Е.В. ПУГИН,
Р.А. СИМАКОВ

Оценка числа уникальных значений входного потока в распределенной среде

Муромский институт (филиал) ФГБОУ ВПО «Владимирский государственный университет имени А.Г. и Н.Г. Столетовых»

УДК 004.047

В статье рассматриваются вопросы определения числа уникальных значений во входном потоке в распределённой среде.

In this paper, estimating the number of unique input values in a distributed environment is described.

В некоторых системах присутствует необходимость подсчёта уникальных значений входного потока для последующей статистической оценки или применения в дальнейшей обработке поступающей информации [7]. Не всегда нужно знать сколько раз встречается конкретный элемент в потоке, а необходима информация об уникальности значений. Примером такой ситуации служит определение числа различных пользователей, посетивших тот или иной ресурс, размещённый на нескольких узлах распределённой сети [8]. Уникальность пользователей может определяться по IP адресам или другим идентификаторам [6]. Ещё одной областью применения результата операции нахождения уникальных значений может быть использование в СУБД для подсчёта нового количества ячеек после операции загрузки массива и его преобразования в другой с изменением числа измерений. Например, в распределённой СУБД SciDB [5,9] этим занимаются операторы redimension и redimension_store.

В данном случае проблема может быть решена хранением в памяти ЭВМ копий различающихся элементов. Такой простейший случай плох тем, что при огромном количестве уникальных значений, память машины быстро заполнится и иссякнет. Другой проблемой здесь является распределённость датчиков. Если источников снимаемой информации несколько, то необходимо организовывать

обмен между ними [8]. Каждое новое значение нужно пересылать на все другие узлы такой сети.

Наилучшим подходом к решению этой проблемы будет являться примерная оценка числа уникальных значений. Основные преимущества, которые даёт этот метод – это уменьшение потребления памяти вплоть до 64 КБ, что зависит от алгоритма, а также устойчивость к появлению повторяющихся элементов, что базируется на свойствах хеш-функций.

Суть данного метода заключается в нахождении некоторого хеша исходного значения. К примеру идеальная хеш-функция без коллизий с хешем длиной 32 бита сможет отразить 2^{32} уникальных элемента. Длину хеша следует подбирать из приблизительных оценок количества различных значений. Если заранее известно, что число уникальных элементов превышает максимальное значение данного хеша, то необходимо увеличить его длину.

Использование хешей значений основывается на следующем принципе. Вероятность того, что первый бит хеша будет равен 1 равна 0.5. Вероятность того, что первый бит установлен в 0, а второй – 1 равна 0.25. Таким образом, вероятность того, что i-ый бит равен 1, а все биты справа от него равны 0, равна $\frac{1}{2^i}$.

Пример. 32-битные хеши некоторых значений:

10101000101011111010100111010**10**,	P=0.25
00100000011011011001011011011**1**,	P=0.5
11101100110011100111011111010**1**1,	P=0.5
01100100001011000100001111011001,	P=0.5
10001011000111111001001110**10000**,	P=0,03125

Во входном потоке обязательно найдутся такие значения, хеш функция которых будет иметь описанные свойства. Причём, чем больше различных значений содержит входной поток, тем больше вероятность встретить элементы, хеш которых содержит большее число подряд идущих нулей в правой части. Имея такую информацию, мы можем оценить количество уникальных элементов между 2^i и 2^{i+1}.

Отсюда можно сделать вывод, что при большом числе различных значений, точность оценки может оказать очень низкой. Поэтому в этом случае используется специальная методика уменьшения ошибки вычислений. Существует вариант использования нескольких хеш-функций с усреднением их результатов. Недостатком будет яв-

ляться увеличение процессорного времени на обработку всеми функциями каждого значения. Следует отметить, что для увеличения точности оценки, необходимо выбирать наиболее устойчивую к коллизиям функцию. Это может быть достигнуто тестированием и сравнением результатов работы имеющихся функций.

Важной особенностью алгоритмов подсчёта уникальных элементов является то, что на небольших количествах различных значений, они дают очень большую погрешность. Это связано с некоторой вероятностью появления нескольких подряд идущих нулей в хеше значения. Поэтому необходимо учитывать данный вариант.

В качестве решения этой проблемы следует разделить учёт уникальных значений до определённого количества и после. Например, возможно вначале хранить различные элементы входного потока в некотором множестве, а затем при превышении таким хранилищем заранее установленного лимита памяти, переводить значения множества в структуры данных, присущие конкретному алгоритму.

Основным преимуществом такого подхода является то, что при небольшом количестве уникальных значений, мы будем знать их точное число, а не приближённую оценку.

Рассмотрим более подробно алгоритм подсчёта различных значений LogLog, описанный М. Дюранд и Ф. Флажолетом [1], как один из наиболее известных и простых разновидностей существующих алгоритмов.

Имеем набор входных значений, хеши которых найдены и доступны. Длина хеша – 32 бита. Выбираем такие числа k и m, что $3 < k < 17$

$$m = 2^k$$

Идея заключается в разделении элементов в m групп M_j, которые также называются корзинами (от англ. bucket – корзина). Для обращения к каждой группе используются первые k битов хеша значения в качестве индекса группы. Это делается для уменьшения результирующей ошибки вычислений. При определении результата будет найдено усреднённое значение по всем группам, что даст наиболее верный результат.

При добавлении нового значения v, находится его хеш $h = hash(v)$. Первые k бит используются в качестве индекса группы.

Затем происходит нахождение ранга, то есть индекса первого бита, установленного в 1 справа. Результат сравнивается с текущим значением группы и выбирается максимальное значение.

$$h = hash(v)$$

$$M_j = max(M_j, rank(h))$$

Результат, то есть количество уникальных элементов, определяется по следующей формуле с учётом погрешности:

$$E = a_m * m * 2^{\frac{1}{m}\sum M_j},$$

где

$$a_m = (\Gamma(-1/m)\frac{2^{-1/m}-1}{\log 2})^{-m},$$

где Г(x) – гамма-функция:

$$\Gamma(x) = \frac{1}{x}\int_0^{\infty} e^{-t}t^x dt$$

Стандартная ошибка в данном случае будет равна:

$$\sigma \approx \frac{1.30}{\sqrt{m}}$$

Видно, что чем больше параметр *k*, тем больше будет *m* и тем меньше будет ошибка. Также следствием будет являться увеличение используемой памяти. При длине хеша меньшей 256 бит, на каждую группу M_j можно отводить 1 байт памяти. Для меньших *k* и *m* можно выделять всего лишь по несколько бит в тех системах, где объёмы доступной памяти очень малы. Следовательно, при *k* = 16, *m* = 65536, будет затрачено 64 КБ памяти, что вполне приемлемо для современных компьютеров. Стандартная ошибка в этом случае будет равняться 0.5% при условии идеальной хеш-функции. Такое отклонение является вполне приемлемым при оценке большого числа уникальных значений.

Для сравнения приведём стандартные ошибки других алгоритмов в табл. 1, а также необходимые для них единицы памяти и требуемый объём.

Таблица1

Сравнение алгоритмов подсчёта числа уникальных значений

Алгоритм	**Стандартная ошибка (σ)**	**Единицы памяти**	**Объём памяти при σ = 0.02, $n = 10^8$**
Adaptive Sampling	$1.20/\sqrt{m}$	Записи (слова по 24 бита и больше)	10.8 КБ
Prob. Counting [3]	$0.78/\sqrt{m}$	Слова (24-32 бита)	6.0 КБ
Multires. Bitmap [2]	$\approx 4.4/\sqrt{m}$	Биты	4.8 КБ
LogLog [1]	$1.30/\sqrt{m}$	«Маленькие байты» (5 бит)	2.1 КБ
Super-LogLog	$1.05/\sqrt{m}$	«Маленькие байты» (5 бит)	1.7 КБ
Hyper-LogLog	$1.05/\sqrt{m}$	«Маленькие байты» (3-5 бит)	1.7 КБ

Видим, что большинство алгоритмов имеют примерно одинаковую стандартную ошибку, но могут отличаться в плане реализации и использования памяти. Самыми оптимальными являются улучшенные алгоритмы на основании алгоритма LogLog под названием Super-LogLog и Hyper-LogLog, также описанные М. Дюранд и Ф. Флажолетом [2].

После обработки входных потоков на большом числе узлов, нам необходимо объединить полученные результаты в одном месте и рассчитать итоговое число уникальных элементов. Практически все алгоритмы подсчёта различных значений имеют такую возможность [4]. На примере алгоритма LogLog мы видим, что значения групп ищутся из максимума ранга текущего хеша и самого значения группы. Следовательно, при объединении групп необходимо провести ту же операцию, что никак не отразится на результате. Таким образом, при объединении результата все узлы пересылают значения своих групп некоторому главному узлу, который выполняет операцию слияния своих групп с принятыми, а затем рассчитывает результат с учётом стандартной ошибки.

В случае, когда число различных значений на входе мало (приблизительно до 2000-4000), то их можно хранить в оригинальном виде и пересылать на другие узлы. Если число уникальных значений превысит норму, то необходимо провести одноразовую опера-

цию конвертации этих значений в группы алгоритма подсчёта. Аналогичная операция выполняется при пересылке: если на принимающем узле не используется счётчик уникальных значений, а были приняты новые значения, в сумме превысившие норму, то происходит преобразование. Если на принимающем узле используется такой счётчик, то происходит преобразование принятых значений «на лету».

Таким образом, применение описанных алгоритмов подсчёта уникальных значений позволяет с достаточно низкой погрешностью оценивать данные со входных потоков информации в распределённой системе. Возможно применение таких алгоритмов в устройствах с жёсткими ограничениями на объёмы используемой памяти и процессорного времени, так как основной операцией является добавление значения к группам.

Для повышения точности оценки следует увеличивать параметры k и m, а, следовательно, и объём памяти. Также необходимо исследовать хеш-функции на предмет коллизий, так как слабые функции вносят огромную погрешность в результат.

В целом, описанный алгоритм LogLog и его разновидности идеально подходят для непрерывной работы с «живыми» потоками данных.

Литература

1. Durand, M., Flajolet, P.: Loglog counting of large cardinalities. In: Proc. 11th European Symp. on Algorithms. (2003) 605–617

2. Estan, C., Varghese, G., Fisk, M.: Bitmap algorithms for counting active flows on high speed links. In: Proc. 3rd ACM SIGCOMM Conf. on Internet Measurement. (2003) 153–166

3. Flajolet, P., Martin, G.N.: Probabilistic counting algorithms for data base applications. J. of Computer and System Sciences 31 (1985) 182–209

4. Gibbons, P.B., Tirthapura, S.: Distributed streams algorithms for sliding windows. In: Proc. 14th ACM Symp. on Parallel Algorithms and Architectures. (2002) 63–72

5. Overview of SCIDB: Large scale array storage, processing and analysis. Rogers J., Zdonik S., Simakov R., Velikhov P., Smirnov A., Knizhnik K., Soroush E., Balazinska M., DeWitt D., Heath B., Maier D., Madden S., Stonebraker M., Patel J., Brown P.G. Proceedings of the ACM SIGMOD International Conference on Management of Data 2010 International Conference on Management of Data, SIGMOD '10. Сер. "Proceedings of the 2010 International Conference on Management of Data, SIGMOD '10" sponsors: ACM SIGMOD. Indianapolis, IN, 2010. С. 963-968.

6. Venkataraman, S., Song, D., Gibbons, P.B., Blum, A.: New streaming algorithms for high speed network monitoring and internet attacks detection. In: Proc. 12th ISOC Network and Distributed Security Symp. (2005)

7. Пугин Е.В., Симаков Р.А. Распараллеливание математических вычислений на примере операторов линейной алгебры. Алгоритмы, методы и системы обработки данных: Электронный научный журнал /под ред. С.С.Садыкова, Д.Е. Андрианова.- Вып. 17.- Электрон. журн.- Муром: Муромский институт (филиал) ВлГУ, 2011. - Режим доступа: http://amisod.ru, свободный.-ISSN 2220-878X (Online)

8. Симаков Р.А., Смяткин М.А. Особенности архитектуры распределенной массивно-реляционной СУБД. Алгоритмы, методы и системы обработки данных: Электронный научный журнал /под ред. С.С.Садыкова, Д.Е. Андрианова.- Вып. 18.- Электрон. журн.- Муром: Муромский институт (филиал) ВлГУ, 2011. - Режим доступа: http://amisod.ru, свободный.-ISSN 2220-878X (Online)

9. Смирнов А.В. Особенности распределения данных в многомерной СУБД SCIDB. Алгоритмы, методы и системы обработки данных: Электронный научный журнал /под ред. С.С.Садыкова, Д.Е. Андрианова.- Вып. 18.- Электрон. журн.- Муром: Муромский институт (филиал) ВлГУ, 2011. - Режим доступа: http://amisod.ru, свободный.-ISSN 2220-878X (Online)