Large-scale Data Processing on the Cloud

MTAT.08.036

Lecture 8: Streaming and Parallel R

Briti Deb

deb@ut.ee
22-10-2014
Overview

• Introduction to Streaming Algorithm
• Streaming Model
• Sampling Techniques
• Sketching Techniques
• Parallel R
Overview

• Introduction to Streaming Algorithm
• Streaming Models
• Sampling Techniques
• Sketching Techniques
• Parallel R
Why Streaming Algorithm and Parallel R?

Science  
Social Networks  
Web Analytics

Telecommunication

Sensors (Internet of Things)
- Environmental Monitoring (weather, tsunami, earthquake, wildlife)
- Industrial Applications (industrial process control, supply chain network)
- Medical and Healthcare Systems (wearable health monitoring)
- Building and Home Automation (smart home)
- Transport Systems (railway tracks)

REQUIRE VISUALIZATION AND ANALYTICS OVER DATA STREAM

4/47
Why Streaming Algorithm and Parallel R?

User need interactive **visualization** and advanced **analytics**
But no tools available on the Hadoop stack offers this

R language provides visualization and analytics
But (standard) R is in-memory and single threaded

Interfacing R with Hadoop would bring parallel processing capability to R

Stream data require novel algorithms given memory and time constraints (Hadoop initially designed for batch processing)
Streaming Idea

Store a lossy compressed representation of the data and deliver user queries from this subset instead of the entire set.
Limitations of Hadoop-Mapreduce

- Analysis of large data sets often requires powerful **distributed data stores** like Hadoop and **parallel data processing** techniques like MapReduce.

- Hadoop approach (**batch processing**) often leads to heavyweight **high-latency** analytic processes and poor applicability to **real time** use cases.

- Stream processing eases computation of metrics like a number of unique visitors, most frequent items, ...

- Trade **precision** of the estimations for **memory** consumption.
Limitations of Streaming Algorithm

- Produces an approximate answer (in contrast to accurate answering algorithms)

- Must deliver an accuracy with a reasonably high probability bound

- Probability of an error can be quantified and the programmer can trade off the expected error rate with the amount of resources (space, time)

- Order invariant
Online Algorithm vs Streaming Algorithm

Similarities
● Decisions made before all data available
● Works on sub-linear space (algorithm cannot see all data)
● Provides approximate answer

Differences
● Streaming algorithms – can defer action until a group of items arrive
● Online algorithms – must take action as soon as each item arrives

● Online algo eg: Perceptron, Reservoir Sampling
● Streaming algo eg: sketching

Offline Algorithm
● Whole data is given in the beginning

● Selection sort requires the entire list be given before it can sort it, while insertion sort doesn't. Is insertion sort an online algorithm?
Revision - Asymptotic Notation

f(n) = O(g(n)) means f(n) is less than or is bounded by some constant multiple of g(n)

If \( f(n) \leq c \cdot g(n) \)
when \( n \geq n_0 \), \( c > 0 \), \( n_0 \geq 1 \).

Then \( f(n) = O(g(n)) \)

Let \( f(n) = 3n + 2 \) and \( g(n) = n \).

Can we say? Necessary condition

\( f(n) = O(g(n)) \)
\( f(n) \leq c \cdot g(n) \)
\( c > 0 \)

\( 3n + 2 \leq cn \)
\( c = 4 \)
\( 3n + 2 \leq 4n \)
\( m \geq 2 \)

Big O gives the worst case or upper bound, i.e., time will not exceed this; i.e., \( f(n) \) is upper-bounded by \( cg(n) \)

Streaming Algorithms are Sub-linear Algorithms

Sub-linear Algorithm: An algorithm whose time / space requirement f(n) grows slower than the size of the problem n (gives an approximate answer)

Note: A sub-linear algorithm doesn't have the time / space to consider all the input; it samples the input

Overview

- Introduction to Streaming Algorithm
- Streaming Models
- Sampling Techniques
- Sketching Techniques
- Parallel R
Data Stream

**Constraints:**

1) Items can only be read sequentially in the order it arrives (no random access)

2) Items can be examined in only a few passes (typically just one)

3) Limited working memory (much less than the input size), sublinear in n and m

4) Limited processing time per item

Existing (deterministic) models are too computationally heavy for this type of problems
Data Stream

Stream: Let \( <a_1, a_2, ..., a_m> \) be a stream of items drawn from universe \([n] = \{1, 2, ..., n\}\).

Update: j-th update of item \(<item, c[j]>\) implying \(item = item + c[j]\); (\(c[j]\) is count[j], can be >0 or <0)

- Streaming models specify different update methods.

Goal: Process a large stream in a smaller memory (s). Compute a function over the stream, eg: median, number of distinct elements, longest increasing sequence, etc.

s needs to be sublinear in m and n. s stores a constant number of items from the stream and a constant number of counters that can count up to the length of the stream.

\[s = o(\min\{m, n\})\]

More specifically,

\[s = O(\log m + \log n)\]
Cash Register Model

- Arrivals-Only Streams
- $c[j]$ is always $> 0$
- Typically, $c[j]=1$

- Example: <item1, 3>, <item2, 2>, <item1, 2> encodes the arrival of
  - 3 copies of item1
  - 2 copies of item2
  - 2 copies of item1

- Could represent packets in a network
Turnstile Model

- Stream updates can be arrivals or departures
- $c[j]$ can be $>0$ or $<0$

Example:
- $<\text{item1}, 3>, <\text{item2}, 2>, <\text{item1}, -2>$ encodes final state of $<\text{item1}, 1>, <\text{item2}, 2>$
- Can represent fluctuating quantities

Other Streaming Models

- In **Sliding Window model**, the function of interest is computed over a fixed-size window in the stream. As the stream progresses, items from the end of the window are removed from consideration while new items from the stream take their place.

- In **Time-Series model**, only x-th update is processed i.e., item = c[x]

- Cash Register, Turnstile, Sliding Window, and Time Series models are models for **frequency-based problems**

- Besides the above frequency-based problems, **graph problems** can also be addressed by streaming the adjacency matrix or the adjacency list of the graph
References

Overview

- Introduction to streaming algorithm
- Streaming Models
- **Sampling Techniques**
- Sketching Techniques
- Parallel R
Sampling

- Technique for tackling data stream by sampling items from the stream

- **Application:** Compute the median packet size of some IP packets by sampling and use the median of the sample as an estimate for the true median

- **Problem:** How to find uniform sample $s$ from a stream of unknown length $N$?

- **Simple solution:** Guess $N$. Obtain random numbers of size $s$ (where $s<N$), and sample those indexes from $N$

- **Drawbacks:** Incorrect $N$, need for multiple pass, unevenly distributed sample
Reservoir Sampling

- After reservoir is full, replace items in reservoir based on probability.
- Sample from input stream \(<a_1, a_2, \ldots, a_m>\) a single item \(j\) uniformly at random in a reservoir of size \(k\) (\(k << m\)).
- So that the probability of being the sample is the same for every input item.
- Hence, we require \(P[a_j \text{ is the sample } s] = k/j\) for \(1 \leq j \leq m\).
- And process the \(j^{th}\) element (\(j > k\)) whether it will replace an item already in reservoir or not.
- Every element (>k) belongs to the reservoir with same probability (proof later).
Reservoir Sampling

Problem: Find uniform sample from a stream of unknown length

Algorithm:

```
array R[k];  // Reservoir array of size k
integer i, j;

// Fill the reservoir with the first k items of stream S
for each i in 1 to k do
    R[i] := S[i]
done;

// Replace items in reservoir
for each i in k+1 to length(S) do
    j := random(1, i);  // Generate a random number between 1 and i
    if j <= k then
        R[j] := S[i]
    fi
done
```

Why it works

- Lets prove that every element belongs to the reservoir (size k) with the same probability
- Let $x_i$ be the i-th element and $S_i$ be the reservoir elements after seeing first i elements
- We will prove by induction that $P[x_j \in S_i] = k / i$ for all $j \leq i$ with $k \leq i \leq n$
- Which implies that the probability that any element is in the reservoir is $k/n$
- In the base case $i=k$ and the first k elements are in reservoir with probability 1
- Lets look at the i-th element for some $i>k$. This element will enter the reservoir $S_i$ with probability $k/i$
- On the other hand, for any of the elements $j<i$, it will be in $S_i$ only if it was in $S_{i-1}$ and is not removed by the i-th element
- So $P[x_j \in S_{i-1}] = k/(i-1)$, whereas the probability that $x_j$ is not removed by the current element is $(1-1/i) = (i-1)/i$
- We can say that $P[x_j \in S_i] = k/(i-1) \cdot (i-1)/i = k / i$
Revision - Map Reduce

- **WordCount Problem** - Parallel Map Tasks takes I/P, tokenizes words, and O/P <word, 1>

- **Map-1 O/P**
  - < Hello, 1>
  - < World, 1>
  - < Bye, 1>
  - < World, 1>

- **Map-2 O/P**
  - <Hello, 1>
  - <World, 1>
  - <Bye, 1>
  - <World, 1>

- **Parallel Reduce Tasks**
  - Reducer-1 groups I/P by keys and O/P <key, count>
    - <Hello, 2>
    - <Bye, 2>
  - Reducer-2 groups I/P by keys and O/P <key, count>
    - <World, 4>
Distributed Reservoir Sampling

- Let's take the problem of having a number of files where the sum of all the lines in all the files is $n$ (does not fit in memory) and we have to sample $k$ of these lines.

- How to solve it distributively?

- Each machine takes some fraction of the input to process and generate their own reservoir sample from their subset of the data.

- Then, a final process takes the output of all reservoirs and merges them using the original keys.

- E.g., if one of the 10 machines processed only 10 items and the other 9 machines each processed 1 million items, it is expected that the one machine with 10 items would likely have smaller keys and hence be less likely to be selected in the final output. If the keys are recomputed in the final process, then all of the input items would be treated equally when they shouldn't.
Distributed Reservoir Sampling

- Pick a random sample of size $k$ which is equivalent to generating a random permutation (ordering) of the items. Pick the top $k$ items.

- The ordering is computed distributedly with each mapper associating a (distinct) random id with each item and keeping track of the top $k$ items (as opposed to the whole dataset).

- Binary min heap data structure can be used to keep track of the top $k$ elements.

http://had00b.blogspot.com/2013/07/random-subset-in-mapreduce.html
Distributed Reservoir Sampling

- Initialize a binary min heap with the first $k$ elements, and when a new element comes, associate a random id with it.

- If that id is larger than the smallest id in heap then replace it with the new id.

- Top $k$ elements of each mapper are sent to a single reducer which extracts the top $k$ elements among all.

- The amount of data sent out by the map phase is reduced to the top $k$ elements of each mapper as opposed to the whole dataset.

- Hadoop framework present the values to the reducer in order of keys from lowest to highest.

- Using the negation of the id as key, the first $k$ element read by the reducer will be the top $k$ elements that the user wants to find.
#!/usr/bin/python
# rand_subset_m.py

import sys, random
from heapq import heappush, heappop

k = int(sys.argv[1])
H = []

for x in sys.stdin:
    r = random.random()
    if len(H) < k:
        heappush(H, (r, x))
    elif r > H[0][0]:
        heappop(H)
        heappush(H, (r, x))

for (r, x) in H:
    # by negating the id, the reducer receives
    # the elements from highest to lowest
    print '%f\t%s' % (-r, x),
Reducer

```python
# The Reducer simply returns the first k elements received
#!/usr/bin/python
# rand_subset_r.py

import sys

k = int(sys.argv[1])
c = 0

for line in sys.stdin:
    (r, x) = line.split(' \t ', 1)
    print x,
    c += 1
    if c == k: break
```
Other Sampling Techniques

**Drawbacks of Reservoir Sampling**
- Hard to parallelize
- Difficult to problems like counting how many distinct items in the stream because large fraction of items may not be sampled

**Priority Sampling**

**Minwise Hashing**
References

* http://had00b.blogspot.com/2013/07/random-subset-in-mapreduce.html
* http://blog.cloudera.com/blog/2013/04/hadoop-stratified-randsampling-algorithm/
Overview

- Introduction to streaming algorithm
- Streaming Models
- Sampling Techniques
- Sketching Techniques
- Parallel R
Difference between Sampling and Sketching

- Sample sees only those items which were selected to be in the sample; whereas the sketch sees the entire input, but is restricted to retain only a small summary of it.

- There are queries that can be approximated well by sketches that are provably impossible to compute from a sample (because large fraction of items may not be sampled).
Revision - Hashing

- Hash function maps values from a *domain* to its *range*
- It takes a **key** and outputs an index number for its **value**
- Ideally hash function assign each key to a unique value
- Collision can occur, can be addressed by chaining with linked list
- Time for hash operations (add, delete, find) is the time to find the key in the bucket (which is constant) plus the time for the linked list operation

\[
U = \{1, 2, \ldots, n\} \text{ is our universe and let } V = \{0, 1, 2, \ldots, q - 1\} \text{ be a set with } q \leq n. \text{ A family of hash functions } \mathcal{H} \text{ from } U \text{ to } V \text{ is said to be 2-universal if, for all } x_1, x_2 \in U \text{ with } x_1 \neq x_2, \text{ and for } h \text{ chosen uniformly at random from } \mathcal{H} \text{ we have}
\]
\[
Pr[h(x_1) = h(x_2)] \leq \frac{1}{q}.
\]
Count Min Sketch

- Suppose we want to maintain page view counts for millions of unique URLs.

- To store the set of all the $M = \langle \text{item, count} \rangle$ using an associative array that maps every item to its corresponding count would need huge memory.

- This problem can be addressed by count min sketch using a property of pairwise independent hash function / 2-universal hash function that they do not collide too frequently.

- A list of most viewed pages can be maintained by pseudo-random function such as 2-universal hash function / pairwise-independent hash function.
Count Min Sketch

- Model input data stream as vector
  
  Where initially

- The $t^{th}$ update is $(i_t, c_t)$

\[
\alpha(t) = (a_1(t), \ldots, a_i(t), \ldots a_n(t))
\]

\[
a_i(0) = 0 \quad \forall i
\]

\[
a_{i_t}(t) = a_{i_t}(t - 1) + c_t
\]

A Count-Min (CM) Sketch Matrix with parameters $(\varepsilon, \delta)$ is represented by a two-dimensional array (a small summary of input) counts with width $w$ and depth $d : \text{count}[1,1] \ldots \text{count}[d, w]$

Given parameters $(\varepsilon, \delta)$, set

\[
w = \left\lceil \frac{\varepsilon}{\delta} \right\rceil \quad \text{and} \quad d = \left\lceil \ln \frac{1}{\delta} \right\rceil
\]

where $(\varepsilon, \delta)$ are the error of estimation and the probability of estimation respectively.

$d$ hash functions are chosen uniformly at random from a pairwise independent family which map vector entry $[1, \ldots, n]$ to $[1 \ldots w]$, i.e.,

\[
h_1, \ldots, h_d : \{1 \ldots n\} \to \{1 \ldots w\}
\]

---

**Initialize**

1. $d$;
2. $w$;
3. $C[1 \ldots d][1 \ldots w] \leftarrow 0$;
4. Pick $d$ independent hash functions $h_1, h_2, \ldots, h_d : [n] \to [w]$, each from a 2-universal family;

**Process $(j, c)$:**

5. for $i = 1$ to $d$ do
6. \[C[i][h_i(j)] \leftarrow C[i][h_i(j)] + c;\]

**Output**

- On query $a$, report $\hat{f}_a = \min_{1 \leq i \leq d} C[i][h_i(a)]$

---

Count Min Sketch

- At any point in time if we want to know the approximate value of an item $a_i$, we can get it by computing the minimum of all values in each of the $d$ rows where $a_i$ hashes to.

- The general intuition is that since we are using hash functions, there's always a possibility of multiple $i$'s colliding on to the same cell and contributing additive to the value of the cell.

- Hence the minimum among all hash values is the closest candidate to give the correct result for a query.
Other Sketching Techniques

● **Bloom Filter**

● **Counting Sketches**
Applications

- **Google Chrome** use Bloom filter to represent a blacklist of dangerous URLs
  - Each time a user is about to navigate to new page, the corresponding URL is hashed and compared to a local Bloom filter that represents the set of all malicious URLs

- **Twitter's algebird** offer implementations of Count-Min sketch

- Also in **web analytics, internet marketing**
  - Find most frequent items in a stream
  - Estimate rare values in a stream
  - Estimate number of distinct elements in a stream
References

- http://lkozma.net/blog/sketching-data-structures/#comment-85892
- http://research.neustar.biz/tag/count-min-sketch/
- https://tech.shareaholic.com/2012/12/03/the-count-min-sketch-how-to-count-over-large-keyspaces-when-about-right-is-good-enough/
- http://java.dzone.com/articles/count-min-sketch-data
- https://tech.shareaholic.com/2012/12/03/the-count-min-sketch-how-to-count-over-large-keyspaces-when-about-right-is-good-enough/
- http://java.dzone.com/articles/count-min-sketch-data
- http://www.cs.yale.edu/homes/el327/datamining2013aFiles/03_frequent_items_in_streams.pdf

Overview

- Introduction to streaming algorithms
- Streaming Models
- Sampling Techniques
- Sketching Techniques
- Parallel R
Why do we need Parallel R?

**R**

Programming language with modeling and visualization facilities

Works on data as in-memory, stand alone, mostly single threaded

**Hadoop**

Distributed file system (HDFS) and parallel processing framework (MapReduce)

Perform job scheduling, data handling

Written in Java and originally designed to help write Java programs

Interfacing R with Hadoop would bring parallel processing capability to R
R can be used for data processing on Hadoop using **Hadoop streaming**

Hadoop streaming is an utility that comes with the Hadoop distribution, and allows to run Map/Reduce jobs with **any** script as the mapper/reducer

**RHadoop** package is one way to to use R on Hadoop

**RHadoop** consists of several packages such as ‘**rmr**’, ‘**rhdfs**’

**rmr** contains the functions for moving data in and out of a Hadoop cluster and executing Map/Reduce jobs on the cluster

Processing Stream Data using RMR

- Users clicks on different parts of screen while using browser or application
- These click data known as clickstream can be analyzed to understand user behavior
- As an example, a clickstream data may contain URL, Timestamp, IP Address, User ID, etc
- Using such data, statistical models (e.g., linear regression) can be built for prediction
- rmr can be used to process clickstream data and predictive models can be built using the \texttt{lm} function in the \texttt{stats} package
Linear regression models the relationship between a dependent variable $y$ and one or more explanatory variables $X$

For example, given a data set $\{y_i, x_{i1}, x_{i2}, \ldots, x_{ip}\}$ where $p$ is the number of explanatory variables and $i = 1 \ldots n$, a linear model assumes a linear relationship between dependent variable $y_i$ (to be predicted) and independent variables $X_i$ and is modeled as:

$$ y_i = b_0 + b_1 x_{i1} + \ldots + b_p x_{ip} + \text{error} $$

The goal is to find the best estimate of the coefficients $b_p$ by minimizing the residual error between the experimental and predicted signal.

In R, the syntax for regression analysis is `lm(y ~ model)` where model is the formula for the chosen mathematical model.
References

- http://hadoop.apache.org/docs/r1.2.1/streaming.html
- https://github.com/hortonworks/hadoop-tutorials/blob/master/Community/T01_RHadoop_visitors_prediction.md
- Foundations of Data Science, John Hopcroft, Ravindran Kannan (ebook available free online)
- http://www.cse.iitk.ac.in/users/sganguly/workshop.html
- Data Streams: Algorithms and Applications S. Muthukrishnan (2005)
- Joseph Adler, R In a Nutshell, 2ed, Oreilly (2012)
Summary

- Introduction to Streaming Algorithms
- Streaming Model
  - Cash Register Model
  - Turnstile Model
- Sampling Techniques
  - Reservoir Sampling
- Sketching Techniques
  - CountMin Sketch
- Parallel R

Streaming – Shift from batch processing / persistent database paradigm
Fundamental re-thinking of models, assumptions, algorithms, system, ...