Теорія оптимальних рішень

УДК 519.85

В.В. БОЙКО, В.В. ГОРИН, В.Н. КУЗЬМЕНКО

ОПТИМИЗАЦИЯ РАБОТЫ АЛГОРИТМА РАСПЫЛЕНИЯ ИНФОРМАЦИИ В СИСТЕМЕ РАСПРЕДЕЛЕННОГО ХРАНЕНИЯ

Рассматривается задача оптимизации работы алгоритмов распыления информации и ее последующего восстановления в системе распределенного хранения. К системе хранения информации предъявляется требование надежности и конфиденциальности. Приводятся результаты тестирования предложенных алгоритмов.

Введение. Алгоритмы распыления (распределения) информации – одна из основных компонент в системах распределенного хранения, с помощью которых решаются такие задачи обработки и хранения информации как надежность, безопасность, защищенность, скорость доступа и другие. Распределенное хранение информации может применяться как в компьютерной сети, так и на отдельном носителе информации для ее более эффективной обработки. Вопрос надежности передачи информации возникает при параллельной ее обработке, при параллельных и распределенных вычислениях, при использовании облачных технологий [1–3].

В общем задача ставится следующим образом. Необходимо обеспечить надежное хранение конфиденциальной информации на распределенных носителях, а также быстрый доступ к ней и обработку.

Простейший вариант решения задачи – это шифрование файлов и хранение нескольких копий зашифрованных файлов на различных носителях. Однако такой вариант решения задачи является самым затратным по объемам хранения и передачи данных и избыточным по надежности. Если зашифрованный файл F хранится в k копиях и вероятности потери одной копии независимы и равны p_1, то общий объем хранения равен $k \cdot |F|$, а вероятность сохранения информации равна $p_{sf} = 1 - p_1^k$, где $|F|$ – размер файла.

Другой вариант решения – это распределить информацию, хранящуюся в файле, на n файлов F_i, $i=1,...,n$ меньшего размера $|F_i|<|F|$ таким образом, чтобы исходный файл можно было восстановить по любым $k<n$ частям, а общий объем хранения $\sum_{i=1,...,n}|F_i|$ был меньше, чем в первом варианте $k\cdot|F|>\sum_{i=1,...,n}|F_i|>|F|$. Вероятность сохранения полной информации в этом случае определяется биноминальным распределением

$$p_{sf}=\sum_{t=0}^{n-k}C_n^t p_1^t(1-p_1)^{n-t}=1-\sum_{t=0}^{k-1}C_n^t p_1^{n-t}(1-p_1)^t .$$

При хранении и передаче информации наиболее уязвимым этапом является передача файлов или его частей по каналам связи. Поэтому при потере некоторых передаваемых частей исходного файла возможно его восстановление, но для этого необходимо выполнить передачу дополнительных частей или повторить передачу не прошедших частей.

Для первого варианта, когда файл хранится целиком, математическое ожидание времени передачи файла при повторных попытках передается до тех пор пока файл не пройдет и оценивается так:

$$\bar{t}_F=\sum_{s=1}^{\infty}t_F s\cdot p_1^{s-1}(1-p_1)=t_F(1-p_1)\sum_{s=1}^{\infty}s\cdot p_1^{s-1}=t_F(1-p_1)\frac{1}{(1-p_1)^2}=\frac{t_F}{(1-p_1)},$$

где t_F – время передачи файла; s – попытка, с которой удалось передать файл; p_1 – вероятность не прохождения передачи.

При распределенном хранении файла возможна параллельная передача его частей с пропорциональным уменьшением времени. В этом случае будем считывать все n частей. Если будет удачно считано k частей и больше из n, то файл будет восстановлен, если удачно считанных частей будет меньше, то эти части параллельно будут считаться еще раз. Для такого варианта время считывания оценим следующим образом. Пусть d_t – это доля файла, считанная при условии, что $t=0,...,n$ частей не были считаны. Для $t=0,...,n-k$ файл может быть восстановлен, следовательно $d_t=1$, для $t=n-k+1,...,n$ $d_t=(n-t)/k$.

Математическое ожидание доли файла, которая будет считана за первую попытку считывания равно

$$\bar{d}=\sum_{t=0}^{n}d_t C_n^t p_1^t(1-p_1)^{n-t}=\sum_{t=0}^{n-k}C_n^t p_1^t(1-p_1)^{n-t}+\sum_{t=n-k+1}^{n}\frac{n-t}{k}C_n^t p_1^t(1-p_1)^{n-t} .$$

Среднее время считывания файла в этом случае оценим как $\bar{t}_{F_i}=t_{F_i}/\bar{d}$, где t_{F_i} – время считывания одной части, одинаковое для всех частей.

Дадим формальное определение алгоритму распыления информации. Алгоритм распыления информации (Information dispersal algorithm – IDA) – это алгоритм разделения секрета, который используется для побитовой «нарезки» данных таким образом, что когда данные передаются по сети либо сохраняются на носителях информации, то невозможно определить исходную последовательность бит, не имея соответствующего ключа. При наличии ключа информация может быть восстановлена. Таким образом, можно сказать, что IDA – это функция, принимающая, как минимум, два аргумента – исходное сообщение m и некоторый секретный ключ S_k. Результат работы данной функции – набор сообщений m_i одинаковой длины, сумма длин которых равна длине исходного сообщения. Сообщения m_i состоят из тех же блоков (битов, байтов и т. д.), что и исходное сообщение, но в другой последовательности, которая однозначно определяется ключом S_k.

Для последующего восстановления исходного сообщения необходимо использовать алгоритмы восстановления или коды восстановления. Код восстановления от потерь (erasure code – EC) – это разновидность кодов прямой коррекции ошибок (forward error correction – FEC), который преобразует сообщение из k символов в более длинное сообщение (кодовое слово), состоящее из n символов таких, что исходное сообщение может быть восстановлено из некого подмножества этих n символов.

В литературе алгоритмы распыления и восстановления не всегда описываются как самостоятельные алгоритмы. Некоторые авторы оба вышеописанных понятия объединяют в одно и в зависимости от того, чему уделяется больше внимания, называют объединенный алгоритм как IDA, так и EC [1, 2]. В данной работе эти алгоритмы разделяются.

Рассматривается использование оптимальных кодов восстановления. Оптимальные коды восстановления потерь обладают тем свойством, что любых k из n кодовых слов достаточно для восстановления исходного сообщения.

Оба алгоритма IDA и EC могут быть эффективно применены: 1) при передаче данных по небезопасному каналу с возможной потерей части информации; 2) при сохранении данных на массив запоминающих устройств (storage array, далее – массив). Рассмотрим более подробно пункт 2).

Постановка задачи. Необходимо хранить исходные данные на массив так, чтобы:

1) потенциальный злоумышленник не смог восстановить исходные данные, даже при наличии у него доступа ко всем устройствам (storage nodes);

2) выход из строя некоторого количества устройств не привел к потере исходных данных и не нарушил требования пункта 1).

Будем считать, что у нас есть исходное сообщение m длины s бит, которое мы сохраним на n устройств так, чтоб любых k устройств было достаточно для восстановления исходного сообщения.

Для достижения поставленных целей последовательно применяются алгоритмы IDA и EC (рисунок) следующим образом:

1) с помощью IDA исходное сообщение m разбивается на k сообщений m_i. На этом шаге выбирается секретный ключ S_k, который необходим для обратной сборки исходного сообщения;

2) с помощью EC генерируется дополнительно $n-k$ кодовых сообщений m_j. Для создания оптимального кода восстановления потерь здесь используется алгоритм, основанный на алгоритме Рида – Соломона [3];

3) полученные сообщения m_i и m_j объединяются в одно множество и записываются на устройства хранения данных.

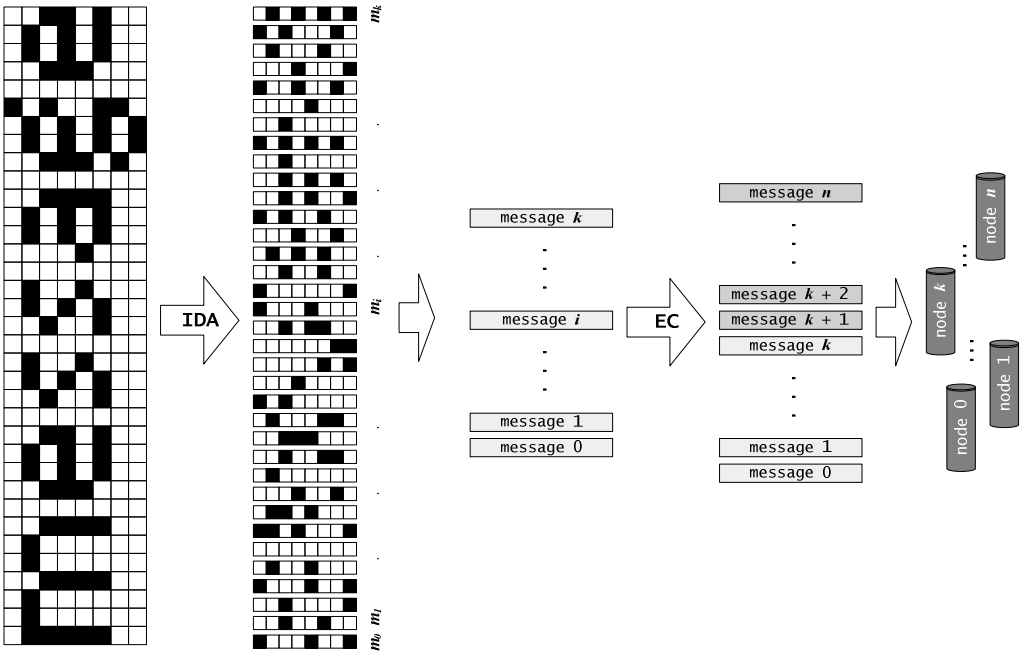

РИСУНОК. Схема последовательной работы алгоритмов IDA и EC

Выполнение перечисленных пунктов позволяет достичь поставленной задачи и гарантировать выполнение свойств пп. 1) и 2). К тому же хранение исходного сообщения в таком виде имеет некоторые дополнительные преимущества.

Поскольку для получения исходного длинного сообщения нужно прочесть k коротких сообщений m_i с разных устройств, то операцию чтения можно распараллелить, уменьшив в k раз время чтения с массива запоминающих устройств. Таким образом к заданным свойствам пп. 1) и 2) добавляются следующие свойства, касающиеся обработки информации:

4) операцию чтения можно ускорить в k раз по сравнению с операцией чтения одного длинного сообщения с одного устройства. Аналогичное свойство справедливо и для операции записи на устройства.

Алгоритм оптимального кода восстановления потерь можно выбрать так, чтобы при необходимости сгенерировать дополнительное кодовое слово (сообщение) не нужно было повторять генерацию предыдущих слов. Таким образом получаем еще одно важное свойство:

5) уровень избыточности n/k можно менять, не затрагивая уже сохраненные на запоминающих устройствах сообщения.

Рассмотрим подробнее, как можно построить необходимые алгоритмы.

Для построения IDA можно воспользоваться криптостойким генератором псевдослучайных чисел на базе AES (Advanced Encryption Standard). Ключ S_k можно использовать как инициализирующее случайное число (random seed) генератора. Числа, получаемые на выходе генератора, можно использовать в качестве номера сообщения i, в которое записывается следующий блок (бит/байт/…) из исходного сообщения m.

Для оптимизации работы IDA могут быть использованы различные критерии и, в частности, те, которые описаны в начале статьи – оптимизация по математическому ожиданию времени записи-считывания информации, по объему хранимой информации, по надежности ее хранения и передачи.

Результаты тестирования. Тестирование проводилось для проверки и подтверждения работоспособности и эффективности применения алгоритма распыления информации. Было проведено измерение скорости передачи данных с использованием IDA/EC алгоритма и без него. Тестовое приложение разворачивалось на серверах Windows Azure (один сервер на восточном побережье США, один – на западном). Тестовый файл «распылялся» на сервера Amazon S3 (три точки в США), Google Cloud (одна точка в США), Windows Azure (две точки в США) с избыточностью 50% так, чтоб его можно было собрать при потере соединения с любым из трех провайдеров. По результатам, представленным далее, можно видеть, что скорость «распыления» данных выше, чем наибольшая из прямых скоростей (raw speed) доступа Azure/Amazon/Google и более чем в 10 раз больше наименьшей. В табл. 1 приведена скорость закачки файла на сервер, в табл. 2 – скорость скачивания файла с сервера. Скорость в таблицах указана в мегабитах в секунду.

ТАБЛИЦА 1. Скорость закачки (upload speed) файла объемом 1Гб

Test Application Location	Raw Speed			IDA/EC Algorithm Speed
	Google Cloud	Amazon S3	Windows Azure	
US East	8,46 Mbps	72,72 Mbps	97,68 Mbps	119,19 Mbps
US West	2,68 Mbps	44,98 Mbps	70,54 Mbps	78,66 Mbps

ТАБЛИЦА 2. Скорость загрузки (download speed) файла объемом 1Гб

Test Application Location	Raw Speed			IDA/EC Algorithm Speed
	Google Cloud	Amazon S3	Windows Azure	
US East	113,77 Mbps	34,27 Mbps	95,35 Mbps	162,01 Mbps
US West	85,53 Mbps	44,88 Mbps	105,02 Mbps	156,49 Mbps

Заключение. В результате выполненной работы построены алгоритмы распыления и восстановления информации, которые показали высокие скорости записи и считывания. Дальнейшее развитие работы будет направлено на улучшение эффективности работы алгоритмов и оптимизации их работы по другим критериям.

В.В. Бойко, В.В. Горін, В.М. Кузьменко

ОПТИМІЗАЦІЯ РОБОТИ АЛГОРИТМА РОЗПИЛЕННЯ ІНФОРМАЦІЇ
У СИСТЕМІ РОЗПОДІЛЕНОГО ЗБЕРІГАННЯ

Розглядається задача оптимізації роботи алгоритма розпилення інформації та її наступного відновлення у системі розподіленого зберігання. До системи зберігання інформації висуваються вимоги надійності та конфіденційності. Наводяться результати тестування запропонованих алгоритмів.

V.V. Boyko, V.V. Gorin, V.M. Kuzmenko

INFORMATION DISPERSAL ALGORITHM OPTIMIZATION IN A DISTRIBUTED STORAGE SYSTEM

The problem of information dispersal algorithm optimization in distributed data storage is discussed. Necessary security, reliability and confidentiality requirements are claimed. Testing results for algorithms proposed provided.

1. *Lin S.J.*, *Chung W.H.* An efficient (n, k) information dispersal algorithm for high code rate system over fermat fields // IEEE Communications Letters. – 2012. – Vol. 16, **12**. – P. 2036–2039.
2. *Weatherspoon H.*, *Kubiatowicz J.D.* Erasure coding vs. replication: A quantitative comparison // Lecture Notes in Computer Science. – 2002. – Vol. 2429. – P. 328–337.
3. *Горин В.В., Лютенко В.М.* Имплементация и оптимизация алгоритма Рида-Соломона для создания кодов восстановления потерь в данных // Теорія оптимальних рішень. – 2012. – С. 126–135.

Получено 13.03.2013