Strainer

Windowing-based Advanced Sampling in Stream Processing Systems

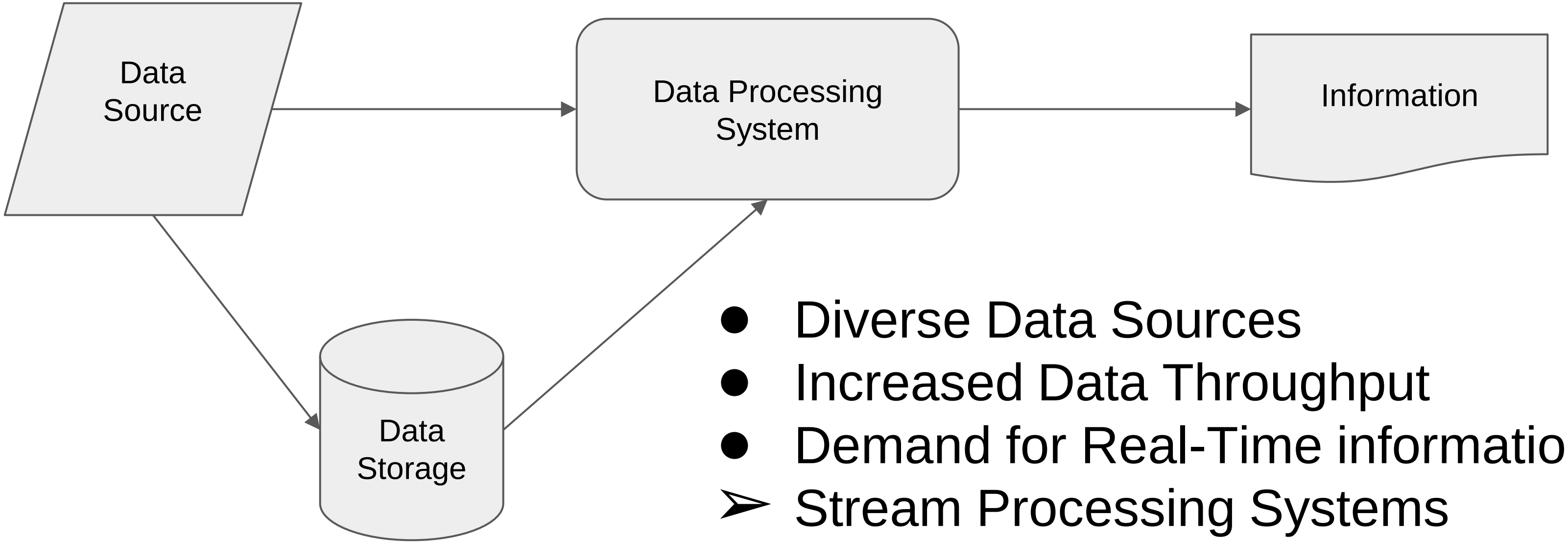
Nikola Koevski, Sérgio Esteves, Luís Veiga

GECON 2024 - 20th International Conference on the Economics of Grids, Clouds, Systems, and Services
Rome, Italy, 2024/09/27

Agenda

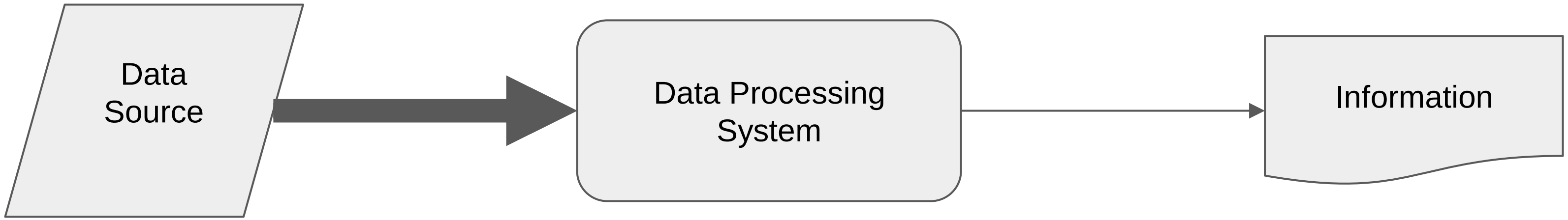
- Introduction
- Architecture
- Implementation details
- Evaluation
- Conclusions
- Future work

Introduction



- Diverse Data Sources
- Increased Data Throughput
- Demand for Real-Time information
- ➢ Stream Processing Systems

Motivation



On Data Throughput spike:

- Performance degradation
- Accuracy deterioration

- Solutions:
  - Scale resources
  - Reduce ingested data

Related work

Controlled data reduction - Approximate computing systems:

- Load shedding - multi-point reduction
  - Aurora
  - Borealis
  - Apache Shark
- Sampling - single-point reduction
  - BlinkDB system
  - ApproxHadoop
  - IncApprox

Current Shortcomings

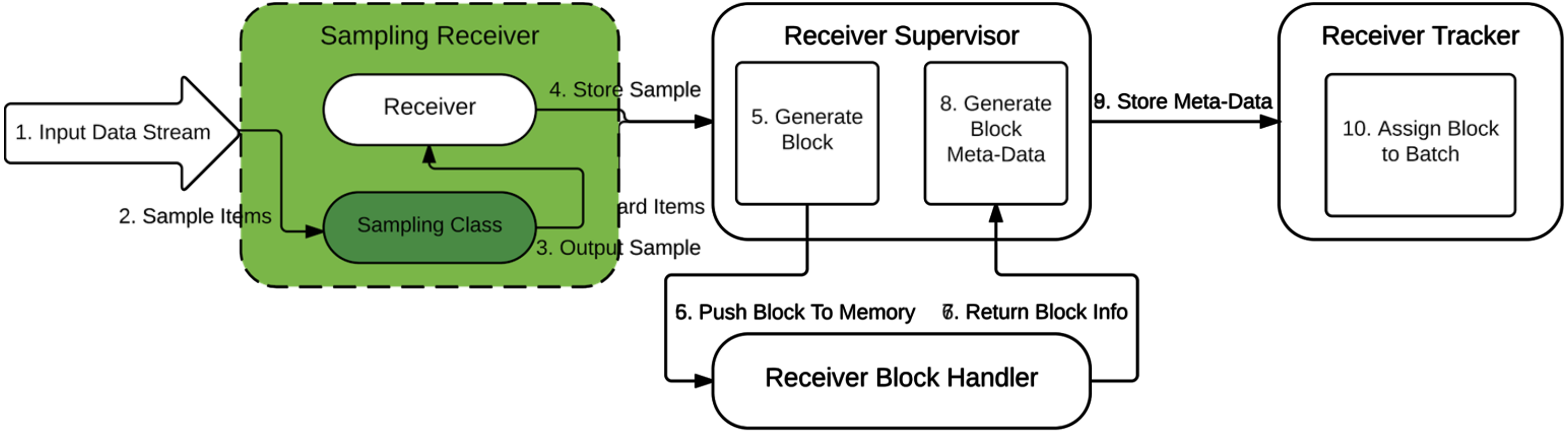
- Load Shedding
  - Large operator tracking data structures
  - Operator error recalculation overhead
  - Ignore data distribution
- Multiple points of data reduction
  - More invasive to the system
  - Higher error probability
- Sampling over stored data
- Systems are not distributed

Goals

Implementation of a sampling framework

- Use a well-established processing platform
- Provide a single-point, user-friendly API
- Implement advanced sampling techniques
- Show performance & accuracy gains

Architecture



- Using Spark Streaming from Apache Spark
- Spark Streaming uses a Batching module
- Sampling framework inserted into the Batching module
- Rest of the system is unchanged

Architecture

Sampling Algorithms

Selection criteria:

1. Provide a fixed-size sample with a single pass over the data. (Gibbons and Matias, 1998)
2. Prevent data distribution skew when sampling. (Chaudhuri et al., 2001).
3. Provide accuracy guarantees for the sampled subset.
4. Provide timeliness guarantees for the sampling algorithm.

- ➢ Decision: A Reservoir-scheme sampling algorithm
    - ➢ implementing a biased sampling technique with a bounded error and low computing overhead.

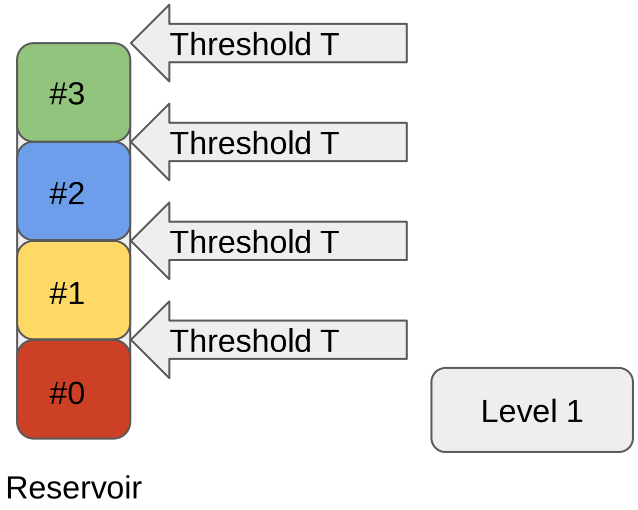
Architecture

Algorithm 1: Distinct Value sampling (Gibbons, 2001)

- Reservoir Scheme
- Attribute-to-Integer hash mapping
- 0-10% bounded error
- Low space requirement

Parameters

- Target attribute
- Sample size
- Threshold
- Domain size



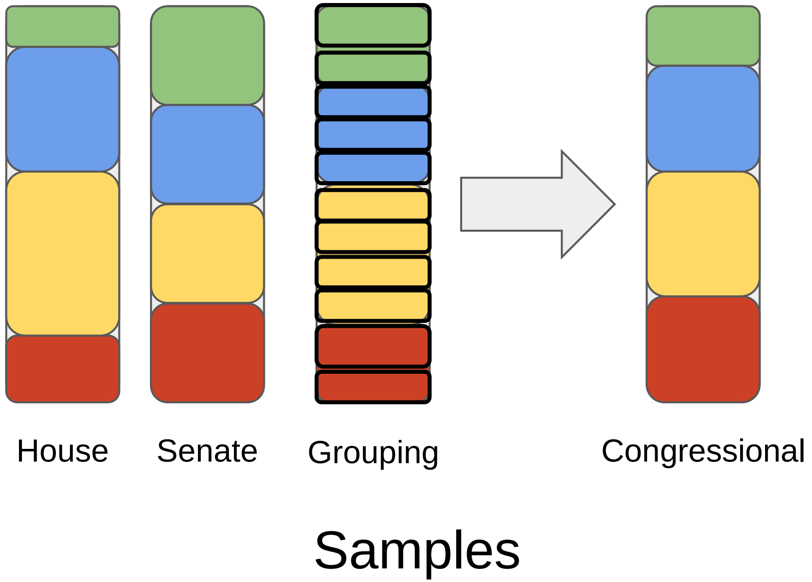
Architecture

Algorithm 2: Congressional sampling (Acharya et al., 2000)

- Reservoir scheme
- Hybrid reservoir-biased method
- Low, < 10% error

Parameters

- Group-by attributes
- Sample size



Implementation

- Spark Streaming Receiver extension
  - Integration with StreamingContext
  - Accommodate sampling classes
  - Override Receiver Supervisor communication
  - JobGenerator-inspired recurring timer
  - Batch Interval coordinated sampling
  - StreamingListener-enabled coordination

- Interface for easy implementation of new one-pass sampling algorithms

Evaluation

Experimental Setup

- 8-core, 2.93GHz Intel i7, 12GB RAM; 64-bit Ubuntu
- Apache Spark
- 1800 items/sec ingestion rate
- Spark’s monitoring capabilities
- JvmTop for Heap memory usage
- Relative error for sample error

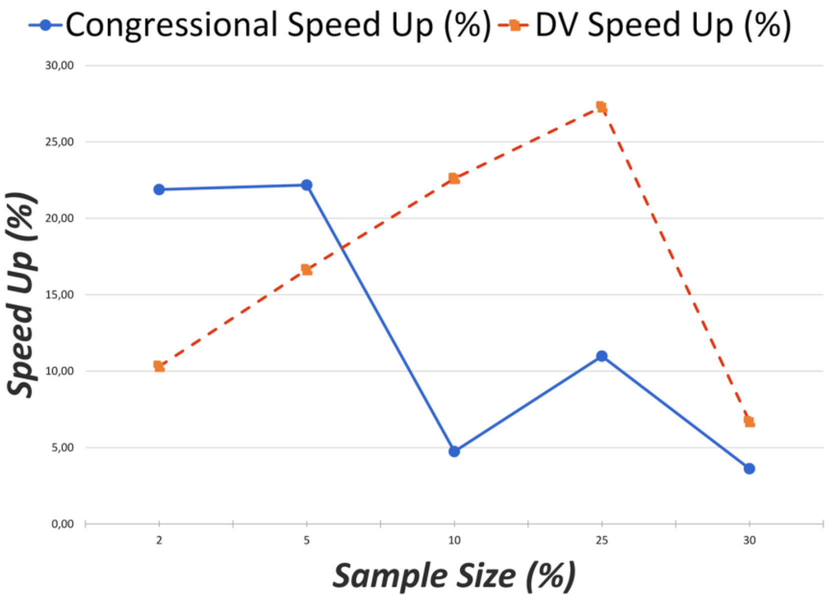
Metrics

- Qualitative
  - Sample Error
- Quantitative
  - Execution time Speed Up
  - Memory Variation

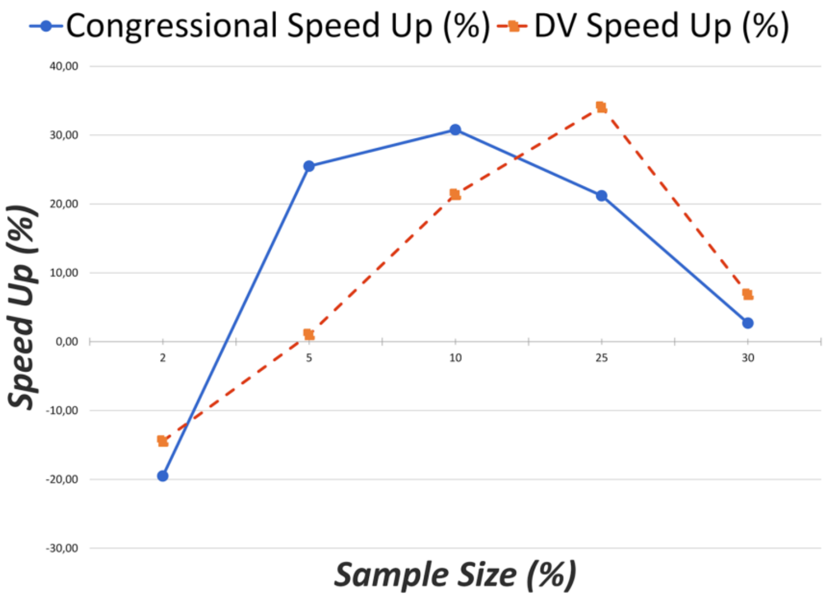
Benchmark Applications

- Apple NASDAQ Tweets
- US IT Stock
- Online Retail
- NY Taxi Logs

Execution Time Speed Up



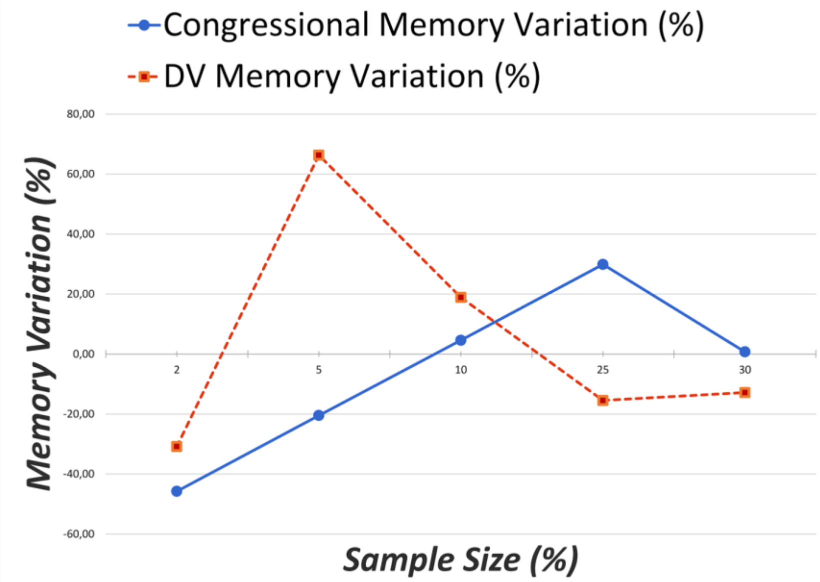
Apple Tweets



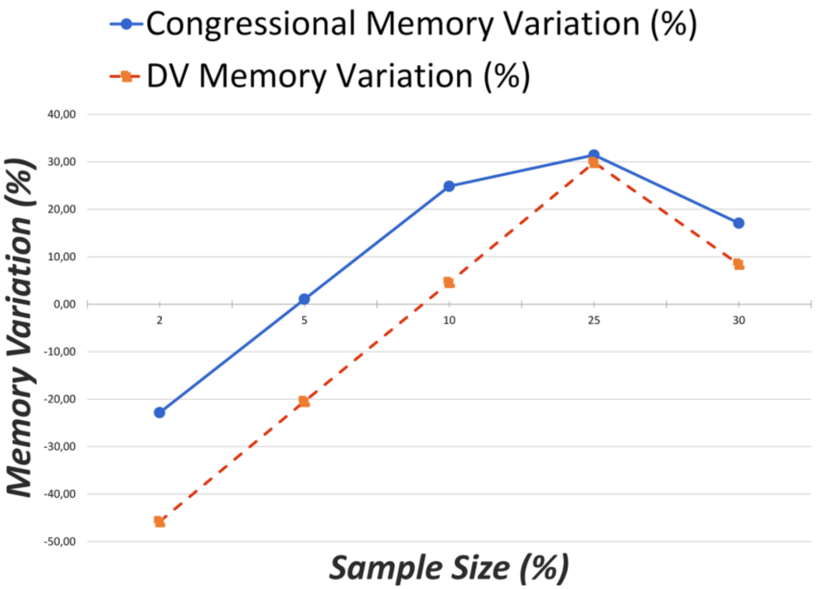
US IT Stocks

- Best speed up times 5-25% sample interval
- Maximum speed up 34%
- Distinct Value sampling better at higher sample sizes

Memory Variation



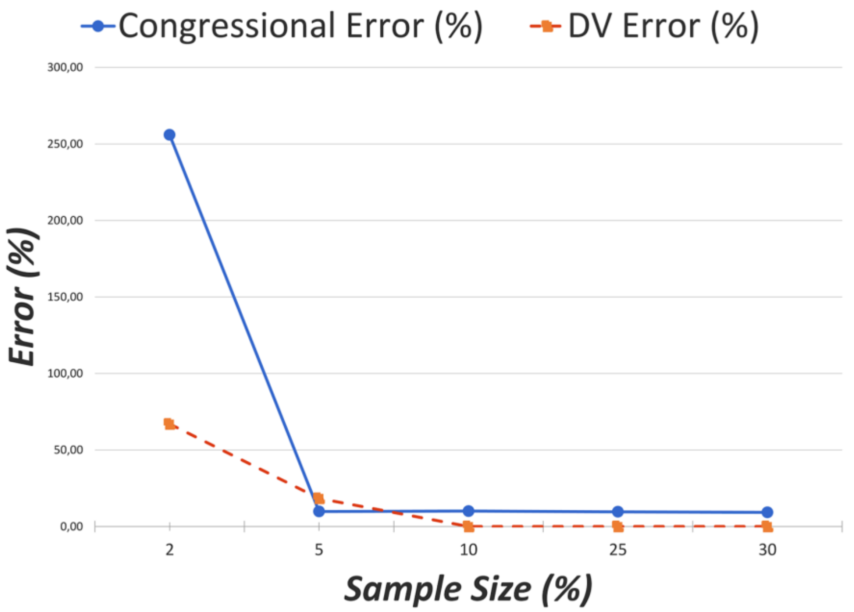
Online Retail



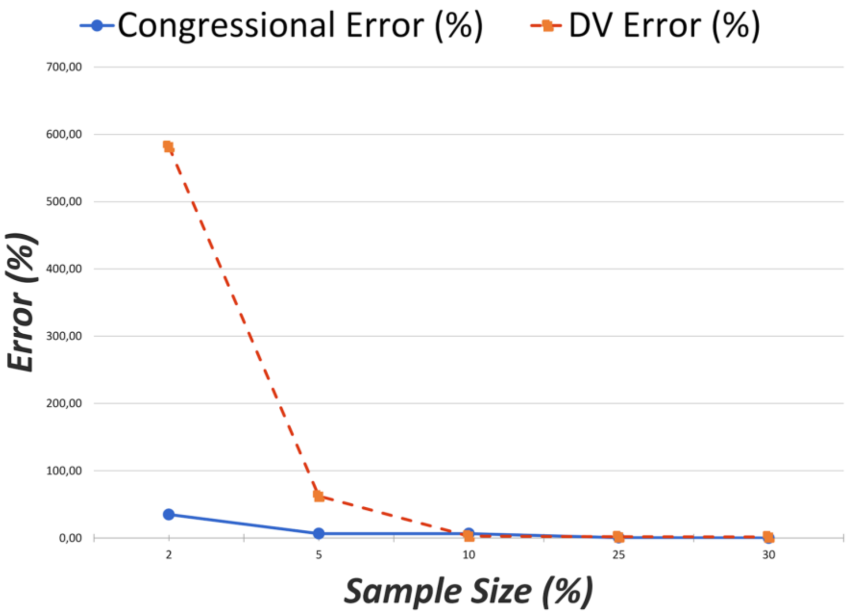
NY Taxi Logs

- Additional memory usage overall
- Up to 45% memory reduction
- DV sample - less memory at larger sample sizes

Sample Error



Online Retail



US IT Stocks

- Low overall error
- Exception for 2% sample
  - Frequent re-sampling

Conclusions

- Approximate computing system with advanced sampling techniques
- Modular design
- User-transparent
- Easy integration with Spark
- Up to 34% faster execution time
- Can provide up to 45% less memory consumption
- Bounded, <10% error rate in typical *well-behaved* workloads

Future work

- Module for heavy load detection and automatic shift from normal to sampled execution
- (Intelligently) Defining QoS thresholds for error and accuracy
- Self-adjusting sample size according to error and accuracy

Thank you for your attention.

Questions?