Probabilistic Counters

Joaquim Madeira

Version 0.1 – November 2017

Overview

- Motivation
- Counting with probability 1 / 2
- Counting with probability 1 / 2^k
- Counting with decreasing probability
- Other kinds of probabilistic counters

Motivation

- Is it possible to use a small counter to keep approximate counts of large numbers ?

- Use a large number of such counters to keep track of the number of occurrences of many different events
  - E.g., 8-bit counters

- Morris, Approximate Count Algorithm, 1978

Motivation

- But, nowadays memory is no longer scarce…
- Is such an approach still interesting ?
- Yes !!
- Massive data volumes !!
- Need quick and memory-efficient processing

Application areas

- Online social networks
- Large-scale scientific experiments
- Search engines
- Online content delivery
- Product and consumer tracking
- ….

- Data too large to fit in memory must be analyzed !!

Application areas

- System performance monitoring and diagnosis
  - Detecting excessively high rates of various system events
- Statistics counters
  - Used to count events that may occur with high frequency
  - BUT, counter values are read infrequently

Big-Data

- Medical data
  - Genetic sequences, time series, …
- Activity data
  - GPS location, social network activity, …
- Business data
  - Costumer behavior tracking, …
- …

Big-Data – Scale up vs Downsize

- Scale up the computation
  - Replicate cheap hardware / devices
  - Build massive DBMSs and warehouses
  - …
  - BUT, expensive equipment / energy !!
- Downsize the data
  - Compact representations of large data sets
  - Approximate answers
  - Probabilistic methods

Data streaming

- Data arrives in a streaming fashion
- Must be processed on the fly

- Packets in network traffic monitoring
- Queries arriving at a Web service
- …

- Make just one pass over the data
  - Use memory that is sublinear on the amount of data

Probabilistic Counters – Goal

- Avoid using “large” counters when dealing with large data volumes !!
- A counter with n bits counts up to 2^n events
- Can we use less bits ?
- What is the cost ?

1^{st} Method

- For each event, increment the counter with probability 1 / 2
- Intuition: just incrementing for half of the events !!
- We can now count up to 2^{n+1} events
  - Using just n bits !!
- Is that what happens ?
- Draw the state diagram / triangular diagram

1^{st} Method – Tasks

- Simulate such a counter for 10, 100, 1000 and 10000 events
  - Repeat the experiments several times !
  - What can you conclude ?

- How to evaluate the accuracy ?
  - Relative error or accuracy ratio
  - When knowing the exact value…

Counting 100 events – 10000 trials

```
counter value:      43 -      277 times -  2.770%
counter value:      44 -      373 times -  3.730%
counter value:      45 -      471 times -  4.710%
counter value:      46 -      599 times -  5.990%
counter value:      47 -      688 times -  6.880%
counter value:      48 -      715 times -  7.150%
counter value:      49 -      788 times -  7.880%
counter value:      50 -      836 times -  8.360%
counter value:      51 -      757 times -  7.570%
counter value:      52 -      733 times -  7.330%
counter value:      53 -      681 times -  6.810%
counter value:      54 -      565 times -  5.650%
counter value:      55 -      551 times -  5.510%
counter value:      56 -      398 times -  3.980%
counter value:      57 -      295 times -  2.950%
```

1st Method – Expected value (mean)

- Counter is a random variable
  - Resulting from a succession of random events
- What is the expected value after k events ?
- X_i represents the i^{th} increment
  - $X_i = 1$: counter is incremented
  - $X_i = 0$: counter is not incremented
  - $P[\, X_i = 0 \,] = P[\, X_i = 1 \,] = 1 / 2$

1st Method – Expected value (mean)

- $E[\, X_i \,] = 0 \times P[\, X_i = 0 \,] + 1 \times P[\, X_i = 1 \,] = 1 / 2$
- Counter value after k events is

$$S = \sum X_i$$

- $E[\, S \,] = E[\, \sum X_i \,] = \sum E[\, X_i \,] = k / 2$
- Number of events can be estimated by $2 \times S$

1st Method – Variance

- $\sigma^2(X_i) = E[X_i^2] - \{ E[X_i] \}^2 = E[X_i^2] - 1 / 4$
- $E[X_i^2] = 0^2 \times P[X_i = 0] + 1^2 \times P[X_i = 1] = 1 / 2$
- $\sigma^2(X_i) = 1 / 4$
- $\sigma^2(S) = \sigma^2 (\sum X_i) = \sum \sigma^2 (X_i) = k / 4$
- Standard deviation: $\sigma (S) = \sqrt{k} / 2$

1^{st} Method – Tasks

- Simulate such a counter for 10, 100, 1000 and 10000 events
  - Repeat the experiments many times !!
- For each counter, compute the mean, variance and standard deviation of the experimental results
- Compare with the theoretical results !

Counting 100 events – 10000 trials

```
Expected value: 50.000
Variance: 25.000
Standard deviation: 5.000

Mean Absolute Error: 3.914
Mean Relative Error: 7.828%
Mean Accuracy Ratio: 100.061%

Smallest counter value: 33
Largest counter value: 68

Mean counter value: 50.031
Mean absolute deviation: 3.917
Standard deviation: 4.905
Maximum deviation: 17.969
Variance: 24.055
```

Classical Summary Statistics

- Provide a summary of the essential features of a dataset

- To enable answering simple questions
  - What are typical values ?
  - How much variation is in the data ?
  - …

- Robustness against outliers !

Mean value

- Mean value or average of a dataset

$$\mu(X) = \frac{1}{n}\sum_{i=1}^{n} x_i$$

```
def mean(X): return sum(X) / len(X)
```

- If the x_i values are close together, the mean is a good representation of a typical sample

Deviation measures

- Deviation of individual samples from the mean value
- Maximal deviation

$$maxdev(X) = max\{|x_i - \mu|, i = 1,2, \dots, n\}$$

- Mean absolute deviation

$$mad(X) = \frac{1}{n}\sum_{i=1}^{n}|x_i - \mu|$$

Deviation measures

- Standard deviation

$$stddev(X) = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i - \mu)^2}$$

- In general

$$mad(X) \leq stddev(X) \leq maxdev(X)$$

- Outliers heavily affect these deviation measures

1^{st} Method – Probability distribution

- After n events, what is the probability of the counter value being k ?
  - p (n, k) = ?

- Example for n = 4
  - More probable / Less probable counter values ?
  - p (4, k) = ?

- Binary table / Binary tree / Pascal-like triangle

1^{st} Method – Probability distribution

- Probability of incrementing the counter : p
- Probability of not incrementing : (1 – p)
- Probability of the counter value being k after n events:

$$p(n,k) = \binom{n}{k} p^k \, (1 - p)^{n-k}$$

  - Bernstein-basis polynomials – Features ?

- Check the results of the previous example !

1st Method – Probability distribution

- What if we want to compute several p(n,k) ?
- For large values of n and k…
- Avoid computing factorial values…
- Avoid computing successive powers…
- We have already seen how to do that !!

Computing Bernstein Polynomials

$B_{0,0}(t) = 1$

$B_{n,0}(t) = (1 - t)\ B_{n-1,0}(t)$

$B_{n,n}(t) = t\ B_{n-1,n-1}(t)$

$B_{n,j}(t) = (1 - t)\ B_{n-1,j}(t) + t\ B_{n-1,j-1}(t)\ ;\ j = 1, 2, \ldots, n - 1$

t in [0,1]

- Compute $B_{n,j}(t^*)$ using a 2D array

1^{st} Method – Tasks

- For n = 10, 100, … events compute the probability distributions for the possible counter values
- Compute the respective mean and variance
- Compare with
  - The theoretical values
  - The obtained experimental values

Probability Distribution – p = 1 / 2

```
p(   10,    0) =  0.09765625000 %
p(   10,    1) =  0.97656250000 %
p(   10,    2) =  4.39453125000 %
p(   10,    3) = 11.71875000000 %
p(   10,    4) = 20.50781250000 %
p(   10,    5) = 24.60937500000 %
p(   10,    6) = 20.50781250000 %
p(   10,    7) = 11.71875000000 %
p(   10,    8) =  4.39453125000 %
p(   10,    9) =  0.97656250000 %
p(   10,   10) =  0.09765625000 %
```

Probability Distribution – p = 1 / 2

```
p(   100,    40) =   1.084386671164 %
p(   100,    41) =   1.586907323654 %
p(   100,    42) =   2.229226954657 %
p(   100,    43) =   3.006864264421 %
p(   100,    44) =   3.895255978910 %
p(   100,    45) =   4.847429662643 %
p(   100,    46) =   5.795839814030 %
p(   100,    47) =   6.659049999098 %
p(   100,    48) =   7.352701040671 %
p(   100,    49) =   7.802866410508 %
p(   100,    50) =   7.958923738718 %
p(   100,    51) =   7.802866410508 %
p(   100,    52) =   7.352701040671 %
p(   100,    53) =   6.659049999098 %
p(   100,    54) =   5.795839814030 %
p(   100,    55) =   4.847429662643 %
p(   100,    56) =   3.895255978910 %
p(   100,    57) =   3.006864264421 %
p(   100,    58) =   2.229226954657 %
p(   100,    59) =   1.586907323654 %
p(   100,    60) =   1.084386671164 %
```

Generalization

- Can we approx. count the same number of events using less bits ?
- Or approx. count more events using the same number of bits ?
- Yes ! Increment the counter with lesser probability
- Increment with probability $1 / 2^k$

Generalization – Tasks

- Incrementing with probability $1 / 2^k$
- Obtain an expression for the mean, the variance and the stdr. deviation after n events
  - $k = 2, 3, \ldots, 6, \ldots$
- Analyze the corresponding probability distributions
  - Pascal-like triangle

Generalization – Mean and Variance

- Probability of incrementing the counter: p
- $q = (1 - p)$

- It is not difficult to check that, after n events:

- $E[S] = n \times p$

- $\sigma^2(S) = n \times p \times q$

Generalization – Tasks

- Set the counting probability to 1 / 32
- Simulate such a counter for 10, 100, 1000 and 10000 events
- Compute the mean, variance and standard deviation of the experimental results
- Compare with the theoretical results !

Counting 100 events – 10000 trials

```
counter value:      0 -     391 times -  3.910%
counter value:      1 -    1256 times - 12.560%
counter value:      2 -    2215 times - 22.150%
counter value:      3 -    2282 times - 22.820%
counter value:      4 -    1817 times - 18.170%
counter value:      5 -    1118 times - 11.180%
counter value:      6 -     551 times -  5.510%
counter value:      7 -     237 times -  2.370%
counter value:      8 -      91 times -  0.910%
counter value:      9 -      23 times -  0.230%
counter value:     10 -      17 times -  0.170%
counter value:     11 -       2 times -  0.020%
```

Counting 10000 events – 10000 trials

```
counter value:     305 -     210 times -  2.100%
counter value:     306 -     204 times -  2.040%
counter value:     307 -     215 times -  2.150%
counter value:     308 -     221 times -  2.210%
counter value:     309 -     237 times -  2.370%
counter value:     310 -     211 times -  2.110%
counter value:     311 -     225 times -  2.250%
counter value:     312 -     197 times -  1.970%
counter value:     313 -     234 times -  2.340%
counter value:     314 -     249 times -  2.490%
counter value:     315 -     242 times -  2.420%
counter value:     316 -     244 times -  2.440%
counter value:     317 -     230 times -  2.300%
counter value:     318 -     220 times -  2.200%
counter value:     319 -     217 times -  2.170%
counter value:     320 -     190 times -  1.900%
counter value:     321 -     189 times -  1.890%
counter value:     322 -     203 times -  2.030%
counter value:     323 -     190 times -  1.900%
counter value:     324 -     158 times -  1.580%
```

Generalization – Tasks

- For n = 10, 100, 1000, … events compute the probability distributions for the possible counter values

- Compute the respective mean and variance

- Compare with the obtained experimental results

Probability Distribution – p = 1 / 32

```
p(   100,     0) =  4.17995447166 %
p(   100,     1) = 13.48372410213 %
p(   100,     2) = 21.53046267920 %
p(   100,     3) = 22.68801443615 %
p(   100,     4) = 17.74788226054 %
p(   100,     5) = 10.99223675491 %
p(   100,     6) =  5.61431447159 %
p(   100,     7) =  2.43200719046 %
p(   100,     8) =  0.91200269642 %
p(   100,     9) =  0.30073207193 %
p(   100,    10) =  0.08827941466 %
p(   100,    11) =  0.02329955225 %
p(   100,    12) =  0.00557435524 %
p(   100,    13) =  0.00121722893 %
p(   100,    14) =  0.00024400672 %
p(   100,    15) =  0.00004512812 %
p(   100,    16) =  0.00000773365 %
p(   100,    17) =  0.00000123268 %
```

Probability Distribution – p = 1 / 32

```
p(10000,  305) =  2.112048727272 %
p(10000,  306) =  2.158582375174 %
p(10000,  307) =  2.198728332976 %
p(10000,  308) =  2.232119159148 %
p(10000,  309) =  2.258450661912 %
p(10000,  310) =  2.277486510363 %
p(10000,  311) =  2.289061745195 %
p(10000,  312) =  2.293085116748 %
p(10000,  313) =  2.289540205200 %
p(10000,  314) =  2.278485305915 %
p(10000,  315) =  2.260052091458 %
p(10000,  316) =  2.234443089605 %
p(10000,  317) =  2.201928043120 %
p(10000,  318) =  2.162839241381 %
p(10000,  319) =  2.117565935387 %
p(10000,  320) =  2.066547965775 %
p(10000,  321) =  2.010268747734 %
p(10000,  322) =  1.949247766912 %
p(10000,  323) =  1.884032746247 %
p(10000,  324) =  1.815191645304 %
```

Fixed Probability Counters – Recap

- For each event, increment the counter with probability $1 / 2^k$, for $k >= 1$
- On average, just incrementing for $1 / 2^k$ of the events !!
- Number of events estimated by 2^k x Counter
- We can now count up to 2^{n+k} events
  - Using just n bits !!

Issues

- What happens when counting a small number of events with probability 1 / 32 ?
- For much larger numbers of events, can we be more economical ?

Approximate Counting – Binary Base

- Morris, 1978 – For an arbitrary counting base

- As the counter value increases, it will be incremented with lesser probability

- If counter has value k
  - Increment it with probability $1 / 2^k$
  - Do not increment it with probability $(1 – 1 / 2^k)$

- Draw the state diagram !

Approximate Counting – Binary Base

- On average, how many events, n, are needed to reach a counter value of k ?

- What does k represent ?

Events	Counter value	Number of events
X	1	1
X	…	…
…	…	…

- Let's do it on the board !

Approximate Counting – Binary Base

- Counter is a random variable
- What is the expected value after n events ?
- X_i represents the i^{th} increment
  - $X_i = 1$: counter is incremented
  - $X_i = 0$: counter is not incremented
  - $P[X_i = 0] = 1 – 1 / 2^{i-1}$
  - $P[X_i = 1] = 1 / 2^{i-1}$

Approximate Counting – Binary Base

- $E[X_i] = 1 / 2^{i-1}$
- Counter value after n events is

$$S = \sum X_i$$

- $E[S] = E[\sum X_i] = \sum E[X_i]$

$$E[S] = 1 + 1/2 + 1/2 + 1/4 + 1/4 + \ldots$$

Approximate Counting – Binary Base

- BUT, we only store integer values !!

Number of events	E[S]	Expected counter value
1	1	1
3	1 + 1 / 2 + 1 / 2	2
7	1 + 2 x 1 / 2 + 4 x 1 / 4	3
15	1 + 2 x 1 / 2 + 4 x 1 / 4 + 8 x 1 / 8	4
…	…	…

- How to estimate the number of events from the counter value ?

Approximate Counting – Binary Base

- After $n = 2^k - 1$ events the expected counter value is k
- $k = \log_2 (n + 1) = \text{floor}(\log_2 n) + 1$

- Generalize !
- After n events the expected counter value is $\text{floor}(\log_2 (n + 1))$

- Logarithmic counter !!
  - For larger values, it counts "slower"

Approximate Counting – Binary Base

- After n probabilistic updates, the counter contains an approximation of log n

- That value is stored in log log n bits !!

Approximate Counting – Binary Base

- How to estimate the number of events from the counter value k ?
  - Compute $2^k - 1$

- How to evaluate the counter's accuracy ?
  - Compare with floor($\log_2 (n + 1)$)

- What is the largest value that we can count with a 4-bit or 8-bit or 16-bit counter ?

Tasks

- Simulate such a counter for 10, 50, 100, 500, 1000, 10000 events
  - Repeat the experiments many times !
- For each counter, compute the mean, variance and standard deviation of the experimental results
- What can you conclude ?

Counting 10000 events – 10000 trials

```
Smallest counter value: 10
Largest counter value: 16

Mean counter value: 13.009
Mean absolute deviation: 0.622
Standard deviation: 0.875
Maximum deviation: 3.009
Variance: 0.766

        counter value:      10 -       2 times -   0.020%
        counter value:      11 -     260 times -   2.600%
        counter value:      12 -    2518 times -  25.180%
        counter value:      13 -    4539 times -  45.390%
        counter value:      14 -    2247 times -  22.470%
        counter value:      15 -     413 times -   4.130%
        counter value:      16 -      21 times -   0.210%
```

Approximate Counting – Binary Base

- After n events, what is the probability of the counter value being k ?
  - p (n, k) = ?

- Example for n = 4
  - More probable / Less probable counter values ?
  - p (3, k) = ?
  - p (4, k) = ?

- Binary tree / Pascal-like triangle

Approximate Counting – Binary Base

- Recurrence
  - $p(1, 1) = 1$ and $p(1, 0) = 0$
  - $p(n, 1) = 1 / 2 \times p(n - 1, 1)$
  - $p(n, n) = 1 / 2^{n-1} \times p(n - 1, n - 1)$
  - $p(n, k) = 1 / 2^{k-1} \times p(n - 1, k - 1) + (1 - 1 / 2^{k}) \times p(n - 1, k)$

Tasks

- For n = 10, 50, 100, … events compute the probability distributions for the possible counter values
- Compute the respective mean and variance
- Analyze the obtained results

Approx. Counting – Arbitrary Base

- For some applications the expected error of the previous method might be too large !

- How to improve the counter performance ?

- If counter has value k
  - Increment it with probability $1 / a^k$
  - Do not increment it with probability $(1 – 1 / a^k)$

- a is now the counter base

Approx. Counting – Arbitrary Base

- Take $a < 2$

- The counter value after m increments will be larger than with the binary base

- Giving a better accuracy !!

- Probabilities can be stored in a table
  - No need to be recomputing !!

Approx. Counting – Arbitrary Base

- Possible values ?
  - $a = 2^{1/2}, 2^{1/4}, \ldots$
- How to estimate the number of events from the counter value k ?
  - Compute $(a^k – a + 1) / (a – 1)$
- What is the largest value that we can count with a 4-bit or 8-bit or 16-bit counter ?

Tasks

- Simulate such a counter, with a = $2^{1/2}$, for 10, 50, 100, 500, 1000, 10000 events
  - Repeat the experiments many times !
- For each counter, compute the mean, variance and standard deviation of the experimental results
- What can you conclude ?

Counting 10000 events – 10000 trials

```
Smallest counter value: 20
Largest counter value: 29

Mean counter value: 23.781
Mean absolute deviation: 0.985
Standard deviation: 1.229
Maximum deviation: 5.219
Variance: 1.510

        counter value:      20 -      18 times -  0.180%
        counter value:      21 -     212 times -  2.120%
        counter value:      22 -    1178 times - 11.780%
        counter value:      23 -    2780 times - 27.800%
        counter value:      24 -    3104 times - 31.040%
        counter value:      25 -    1925 times - 19.250%
        counter value:      26 -     643 times -  6.430%
        counter value:      27 -     122 times -  1.220%
        counter value:      28 -      17 times -  0.170%
        counter value:      29 -       1 times -  0.010%
```

Approximate Floating-point Counter

- Csurös, 2010
- Binary floating-point counter
- d-bit significand and a binary exponent
- d + log log n bits

Approximate Floating-point Counter

- $M = 2^d$, d is a non-negative integer
- Counter X, initialized to $X = 0$

FP-Increment(X) *// returns new value of X*
 set $t \leftarrow \lfloor X/M \rfloor$ *// bitwise right shift by d positions*
 while $t > 0$ **do**
  if RandomBit() $= 1$ **then return** X
  set $t \leftarrow t - 1$
 return $X + 1$

Approximate Floating-point Counter

- First d steps are deterministic !!
  - Accurate count for smaller values
- For $d = 0$, we have $M = 1$ and it is Morris' counter !!

Approximate Floating-point Counter

- Counter value $X = 2^d \times t + u$ is used to estimate the actual count
  - u denotes value of the lower d bits
- Estimate is $(M + u) \times 2^t - M$

Tasks

- Simulate such a counter for 10, 50, 100, 500, 1000, 10000 events
  - Repeat the experiments many times !
- What can you conclude ?

Other approaches

- Flajolet & Martin, 1985
- Approximate counting the number of different elements in a multi-set
- Analyze the tail bits of hash values
- …

One recent paper from 2016

Adding Approximate Counters

Guy L. Steele Jr.
Oracle Labs
guy.steele@oracle.com

Jean-Baptiste Tristan
Oracle Labs
jean.baptiste.tristan@oracle.com

Abstract

We describe a general framework for adding the values of two approximate counters to produce a new approximate counter value whose expected estimated value is equal to the sum of the expected estimated values of the given approximate counters. (To the best of our knowledge, this is the first published description of any algorithm for adding two approximate counters.) We then work out implementation details for five different kinds of approximate counter and provide optimized pseudocode. For three of them, we

and a *read* operation that observes a counter value k returns $2^k - 1$ as a statistical estimate of the actual number of times the *increment* operation has been performed.

Morris furthermore provided a generalization of this algorithm as well as a statistical analysis. The probabilistic decision made by the *increment* operation can rely on the output of a random (or pseudorandom) number generator, and as Morris observes, "The random number generator can be of the simplest sort and no great demands are made on its properties." Flajolet [5] provided a de-

References

- R. Morris, Counting Large Numbers of Events in Small Registers, *Commun. ACM*, Vol. 21, N. 10, October 1978
- P. Flajolet, Approximate Counting: A Detailed Analysis, *Bit*, Vol. 25, 1985
- M. Csurös. Approximate counting with a floating-point counter. In *COCOON*, LNCS vol. 6196, p. 358-367, Springer, 2010