресурсов. Из-за постоянного увеличения объёма данных таким системам необходимо решать вопрос их хранения и обработки наиболее эффективным образом.

Между тем современные распределённые вычислительные системы уделяют все больше внимания таким своим характеристикам, как распределение вычислительной нагрузки, построение эффективной структуры хранилища данных, а также оптимальное использование вычислительных мощностей.

Оптимальное управление имеющимися у вычислительной системы ресурсами вынуждено балансировать между использованием ресурсов каждого отдельно взятого узла и потерей локальности хранения данных, связанной с их неизбежной фрагментацией.

В данной статье мы сформируем задачу оптимизации размещения данных путём максимизации локальности их хранения, а также покажем, что данная задача является NP-полной. Далее мы рассмотрим полиномиальный по времени алгоритм, дающий результат, отличающийся от оптимального на фиксированную константу.

Для доказательства эффективности предложенного алгоритма нами будет доказан ряд вспомогательных утверждений, а также подробно описана основная операция в работе алгоритма, за свою схожесть с процессом обмена участками хромосом в клетках названная кроссинговером.

Ключевые слова: фрагментация, распределённые вычислительные системы, NP-полные задачи, задача об упаковке в контейнеры, локальность хранения данных.

Введение

За последние годы область исследований распределённых вычислительных систем глобального масштаба заметно увеличилась. Например, такие децентрализованные системы хранения данных, как OceanStore [1], CFS [2], PAST [3] и IVY [4], предоставляют возможность постоянного хранения данных, используя распределённые, гетерогенные системы хранения данных.

В таких системах размер вычислительных задач и самих данных, как правило, превышает возможности одного отдельно взятого вычислительного узла, и система обязана распределять различные фрагменты информации по нескольким узлам одновременно. В результате такие системы вынуждены искать компромисс между использованием своих вычислительных ресурсов и фрагментацией данных по всей системе. Задачу поиска такого компромисса условно назовём задачей максимизации локальности хранения данных в распределённых системах.

Например, некоторые децентрализованные системы хранения данных оперируют объектами, обладающими большим размером. И несмотря на то, что объёмы современных накопителей данных достаточно велики, пользователи таких систем могут быть либо не в состоянии хранить такие объекты целиком или просто не иметь такой возможности в силу ограничений в передаче данных такого размера.

Очевидно, вычислительные системы обязаны разбивать большие файлы на несколько фрагментов, которые будут храниться в разных узлах системы. Для восстановления полной копии исходных данных требуется доступность каждого из этих узлов. Тем самым доступность данных обратно пропорциональна количеству фрагментов, из которых они состоят. Чтобы максимизировать доступность данных, следовательно, требуется минимизировать количество фрагментов данных.

Распределённые вычислительные системы сталкиваются с похожей проблемой. Для некоторого программного обеспечения требуются гарантии относительно возможности планировать выполнение вычислительных задач в фиксированный временной интервал. Ситуация усложняется тем, что одна или несколько таких задач может требовать наличие вычислительных ресурсов, которые не в состоянии предоставить ни один из имеющихся вычислительных узлов. Тем самым задачи должны быть распределены между узлами, чтобы обеспечить выполнение данного условия. Это, в свою очередь, приводит к росту издержек на передачу, координацию и объединение как самих данных, так и полученных результатов.

Эти проблемы, являющиеся частными случаями проблемы максимизации локальности хранения данных, представляют из себя специфические разновидности классической задачи об упаковке в контейнеры. Данная задача рассматривает проблему поиска варианта разбиения элементов разного размера по нескольким контейнерам, имеющим, в свою очередь, разную ёмкость. Каждый из таких элементов может обладать размером, не позволяющим разместить его целиком в одном из имеющихся контейнеров, тем самым необходимо разбивать их на фрагменты меньшего размера. Такое разбиение и приводит к потере локальности для данного элемента. В децентрализованных системах хранения данных это приводит к уменьшению доступности файлов, а также увеличению риска их потери. В распределённых вычислительных системах эта проблема приводит к увеличению издержек на передачу и хранение данных. В общем случае, чем больше фрагментов для элемента — тем хуже его локальность.

Одним из возможных решений данной проблемы может являться нахождение такого распределения данных, которое бы максимизировало среднюю (или общую) локальность данных, например, минимизируя среднее (или общее) количество фрагментов данных. Однако минимизация среднего случая не избавляет систему от риска получения наихудшего сценария размещения данных, приводя в серьёзной потере локальности данных в отдельно взятых случаях.

Наиболее выгодным решением данной проблемы кажется такое решение, при котором мы могли бы максимизировать минимальную локальность хранения данных для каждого отдельно взятого файла, или, другими словами, минимизировать максимальное количество фрагментов, на которые он разбит. Данная характеристика является очень важным фактором для распределённых вычислительных систем, поскольку они в таком случае могут обеспечить некоторую гарантию относительно локальности размещаемых данных. При этом в первом случае вычислительная система может гарантировать максимальную доступность файлов, а во втором — минимизацию затрат на их передачу.

Как уже было отмечено, проблема потери локальности влияет как на системы хранения данных, так и на распределённые вычислительные системы. Так что, без потери общности, в данной статье мы рассмотрим задачу максимизации локальности хранения данных. Будет показано, что определение оптимального решения для данной проблемы — NP-полная задача. Далее мы рассмотрим алгоритм, который за полиномиальное время позволяет найти решение данной задачи, которое отличается от оптимального на константу 2. С другими вариантами постановки задач о размещениях можно ознакомиться в [5–8].

1. Постановка задачи максимизации локальности хранения данных

Рассмотрим набор файлов $F = (F_1, F_2, \dots F_m)$ и набор узлов хранения данных $N = (N_1, N_2, \dots N_k)$ в некоторой распределённой вычислительной системе.

Введём следующую метрику $|F_i|$, обозначающую размер i-го файла, а также метрику $|N_j|$, обозначающую ёмкость j-го узла. Без потери общности предположим, что суммарный размер файлов равен общей ёмкости всех узлов распределённой вычислительной системы, т.е. $\sum\limits_i |F_i| = \sum\limits_j |N_j|$.

Определим компоновку K как назначение каждому файлу фиксированного набора узлов, в которых хранятся его фрагменты. В свою очередь, каждый узел может содержать фрагменты нескольких файлов.

Для любой компоновки K определим следующую функцию $\xi_K(F_i) = \xi(F_i)$, которая показывает количество узлов, в которых находятся фрагменты файла F_i. Аналогичным образом определим $\xi_K(F_i \cup F_j)$, как количество узлов, в которых находятся фрагменты файлов F_i и F_j.

Тем самым нам требуется найти такую компоновку, которая бы минимизировала максимальное количество фрагментов для каждого файлов, т.е. максимальное количество таких узлов. Определим данную задачу как

$$OPT(F, N) = \min_K \max_{1 \leqslant k \leqslant m} \xi_K(F_k),$$

где K — любая из всех возможных компоновок файлов F и узлов N.

Утверждение 1. $OPT(F, N)$ — NP-полная задача.

Доказательство. Для доказательства данного утверждения рассмотрим одну из известных NP-полных задач об упаковке в контейнеры (УК) (см. [9], а также [10, С. 223]). А именно: имеется множество S положительных чисел $s_1, s_2, \ldots, s_n$ с общей суммой равной 2σ. Требуется выяснить существует ли подмножество S с суммой элементов, равной σ?

Воспользуемся сводимостью по Карпу, которая формулируется следующим образом: задача разрешения P_1 полиномиально сводится (по Карпу) к задаче разрешения P_2, если существует полиномиально вычислимая функция f, перерабатывающая массивы входных данных C_1 для задачи P_1 в массивы входных данных $C_2 \equiv f(C)$ для задачи P_2 таким образом, что для любого C ответ в задаче P_1 совпадает с ответом задачи P_2 для входных данных $f(C)$.

Следовательно, если мы сведём более узкую формулировку нашей задачи максимизации локальности хранения данных к указанной задаче об упаковке в контейнеры УК, мы тем самым докажем её NP-полноту.

Пусть нам требуется выяснить, верно ли выражение $OPT(F, N) = 1$. Таким образом, при $F = \{F_1, \ldots, F_n\}$, где $|F_k| = s_k$, $1 \leqslant k \leqslant n$, и $N = \{N_1, N_2\}$, где $|N_1| = |N_2| = \sigma$, получаем, что $OPT(F, N) = 1$ тогда и только тогда, когда ответ на УК утвердительный. □

1.1. Основные утверждения

Так как поставленная задача максимизации локальности хранения данных является NP-полной, рассмотрим алгоритм, который решает данную задачу за полиномиальное время и даёт ответ, отличающийся от оптимального на фиксированную константу 2.

Для того, чтобы сформулировать данный алгоритм, нам понадобится доказать ряд вспомогательных утверждений. Для начала оценим нижнюю границу оптимального решения поставленной задачи.

Пусть рассматриваемые файлы отсортированы между собой в убывающем порядке по их размеру, а узлы по ёмкости. Перенумеруем их таким образом, что:

$$|F_1| \geqslant |F_2| \geqslant \ldots \geqslant |F_m|, \quad |N_1| \geqslant |N_2| \geqslant \ldots \geqslant |N_n|.$$

Далее *канонической* будем называть такую компоновку K, для которой отсортированному набору узлов ставятся в соответствие отсортированный набор файлов. К примеру, на рис. 1, фрагменты файла F_1 находятся в узлах N_1, N_2. При этом, ёмкость узла N_1 исчерпана полностью, тогда как остаток в узле N_2 используется для хранения фрагмента файла F_2 и так далее.

Таким образом, файл F_1 «попадает» в узлы N_1 и N_2 и, следовательно, $\xi(F_1)$ равно 2. Так как файлы F_1 и F_2 на двоих попадают в узлы N_1, N_2 и N_3, то получаем, что $\xi(F_1 \cup F_2)$ равно 3.

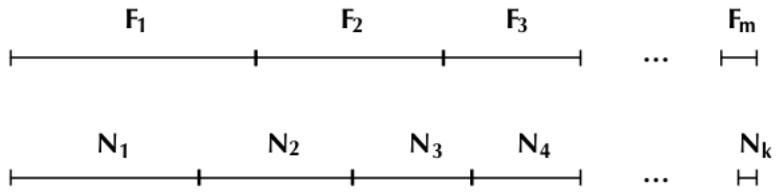

Рис. 1. Каноническая компоновка

Очевидно, что так как каждому узлу N_j, который содержит фрагмент файла F_i, соответствует некоторый интервал данного файла, то порядок этих узлов между собой не имеет значения.

Определим величину $\mu(k) := \lceil \frac{1}{k}\xi(F_1 \cup F_2 \cup \ldots \cup F_k) \rceil$, где $1 \leqslant k \leqslant m$, а также $\mu := \max_k \mu(k)$, где $\xi = \xi_K$, а K — каноническая компоновка. Например, для рис. 1, получаем следующие значения: $\xi(F_1) = 2$, $\xi(F_2) = 2$, $\xi(F_3) = 2$, $\mu(1) = 2$, $\mu(2) = \lceil \frac{3}{2} \rceil = 2$, $\mu(3) = \lceil \frac{4}{3} \rceil = 2$.

Утверждение 2. $OPT(F, N) \geqslant \mu$.

Доказательство. Докажем данное утверждение от противного, а именно, покажем, что $OPT(F, N)$ не может быть меньше μ, что в свою очередь делает μ нижней границей решения поставленной задачи.

Предположим, что

$$OPT(F, N) < \mu. \tag{1}$$

Выберем такое k, что $\mu(k) = \lceil \frac{1}{k}\xi_K(F_1 \cup \ldots \cup F_k) \rceil = \mu$. Следовательно, k соответствует максимальному значению $\mu(i)$, которое равно μ.

Пусть $\xi = \xi_K(F_1 \cup \ldots \cup F_k)$. Из определения μ следует, что существует F_i, $i \leqslant k$, для которого

$$\xi(F_i) \geqslant \mu. \tag{2}$$

Однако из предположения (1) следует, что количество узлов, в которые попал каждый файл по отдельности, должно быть меньше μ. Для этого необходимо изменить компоновку с канонической компоновки K, на некоторую другую.

Если мы изменим порядок узлов для первых k файлов, тем самым получив новую компоновку, неравенство (2) все ещё будет выполнятся, так как значение величины $\xi(F_1 \cup \ldots \cup F_k)$ останется неизменным. Значит для новой компоновки нам необходимо разместить фрагменты первых k файлов по тем узлам канонической компоновки, где их фрагменты ещё не находятся. Но, так как $N_1, \ldots, N_{h-1}$ — $h - 1$ узлов с самой большой ёмкостью, а, следовательно, никакой другой набор из $h - 1$ узлов не в состоянии поместить данный набор файлов $F_1 \cup \ldots \cup F_k$.

Получаем, что, изменяя порядок узлов в исходной компоновке, мы можем только увеличить количество узлов, необходимых для того, чтобы поместить фрагменты первых k файлов. Следовательно, мы никак не можем удовлетворить условие (1), а значит, получили противоречие. □

Утверждение 3. Для канонической компоновки K

$$\xi_K(F_1) + \ldots + \xi_K(F_k) \leqslant \xi_K(F_1 \cup \ldots \cup F_k) + k - 1,$$

для $1 \leqslant k \leqslant m$.

Доказательство. Между файлами F_i и F_{i+1}, $1 \leqslant i \leqslant k - 1$ существует $k - 1$ граница, а значит они делят попарно по не более, чем одному общему узлу N_j. Таким образом, общее количество общих узлов для смежных файлов не более $k - 1$. □

Для произвольной компоновки K определим понятие *отклонения* d_i для файла F_i таким образом, что $d_i = \xi_K(F_i) - \mu$. Обозначим последовательность таких отклонений $D = (d_1, \ldots, d_m)$.

Утверждение 4. Для канонической компоновки K, где $1 \leqslant k \leqslant m$ выполняется неравенство:

$$\sum_{i=1}^{k} d_i \leqslant k-1. \tag{3}$$

Доказательство.

$$\sum_{i=1}^{k} d_i = \sum_{i \leqslant k} \xi_K(F_i) - k\mu \leqslant \xi_K(F_1 \cup \ldots \cup F_k) + k - 1 - k\mu \quad \text{(из Утверждения 3)} =$$

$$= k\left(\frac{1}{k}\xi_K(F_1 \cup \ldots \cup F_k) - \mu\right) + k - 1 \leqslant k\left(\left\lceil \frac{1}{k}\xi_K(F_1 \cup \ldots \cup F_k)\right\rceil - \mu\right) + k - 1 \leqslant$$

$$\leqslant k-1 \quad \text{(из определения } \mu\text{)}.$$

Утверждение 5. Допустим, что для любой компоновки K, $D = (d_1, \ldots, d_m)$ удовлетворяет (3). Пусть $D' = (d_1, d_2, \ldots, d_{j-1}, d_{j+1}, \ldots, d_m) = (d'_1, \ldots, d'_{m-1})$ получается из D путём удаления элемента $d_j \geqslant 1$. Тогда D' удовлетворяет (3).

Доказательство. Для $k \leqslant j-1$ имеем $\sum\limits_{i \leqslant k} d'_i = \sum\limits_{i \leqslant k} d_i \leqslant k-1$ из предположения о D.

Для $k \geqslant j$ имеем

$$\sum_{i \leqslant k} d'_i = \sum_{i \leqslant j-1} d'_i + \sum_{j \leqslant i \leqslant k} d'_i = \sum_{i \leqslant j-1} d_i + \sum_{j \leqslant i \leqslant k+1} d_i - d_j = \sum_{i \leqslant k+1} d_i - d_j \leqslant$$

$$\leqslant k - d_j \leqslant k-1$$

из предположения о D. □

2. Пример решения задачи максимизации локальности данных

Теперь мы перейдём к рассмотрению алгоритма, который даёт решение поставленной нами задачи, такое что для каждого файла F_i требуется не более $\mu+2$ узла. Для этого нам понадобится описать процедуру, которую мы далее будем называть *кроссинговер*. Название выбрано из-за сходства операции с одноимённым биологическим процессом обмена хромосом у клеток.

Будем называть последовательность отклонений D *уменьшенной*, если $\forall i\ d_i \notin \{1, 2\}$. Предположим, что для некоторых $i < j$

$$\xi(F_i) = \mu - a, \quad \xi(F_j) = \mu + b, \tag{4}$$

где $a \geqslant 0, b \geqslant 3$. Расположим F_j под F_i, как показано на рис. 2, и введём ось координат. Определим следующую функцию:

$$\delta(x) := \xi(F_j, x) - \xi(F_i, x),$$

где $\xi(F_j, x)$ — количество узлов, в которых содержатся фрагменты файла F_j на $(0, x]$. При этом $\delta(0) = 0$.

Отметим, что функция $\delta(x)$ носит ступенчатый характер и изменяет своё значение ровно на 1 и только на границе между узлами. Так, если у файлов F_i и F_j есть *общая* граница в некоторой точке x_0, то значение $\delta(x)$ не изменяется при прохождении точки x_0.

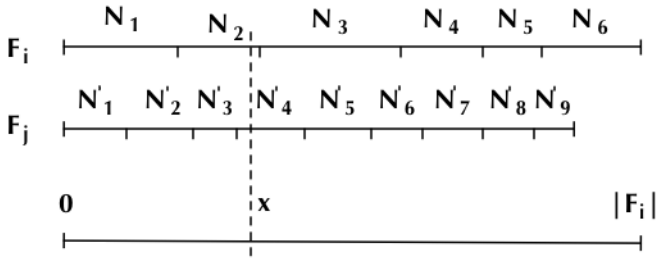

Рис. 2. Определение функции $\delta(x)$

Обратим также внимание на то, что $\delta(|F_j|) \geqslant a+b$. Это неравенство вытекает из первоначальных утверждений о том, что $i < j$, а значит, и $|F_i| \geqslant |F_j|$. Таким образом, $\xi(F_i, |F_i|) \leqslant \mu - a$.

Пусть для некоторого значения $c, 0 < c \leqslant a+b$ x_0 — граница первого узла, для которого $\delta(x_0) = c$. Тогда *кроссинговером* будем называть следующую операцию: фрагменты файлов F_i и F_j, заключённые в отрезке $0 \leqslant x \leqslant x_0$, взаимозаменяются, как показано на рис. 3. В данном примере $c = 2$, а первым узлом, для которого $x_0 = 2$ является N'_4. Следовательно участки от N'_1 до N'_4 заменятся на N_1 и N_2, вместе с частью узла N_3, который теперь поделим между файлами F_i и F_j.

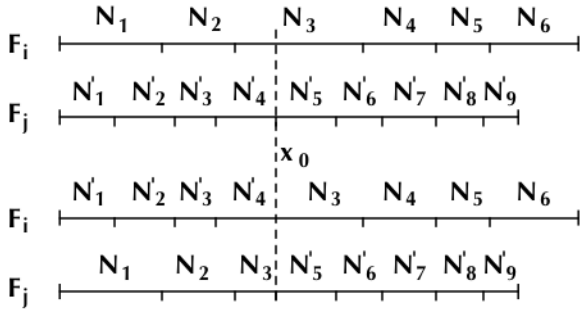

Рис. 3. Кроссинговер

Очевидно, что в результате выполнения операции кроссинговера изменяются значения $\xi(F_i)$ и $\xi(F_j)$. Зачастую x_0 не является граничной точкой для F_i, а значит,

$$\xi'(F_i) = \xi(F_i) + c + 1, \quad \xi'(F_j) = \xi(F_j) - c.$$

Однако если x_0 все же является граничной точкой для F_i, то имеем

$$\xi'(F_i) = \xi(F_i) + c, \quad \xi'(F_j) = \xi(F_j) - c.$$

Так, в указанном примере F_i изначально попадает в 6 узлов, а F_j — в 9. После кроссинговера F_i и F_j попали в 8 узлов. Тем самым множество D изменяется с $(d_1, d_2, \ldots, d_i, \ldots, d_j, \ldots, d_m)$ на $(d_1, d_2, \ldots, d_i + c + 1, \ldots, d_j - c, \ldots, d_m)$.

Теперь вернёмся к описанию самого алгоритма. Для этого вновь рассмотрим некоторую каноническую компоновку K, для которой множество отклонений D представляется как $(d_1, \ldots, d_m)$. Без потери общности допустим, что для D выполняется условие (3), и оно также является уменьшенным.

Пусть j — первый индекс, для которого $d_j \geqslant 3$. Если такого индекса не существует, то алгоритм заканчивает свою работу, так как для всех файлов $\xi(F_i) \leqslant \mu + 2$, что он и должен обеспечивать.

Подставим $k = 1$ в неравенство (3), в результате чего получаем, что $d_1 \leqslant 0$. Следовательно, $j \geqslant 2$, и мы имеем двух кандидатов, F_i и F_j, для операции кроссинговера.

Используя те же обозначения, что и в (4), имеем $d_1 = -a, d_j = b, a \geqslant 0, b \geqslant 3$. Рассмотрим два случая.

Случай I, $b > a$. Применяем операцию кроссинговера для F_1 и F_j, используя $c = a+1 \leqslant a+b$. Это приводит к $D' = (2, \ldots, b-a-1, \ldots\ d_m)$, т.е. $d'_1 = 2$, а $d'_j = b-a-1$. Исключая из D' элемент $d'_1 = 2$, получаем $D'' = (d_2, d_3, \ldots, d'_j, d_{j+1}, \ldots, d_m)$, которое является уменьшенным.

Случай II, $b \leqslant a$. Применяем операцию кроссинговера для F_1 и F_j, используя $c = b-2 \leqslant a+b$. Это приводит к $D' = (b-a-1, \ldots, 2, \ldots\ d_m)$, которое удалением элемента $d'_j = 2$ преобразуем к $D'' = (b-a-1, d_2, d_3, \ldots, d_{j-1}, d_{j+1}, \ldots, d_m)$, также являющимся уменьшенным.

Утверждение 6. Полученное множество D'' удовлетворяет (3).

Доказательство. Докажем данное утверждение сначала для *случая I*, когда $1 \leqslant k \leqslant j-2, D'' = (d_2, \ldots, d_{j-1}, b-a-1, d_{j+1}, \ldots) = (d''_1, d''_2, \ldots)$.

Так как по условию, j — наименьший индекс, такой что $d_j > 2$, то $d_2 \leqslant 0, \ldots, d_{j-1} \leqslant 0$. Из определения уменьшенной последовательности отклонений D'' его элементы не могут быть равными 1 и 2, следовательно, $\sum\limits_{i \leqslant k} d''_i \leqslant 0 \leqslant k-1$.

Для $k \geqslant j-1$ из (3) имеем:

$$\sum_{i \leqslant k} d''_i = \sum_{2}^{j-1} d_i + b - a - 1 + \sum_{i=j}^{k} d''_i = \sum_{i=1}^{j} d_i - 1 + \sum_{i=j+1}^{k+1} d_i = \sum_{i=1}^{k+1} d_i - 1 \leqslant k-1.$$

Теперь, если $b-a-1 \in \{1, 2\}$, то мы удаляем элемент d''_{j-1} из D'', получая при этом новую последовательность D''', которая по *Утверждению 5* также удовлетворяет (3).

Докажем теперь верность указанного утверждения для *случая II*.

Имеем $D'' = (b-a-1, d_2, \ldots, d_{j-1}, d_{j+1}, \ldots) = (d''_1, d''_2, \ldots)$. Так как $b \leqslant a$, аналогично получаем $d''_1 \leqslant 0, \ldots, d_{j-1} \leqslant 0$, значит, для любого $k \leqslant j-1$

$$\sum_{i \leqslant k} d''_i \leqslant 0 \leqslant k-1.$$

Если $k \geqslant j$, то из (3) имеем

$$\sum_{i \leqslant k} d''_i = \sum_{i \leqslant j-1} d''_i + \sum_{i \geqslant j}^{k} d''_i = b - a - 1 + \sum_{i=2}^{j-1} d_i + \sum_{i=j+1}^{k+1} d_i = \sum_{i=1}^{k+1} d_i - 1 \leqslant k-1.$$

Отметим, что из (3) $\Rightarrow d_1 \leqslant 0$, а значит, предложенный алгоритм может продолжить свою работу, используя D'', D''', и так далее. На каждом своём новом шаге алгоритм будет уменьшать количество отклонений $d_i \notin \{1, 2\}$.

Таким образом, за не более чем $m-1$ шаг данный алгоритм получит последовательность D^*, которая будет либо пустой, либо все её элементы $d_i \leqslant 0$. После чего алгоритм прекратит свою работу.

2.1. Оценка времени работы рассмотренного решения

Рассматриваемый в данной статье алгоритм состоит из двух частей. В первой части происходит сортировка множества файлов F и узлов хранения данных распределённой вычислительной системы N для получения канонической компоновки K, которая занимает $O(n \log(n))$ времени.

Вторая часть алгоритма посвящена последовательному выполнению нескольких операции так называемого кроссинговера. Количество таких операций не превышает $m-1$. Внутри каждой из этих операции процесс поиска файлов для

кроссинговера занимает не более m шагов, тогда как подсчёт разниц на границах узлов хранения данных занимает $O(n)$.

Следовательно итоговое время работы рассмотренного алгоритма составляет

$$O(n(m + \log(n))).$$

3. Заключение

В данной статье сформулирована задача обеспечения максимальной локализации распределения элементов по контейнерам на примере распределения фрагментов файлов по узлам распределённой системы хранения данных. Данная задача актуальна для современных распределённых вычислительных систем.

Нами было показано, что нахождение оптимального решения данной задачи является NP-полной задачей, однако для её решения существуют алгоритмы, способные за полиномиальное время дать решение, близкое к оптимальному. В данной статье подробно описан один из таких алгоритмов, а также нами доказано, что данное решение отличается от оптимального на фиксированную константу 2.

Современные распределённые вычислительные системы насчитывают тысячи узлов, способных хранить данные, необходимые для её повседневной работы. Для того чтобы обеспечить максимальную эффективность хранения данных, а также обеспечения защиты от сбоев отдельных узлов и увеличения скорости их обработки требуются высокоэффективные алгоритмы.

Таким образом, поиск наиболее оптимальных решений данных проблем будет продолжаться и будет являются предметом дальнейших исследований в данной области.

Литература

1. Oceanstore: An Architecture for Global-Scale Persistent Storage / J. Kubiatowicz, D. Bindel, Y. Chen et al. // ASPLOS-IX / ASPLOS. — New York, USA: ACM, 2000. — Pp. 190–201.
2. Wide-Area Cooperative Storage with CFS / F. Dabek, M. Kaashoek, D. Karger et al. // SOSP. — 2001. — Pp. 202–215.
3. *Rowstron A., Druschel P.* Storage Management and Caching in PAST, A Large-scale, Persistent Peer-to-peer Storage Utility // SOSP. — 2001. — Pp. 188–201.
4. Ivy: A Read/Write Peer-to-Peer File System / A. Muthitacharoen, R. Morris, T. M. Gil, B. Chen // SOSDI / Ed. by D. E. Culler, P. Druschel. — USENIX Association, 2002.
5. *Coffman E. G. J., Garey M. R., Johnson D. S.* Approximation Algorithms for NP-hard Problems. — Boston, USA: PWS Publications, 1996. — Pp. 46–96.
6. *Coffman E. G. J., Lueker G.* Approximation Algorithms for Extensible Bin-Packing // Journal of Scheduling. — Vol. 9. — Hingham, MA, USA: Kluwer Academic Publishers, 2006. — Pp. 63–69.
7. Perfect Packing Theorems and the Average-Case Behavior of Optimal and Online Bin Packing / E. G. J. Coffman, C. Courcoubetis, M. R. Garey et al. // J. Discrete Math. — 2000. — Vol. 13, No 3. — Pp. 384–402.
8. *Seiden S. S.* On the Online Bin Packing Problem // Journal of the ACM. — 2002. — Vol. 49, No 5. — Pp. 640–671.
9. *Karp R. M.* Reducibility Among Combinatorial Problems // SOSP. — The IBM Research Symposia Series. — Plenum Press, New York, 1972. — Pp. 85–103.
10. *Garey M. R., Johnson D. S.* Complexity Results for Multiprocessor Scheduling under Resource Constraints // SIAM Journal on Computing. — 1975. — Vol. 4, No 4. — Pp. 397–411.

UDC 004.75

The Problem of Data Placement in Distributed Systems

A. V. Zhipa

Department of Information Technology
Peoples' Friendship University of Russia
6, Miklukho-Maklaya str., Moscow, Russia, 117198

Distributed system effectiveness depends dramatically on the way it manages incoming tasks and data against limited computational resources that are at its disposal. Due to ever-inreasing amount of incoming data distributed systems are required to efficiently manage the way its storage and processing are being made.

Nowadays the distributed system design is significantly flounced by the manner it leverages high load scenarios, provides data storage functionality and uses the underlying resources.

An effective distributed system's resource management has to balance trade-offs between single node resource consumption and the overall loss of data locality, that is inevitable due to data fragmentation.

In this article we will formalize the problem of data placement by maximizing data storage locality in distributed data systems, which as it turns out is a NP-complete task. We will later describe a polynomial-time algorithm that is capable of providing us a solution that is within an additive constant from the optimal one.

Key words and phrases: fragmentation, distributed systems, NP-complete problems, bin-packing problems, data storage locality.

References

1. J. Kubiatowicz, D. Bindel, Y. Chen, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells, B. Zhao, Oceanstore: An Architecture for Global-Scale Persistent Storage, in: ASPLOS-IX, ASPLOS, ACM, New York, USA, 2000, pp. 190–201.
2. F. Dabek, M. Kaashoek, D. Karger, R. Morris, I. Stoica, Wide-Area Cooperative Storage with CFS, in: SOSP, 2001, pp. 202–215.
3. A. Rowstron, P. Druschel, Storage Management and Caching in PAST, A Large-scale, Persistent Peer-to-peer Storage Utility, in: SOSP, 2001, pp. 188–201.
4. A. Muthitacharoen, R. Morris, T. M. Gil, B. Chen, Ivy: A Read/Write Peer-to-Peer File System, in: D. E. Culler, P. Druschel (Eds.), SOSDI, USENIX Association, 2002.
5. E. G. J. Coffman, M. R. Garey, D. S. Johnson, Approximation Algorithms for NP-hard Problems, PWS Publications, Boston, USA, 1996, Ch. Approximation algorithms for bin packing — a survey, pp. 46–96.
6. E. G. J. Coffman, G. Lueker, Approximation Algorithms for Extensible Bin-Packing, in: Journal of Scheduling, Vol. 9, Kluwer Academic Publishers, Hingham, MA, USA, 2006, pp. 63–69.
7. E. G. J. Coffman, C. Courcoubetis, M. R. Garey, D. S. Johnson, P. W. Shor, R. R. Weber, M. Yannakakis, Perfect Packing Theorems and the Average-Case Behavior of Optimal and Online Bin Packing, J. Discrete Math. 13 (3) (2000) 384–402.
8. S. S. Seiden, On the Online Bin Packing Problem, Journal of the ACM 49 (5) (2002) 640–671.
9. R. M. Karp, Reducibility Among Combinatorial Problems, in: SOSP, The IBM Research Symposia Series, Plenum Press, New York, 1972, pp. 85–103.
10. M. R. Garey, D. S. Johnson, Complexity Results for Multiprocessor Scheduling under Resource Constraints, SIAM Journal on Computing 4 (4) (1975) 397–411.