

GENERATION OF DATA WITH ZIPFIAN DISTRIBUTION FOR DATABASE TESTING

Siniša Ilić

Faculty of technical sciences of University of Pristina, K. Mitrovica, Serbia

Dragana Radosavljević

*Faculty of technical sciences of
University of Pristina, K. Mitrovica*

Vidosav Stojanović

*Faculty of Electronics in Niš,
University of Niš, Serbia*

Abstract

It is often very hard to perform testing on real data in large databases. In order to execute testing of some performances on databases, data need to be generated synthetically by using distribution that suits to the real data the most. In many papers authors state that the data has been generated by using Zipfian distribution with appropriate z values, but they don't explain the process of generating such set of data, they just give pieces of complete scenario. We have collected those pieces and put some effort in comparison of results they have published with ours. In this paper we present the process of generating the synthetically data for testing of accuracy of approximate queries in large databases based on knowledge we have gathered from papers and we have discovered by ourselves.

Keywords: Zipfian distribution, cumulative distribution, synthetically generated data, database.

INTRODUCTION

It is often almost impossible to test database behavior in production environment with real data. In such conditions the large databases must handle a large number of users that connect to the database and execute their transactions, and additional testing may result in decreasing of performances. However, even outside the working hours, when the database workload is not large, or if there is no workload at all, the owners of data in database rarely allow data to be used for performance research, fearing that this data is not abused. For this purpose, one has to use synthetically generated data and perform various tests with them.

Usually for synthetically generated data, some mathematical distribution is used that matches the real data the best. One of such distributions is Zipfian distribution. According to the [1] the normalized formula of distribution is:

$$f(k; s, N) = \frac{1/k^s}{\sum_{n=1}^N (1/n^s)} \quad (1)$$

where N is the number of elements in distribution, k is the rank of element, s is the value of

the exponent characterizing the distribution. The origin of this distribution is from the studies of the frequencies of occurrence of words in a language. For example, word „the“ is the most frequently occurring word, and by itself accounts for nearly 7% of all word occurrences (69, 971 out of slightly over 1 million). The second-place is for word "of" that accounts for slightly over 3.5% of words (36,411 occurrences), followed by "and" (28,852) etc.

In large databases the often query is to get the sizes (frequencies) of occurring some values in some attribute(s). If the number of all distinct values in that attribute (column) is large and if the number of records in the table is also large (of order 10^6 or more) the time needed for obtaining the query result may be large too. The large table is usually stored on HDD in a number of pages. It requests a lot of I/O operations to read pages, transfer data from pages into RAM and to aggregate data. If RAM memory is not large enough to capture all pages, then the aggregation cannot be performed at once, but partially, so the time needed to obtain result is long. That is the reason why many commercial DataBase Management Systems (DBMS) create histograms or synop-

ses in small amount of RAM memory that show approximation of result of described query very quickly. The approximation does not give exact but rough result, with high accuracy, for quick respond. If one is interested in full accuracy, he could run full query that can be executed for a long time, but if approximation is good enough for quick estimation, the full query execution is not needed.

The testing of accuracy of approximate against the real results in approximate query processing [2, 3, 4, 5] is done on synthetical data that use the Zipfian distribution and modifications of Zipfian distribution.

BASIC CONCEPTS

As stated in [4], the set of all possible values that one can insert in a column of a table is called Domain with the size D . Usually, in real databases, it is very rare that one can find every value from Domain to be present in a column. The set of values that are present in a column is denoted as $\{v_1, v_2, \dots, v_N\}$ where $N < D$. The set of values should be presented in increased order, i.e. $v_1 < v_2 < \dots < v_N$. The result of query, that gets the number of occurrences (f_i) of values v_i from observed column, forms pairs $(v_1, f_1), (v_2, f_2), \dots (v_N, f_N)$. We can define spread as the difference between two consecutive values, that is $s_i = v_{i+1} - v_i$. There are $N-1$ spreads that we can calculate, and by definition N^{th} spread is equal to one.

The cumulative distribution of occurrences of the values can be defined as set of pairs $(v_1, f_1), (v_2, f_1+f_2), \dots (v_N, f_1+f_2+\dots+f_N)$ and cumulative + distribution can be defined as set of pairs of all values from Domain in increasing order and appropriate cumulative frequencies. In cumulative + distribution one can find a lot of values from Domain that don't have any occurrence in a column, but the cumulative frequency will be the same as for the first left neighbor value.

For example, if there are the values 8, 12, 8, 5, 12, 8, 5 in a column of a table, we may say that $v_1=5, v_2=8, v_3=12$. The spread distribution is: $s_1=8-5=3, s_2=12-8=4$ (formally there is also spread $s_3=1$). The distribution of the values is given by pairs (5, 2), (8, 3), (12, 2) and cumulative distribution is (5, 2), (8, 5), (12, 7). If we assume that Domain D is the set of integers that are less than or equal to 16 (i.e.

$D=16$), then the cumulative + distribution is defined by pairs (1, 0), (2, 0), (3, 0), (4, 0), (5, 2), (6, 2), (7, 2), (8, 5), (9, 5), (10, 5), (11, 5), (12, 7), (13, 7), (14, 7), (15, 7) and (16, 7) or simplified by the series: 0, 0, 0, 0, 2, 2, 2, 5, 5, 5, 5, 7, 7, 7, 7, 7. The first element of the pair is the position of the series' member and can be omitted. Our aim is to generate cumulative+ series as described by using Zipfian distribution.

ZIPFIAN DISTRIBUTION AND MODIFICATIONS OF ZIPFIAN DISTRIBUTION

The equation that is used for generation of synthetical data for database testing used in papers [2,3,4,5] is presented in [6]:

$$f_i = \frac{c_z}{i^z} \quad (2)$$

where c_z is an appropriate scalling constant, i is the rank of element and z is the value of the exponent characterizing the distribution. The normalization of equation (2) for probability distribution is given by:

$$\sum_{i=1}^U f_i = 1 \quad (3)$$

where U is the range, or universe size. The graphical representation of the Zipfian distribution for the universe $U=4096$, for $z=0.5$ (solid), $z=1$ (dotted) and $z=1.5$ (dashed) is presented at Figure 1.

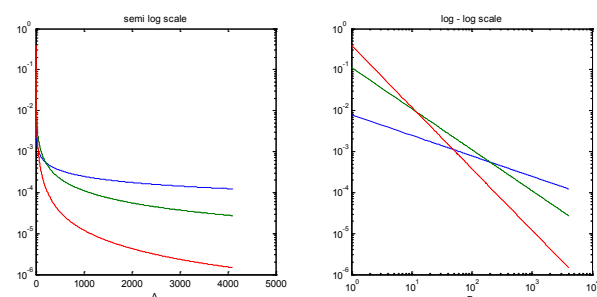


Fig. 1. Normalized Zipfian distribution with $z=0.5$, $z=1$ and $z=1.5$ presented in semi log scale (A) and log-log scale (B)

The z parameter values are chosen in the way to have data with a low skew ($z=0.5$, if $z \rightarrow 0$ distribution tends to be uniform), moderate skew ($z=1$) and high skew ($z=1.5$). As can be

seen from the Figure 1, the Zipfian distribution is decreasing function of the rank i . The modifications of the Zipfian distribution are:

- Increasing function (as the image of Zipfian function in the vertical mirror),
- Cusp_max function (first $N/2$ members create increasing and other $N/2$ members create decreasing function),
- Cusp_min function (first $N/2$ members create decreasing and other $N/2$ members create increasing function) and
- Zipfian function with randomly chosen positions (the regular Zipfian function with randomly assigned values $1, 2, \dots, N$)

GENERATING OF DATA USING THE ZIPFIAN DISTRIBUTION FOR TESTING OF DATABASES

Generating of data by using Zipfian distribution in papers [2, 3, 4, 5] is performed for spreads (that are used for generation of column values) and for frequencies of occurrences of column values. All mentioned distributions (decreasing, increasing, cusp_max, cusp_min and random) are used for spread generation, so we will generate 5 series of column values. For generation of distribution of frequencies the procedure is to create standard Zipfian distribution and to assign its values randomly to the values of values distribution in order to obtain pairs (value/frequency).

If the following parameters are known: Domain D , number of different column values, z parameter and distribution type, we can determine the column values within the domain.

Let's assume that the domain size D is 4096, the number of column values is $N=500$, parameter $z=1$ and the distribution type is cusp_max. It is obvious that we have to generate 499 spreads, from which the 250th spread has to be the largest, because it is placed in the middle of the domain range. We have to generate the series $1/i$, for i in the range 1-250. The generated series is decreasing function of i and it is the last sub-set (from 250-499) of main series of 499 members. The first member of this sub-set (which is in the middle of main set of 499 members) has to be divided by two in order to belong to second half of the main series (normally as the center member it belongs to both halves), and all values of this truncated sub-set is to be summed.

If the sum is denoted with S , then all values of the sub-set need to be multiplied with $D/2$ and divided by S in order to be normalized on $D/2$ (the half of Domain size). By rounding the normalized members of sub-set we will see that the sum of subset is not equal to $D/2$ but it is a little bit higher. In experimental way we have determined that if we have multiplied the members of sub-set with 2041 (that is lower than $D/2$) and divided with S (5.6007) we would get the normalized sum 2047. It means that the appropriate scaling constant c_z from the equation (2) should be 364.4203.

We have created the function **fliplr** (flip left-to-right) that rearranges the elements of a series in the way that first element becomes the last, the second becomes next-to-last etc. By using the **fliplr** function on described normalized subset, we have created the sub-set with increasing function of i with 250 members. We have connected increasing and decreasing sub-sets and deleted 251st element and multiplied 250th element by 2 (because we have divided this element by 2 in the beginning of the process). The resulting series is the spread series of 499 elements ($s_1, s_2, \dots, s_{499}$) that follows the cusp_max distribution. In order to get 500 values which spreads respect cusp_max distribution, we have created the **cumsum** function that gives 1 for the first value, $1+s_1$ for the second value, $1+s_1+s_2$ for the third value and so on. The last value that comes out from cumsum function is $1+s_1+s_2+\dots+s_{499}$. In this way we have created 500 values within the domain D whose spreads follow the cusp_max distribution.

We have created the series of the spreads with cusp_min distribution within the same domain and z parameter in the similar way. The construction algorithm differs from algorithm for cusp_max in the connection sequence (here the increasing sub-set follows the decreasing sub-set), the first element is multiplied by 2 (not the 250th) and the last (500th) element is deleted.

For series with decreasing and increasing function type of Zipfian distribution we have used the equation (2) with $i=1, 2, \dots, 499$, the parameter $c_z = 604.9031$ ($4109/6.7928$). By using these parameters, the sum of rounded spreads was 4095 ($D-1$). The increasing function has been obtained with help of **fliplr** function. We have created the function **rand(x)** that generates

series of x randomly chosen integer numbers in the range $1 - x$ without recurrence of any value. By using function `rand(499)` we have created series of 499 numbers as indexes of the standard Zipfian distribution and created randomly assigned positions i . For example, if the elements of the decreasing series are: 605, 302, 202, 151, 121, 101, 86, 76 and if the `rand(8)` returns the following numbers: 7, 5, 2, 1, 4, 8, 6, 3 then the randomly assigned series is: 86, 121, 302, 605, 151, 76, 101, 202.

The distributions of values that have been generated by using of modifications of Zipfian distribution for spreads generation are shown at Figure 2. The values obtained by using:

- standard Zipfian distribution are marked with '+',
- increasing function with square sign,
- cusp_min with 'X',
- cusp_max with circle sign and
- randomly assigned positions with diamond sign.

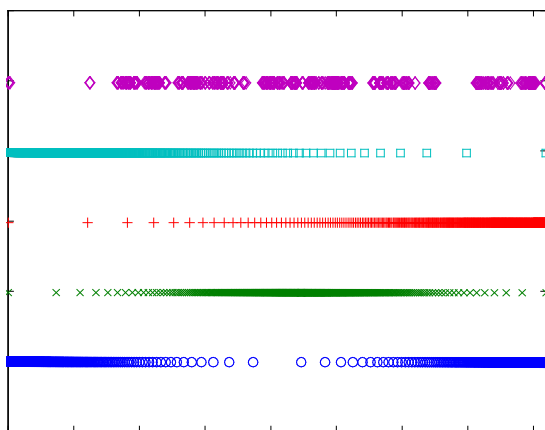


Fig. 2. The distributions of values generated by using of modifications of Zipfian distribution for spreads generation

Determining of column values is the first step in generation of the series that is needed for testing of database behavior needed. As we have mentioned at basic concepts, we need to obtain pairs (value, frequency) for each of the column values.

We have already explained how to obtain column values and we need to describe the procedure for obtaining the respective frequencies. The frequency distribution is obtained by using standard Zipfian distribution and by taking into account that the sum of elements of distribution needs to be equal to the number of records in the table. The generated frequencies are then assigned to column values randomly.

After generating of (value, frequency) pairs, we have generated cumulative pairs (as described in basic concepts). Cumulative + series has

been generated from the cumulative series by adding pairs of values that belong to domain and don't belong to column values and frequencies that are equal to frequencies of left cumulative pairs. At the end we have obtained the series needed that consist of cumulative + frequencies by omitting column values from the (value, cumulative frequency) pairs. For example, if the (value/frequency) pairs are: (1, 605), (2, 76), (4, 202), (7, 151), ... then the cumulative pairs are: (1, 605), (2, 681), (4, 883), (7, 1034), ... and the respective cumulative + series is: 605, 681, 681, 883, 883, 883, 1034, etc.

In order to obtain cumulative + series from randomly generated numbers for random assigning of frequencies to column values, distribution of values, distribution of frequencies and domain size D , we have created the function **generateCum**(indx, val_distr, freq_distr, domainSize), where `indx=rand(N)`. The size of vectors `indx`, `val_distr` and `freq_distr` must be the same (N).

The algorithm of `generateCum` function is presented at Figure 3.

By combining of 5 values distributions, one frequency distribution with parameter $z=0.5$ which cumulative sum is 100,000 and 5 different randomly chosen `indx` values, we have generated 5 cumulative + series that can be used for database testing. These series are presented at Figure 4. The series obtained by using `cusp_max` distribution shown at the figure 4 is the same as in the figure 2 of [4].

```

Funct generateCum(indx, val_distr, freq_distr, domainSize)
check the sizes of indx, val_distr and freq_distr
if not the same exit funct;

size = size(indx);
series(1 to domainSize) = 0;
seriesPosition = val_distr(1)+1;
series(val_distr(1)) = freq_distr(1);

for i = 2 to size
    // copies the same freq. values up to the next column value
    while val_distr(i) > seriesPosition
        series(seriesPosition) = series(seriesPosition-1);
        seriesPosition = seriesPosition+1;

    // set the next cumulative frequency for column value
    series(seriesPosition) = series(seriesPosition-1) +
        freq_distr(indx(i));
    seriesPosition = seriesPosition+1;

if i = size
    // copies the same freq. values up to the domain size
    while domainSize > seriesPosition
        series(seriesPosition) = series(seriesPosition-1);
        seriesPosition = seriesPosition+1;

next i;
return series;

```

Fig. 3. The algorithm of `generateCum` function that builds cumulative + series

The benefit of creating the cumulative series against the pairs of (value/frequency) is that the series can be treated as one-dimensional signal and a lot of signal processing tools can be used.

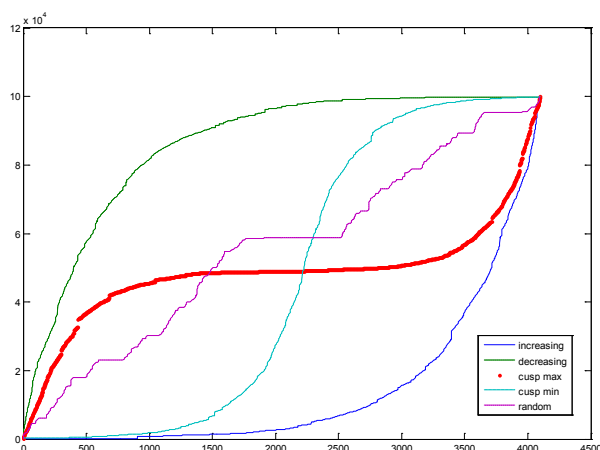


Fig. 4. cumulative + series of data generated by using of 5 modified Zipfian distributions

CONCLUSION

Testing of some performances is not possible on large databases in production environment. In such cases the creation of database with synthetically generated data would be a decent solution.

There is a list of modifications of Zipfian distributions that can be used for testing of accuracy of approximate query results and simulate the database with real data.

We have precisely described in this paper the process of data generation by using mentioned

distributions and presented generated values of attribute observed.

It is also presented the algorithm for final generation of cumulative+ series that is mostly used for testing of accuracy of fast selectivity in approximate queries of large tables. By using cumulative + series, the accuracy is obtained very simple - as the root mean square (RMS) of the sum of differences between generated series' members and its approximation members divided by number of members.

REFERENCES

- [1] http://en.wikipedia.org/wiki/Zipf's_law
- [2] Y. Ioannidis, V. Poosala "Histogram-Based Approximation of Set Valued Query Answers", in Proceedings of the 25th VLDB Conference, Edinburgh, Scotland, 1999.
- [3] V. Poosala, Y. Ioannidis, P. Haas and E. Shekita: "Improved Histograms for Selectivity Estimation of Range Predicates", in Proceedings of 1996 ACM SIGMOD International Conference on Management of Data, Montreal, Canada, 1996.
- [4] Y. Matias, J.S.Vitter and M.Wang: "Wavelet Based Histograms for Selectivity Estimation", in Proceedings of 1998 ACM SIGMOD 26th International Conference on Management of Data, Seattle, Washington, USA, 1998.
- [5] K. Chakrabarti, M.Garofalakis, R.Rastogi and K.Shim: "Approximate query processing using wavelets", The VLDB Journal, volume 10, pages 199-223, 2001.
- [6] G. Cormode and S.Muthukrishnan: "Summarizing and Mining Skewed Data Streams", in SIAM Conference on Data Mining, Newport Beach, California, USA, 2005.