Data streaming fundamentals

Giuseppe Fiameni

g.Fiameni@cineca.it
• Data streaming
• Data models for data streaming
• Approximation and reduction
• Technologies
Overview

• Data Streaming is real-time / unbounded data processing
• Analysis continuous stream of events like heartbeats, ocean currents, machine metrics, GPS signals, etc.
• Data streams require *online mining*, in which we wish to mine the data in a continuous manner. In many data mining situations, we know the entire data set in advance
• Pose special challenges to a number of data mining algorithms because of the fact that the data in the streams may show temporal correlations.
Traditional DBMS

2 Query → Query Processing
1 Data → Main Memory

3 Query results → Disk

DSMS

Data → Continuous Query

Query results → Main Memory
Characteristics of Data Streams

• Data Streams
  – Data streams - continuous, ordered, changing, fast, huge amount
  – Traditional DBMS - data stored in finite, persistent data sets

• Characteristics
  – Huge volumes of continuous data, possibly infinite
  – Fast changing and requires fast, real-time response
  – Data stream captures nicely our data processing needs of today
  – Random access is expensive, single scan algorithm (can only have one look)
  – Store only the summary of the data seen thus far
  – Most stream data are at pretty low-level or multi-dimensional in nature, needs multi-level and multi-dimensional processing
Challenges of Stream Data Processing

• Multiple, continuous, rapid, time-varying, ordered streams
• Main memory computations
• Queries are often continuous
  – Evaluated continuously as stream data arrives
  – Answer updated over time
• Queries are often complex
  – Beyond element-at-a-time processing
  – Beyond stream-at-a-time processing
  – Beyond relational queries
• Multi-level/multi-dimensional
StreamSQL

```sql
SELECT AVG(price) FROM stream
[SIZE 10 ADVANCE 1 TUPLES]
WHERE value > 100.0
```

Data Models

• **Real-time data stream:**
  – sequence of items that arrive in some order and may only be seen once.

• **Stream items:** like relational tuples
  – Relation-based
  – Object-based

• **Window models**
  – Direction of movements of the endpoints: fixed window, sliding window, landmark window
  – Time-based vs. Tuple-based
  – Update interval:
    • eager (for each new arriving)
    • lazy (batch processing)
Data windows

Sliding:

Jumping:

Overlapping

(adapted from Jarle Søberg)
DATA REDUCTION
Data Reduction Techniques

- **Aggregation**: approximations e.g., mean or median
- **Load Shedding**: drop random tuples
- **Sampling**: only consider samples from the stream (e.g., random selection). Used in sensor networks.
- **Sketches**: summaries of stream that occupy small amount of memory, e.g., randomized sketching
- **Wavelets**: hierarchical decomposition
- **Histograms**: approximate frequency of element values in stream
Many things are hard to compute exactly over a stream
- Is the count of all items the same in two different streams?
- Requires linear space to compute exactly

**Approximation:** find an answer correct within some factor
- Find an answer that is within 10% of correct result
- More generally, a \((1 \pm \varepsilon)\) factor approximation

**Randomization:** allow a small probability of failure
- Answer is correct, except with probability 1 in 10,000
- More generally, success probability \((1-\delta)\)

**Approximation and Randomization:** \((\varepsilon, \delta)\)-approximations
Probabilistic Guarantees

- Use *Tail Inequalities* to give probabilistic bounds on returned answer
  - *Markov Inequality*
  - *Chebyshev Inequality*
  - *Chernoff Bound*
  - *Hoeffding Bound*
Basic Tools: Tail Inequalities

• General bounds on *tail probability* of a random variable (that is, probability that a random variable deviates far from its expectation)

• **Basic Inequalities**: Let $X$ be a random variable with expectation $\mu$ and variance $\text{Var}[X]$. Then, for any $\varepsilon > 0$

  **Markov:**
  \[
  \Pr(X \geq (1+\varepsilon)\mu) \leq \frac{1}{1+\varepsilon}
  \]

  **Chebyshev:**
  \[
  \Pr(|X - \mu| \geq \mu\varepsilon) \leq \frac{\text{Var}[X]}{\mu^2\varepsilon^2}
  \]
Not every problem can be solved with sampling

- Example: counting how many distinct items in the stream
  - If a large fraction of items aren’t sampled, don’t know if they are all same or all different

Other techniques take advantage that the algorithm can “see” all the data even if it can’t “remember” it all

“Sketch”: essentially, a linear transform of the input

- Model stream as defining a vector, sketch is result of multiplying stream vector by an (implicit) matrix
Count-Min Sketch [Cormode, Muthukrishnan’04]

• Simple sketch idea, can be used for as the basis of many different stream mining tasks
  — Join aggregates, range queries, moments, ...
• Model input stream as a vector $A$ of dimension $N$
• Creates a small summary as an array of $w \times d$ in size
• Use $d$ hash functions to map vector entries to $[1..w]$

Array: $CM[i,j]$
• Each entry in input vector \( A[] \) is mapped to one bucket per row
  – \( h() \)'s are pairwise independent
• Merge two sketches by entry-wise summation
• Estimate \( A[j] \) by taking \( \min_k \{ CM[k,h_k(j)] \} \)
STREAM FREQUENT PATTERN ANALYSIS
What Is Frequent Pattern Analysis?

• Frequent pattern: A pattern (a set of items, subsequences, substructures, etc.) that occurs frequently in a data set

• Motivation: Finding inherent regularities in data
  – What products were often purchased together? — Beer and diapers?!
  – What are the subsequent purchases after buying a PC?
  – What kinds of DNA are sensitive to this new drug?

• Mining precise freq. patterns in stream data: unrealistic

• How to mine frequent patterns with good approximation?
  – Lossy Counting Algorithm (Manku & Motwani, VLDB’02)
    • Major ideas: not tracing items until it becomes frequent
• **Step 1:** Divide the incoming data stream into windows.
• **Step 2:** Increment the frequency count of each item according to the new window values. After each window, decrement all counters by 1.
• **Step 3:** Repeat – Update counters and after each window, decrement all counters by 1.
TOOLS
There are multiple open source data stream tools, following are the open source tools available...

- Apache Spark
- Apache Flink
- Apache Samza
- Apache Storm
- Apache Kafka
- Apache Flume
- Apache Nifi
- Apache Ignite
- Apache Apex
- Apache Beam
Apache Spark

- Apache Spark is a lightning-fast cluster computing technology, designed for fast computation.
- It is based on Hadoop MapReduce and it extends the MapReduce model to efficiently use it for more types of computations, which includes interactive queries and stream processing.
- The main feature of Spark is its in-memory cluster computing that increases the processing speed of an application.
- Spark is designed to cover a wide range of workloads such as batch applications, iterative algorithms, interactive queries and streaming.
Apache Storm

• Storm makes it easy to reliably process unbounded streams of data, doing for real-time processing
• Storm supports many use cases like real-time analytics, online machine learning, continuous computation, ETL, and more.
• Storm is fast, a benchmark clocked it at over a million tuples processed per second per node.
• It is scalable, fault-tolerant, guarantees your data will be processed, and is easy to set up and operate.
• A Storm topology consumes streams of data and processes those streams in arbitrarily complex ways, repartitioning the streams between each stage of the computation however needed.