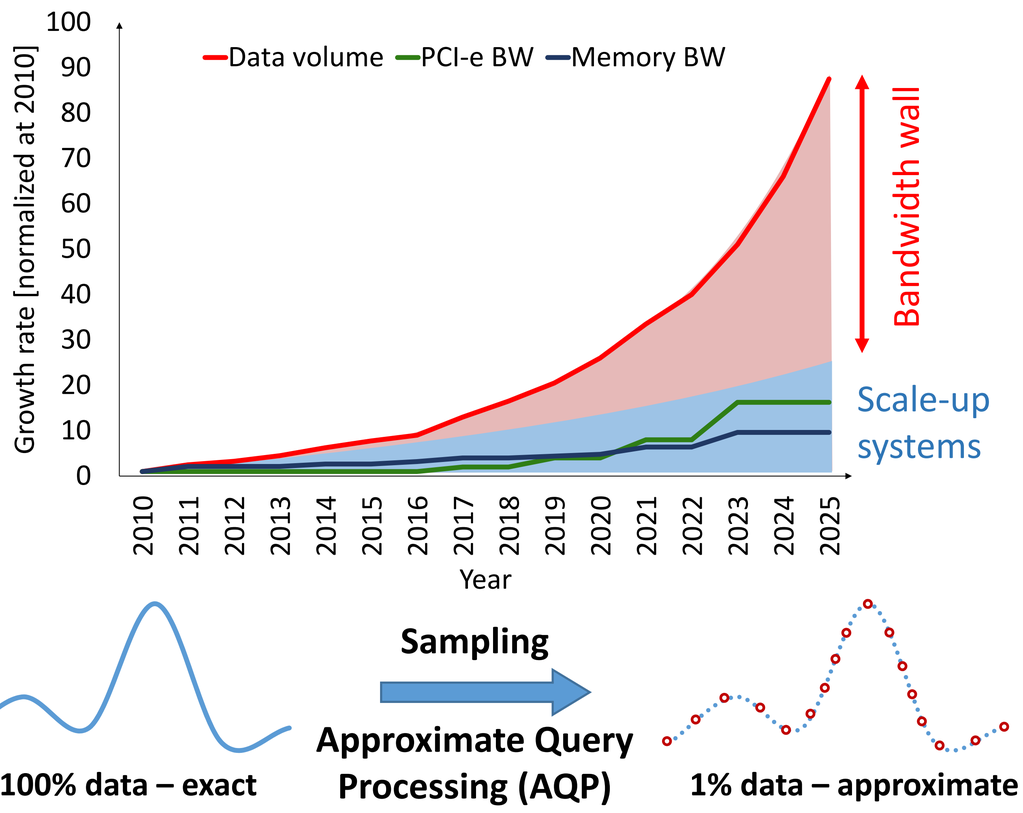
Sampling-Based AQP in Modern Analytical Engines

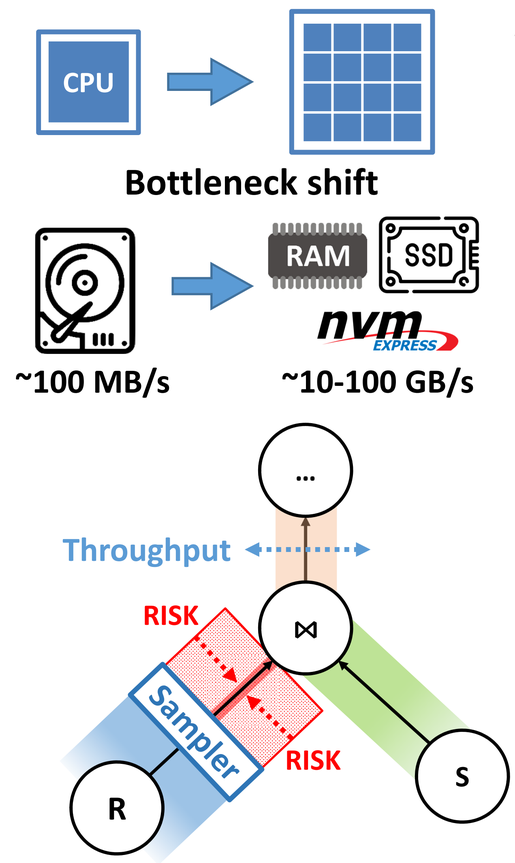
Viktor Sanca and Anastasia Ailamaki

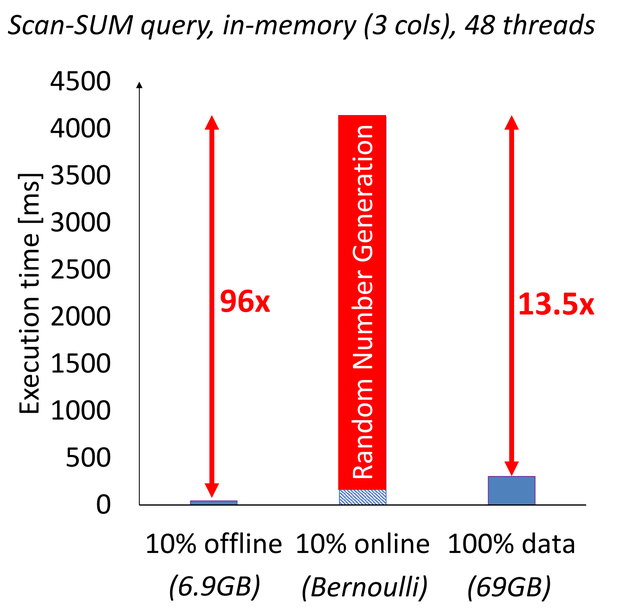
Interactive analytics is an elusive goal



AQP + modern systems = faster analytics

Sampling in the critical path of execution



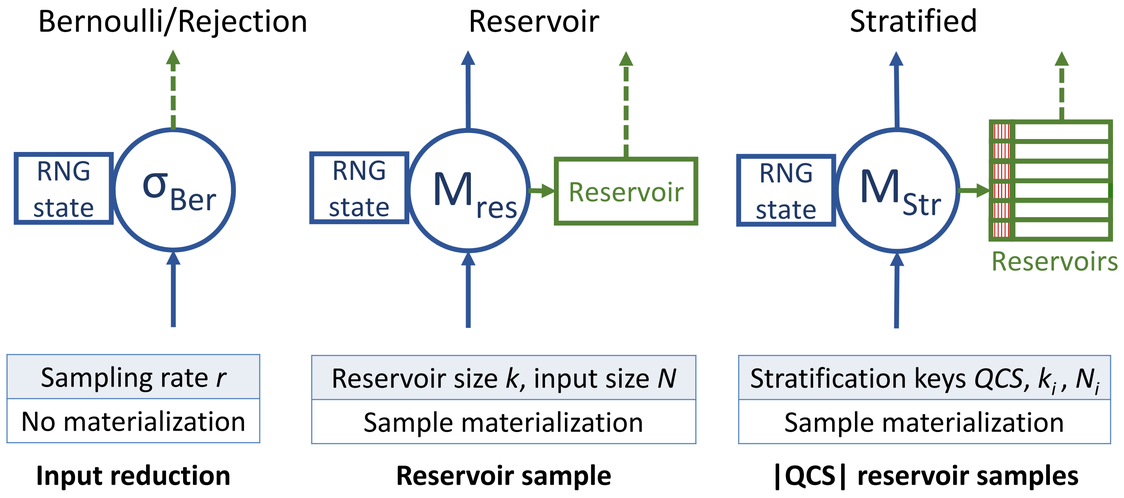


Operator throughput mismatch

Detrimental and impractical?
slower + approximate

Modern analytics require hardware-consciousness

Sampling operator design



Sampling rate *r*
No materialization

Input reduction

Reservoir size *k*, input size *N*
Sample materialization

Reservoir sample

Stratification keys *QCS*, k_i, N_i
Sample materialization

|QCS| reservoir samples

Seamless integration with the existing system

Just-in-time sampling triggered by query operators

Match the throughput of corresponding relational operators

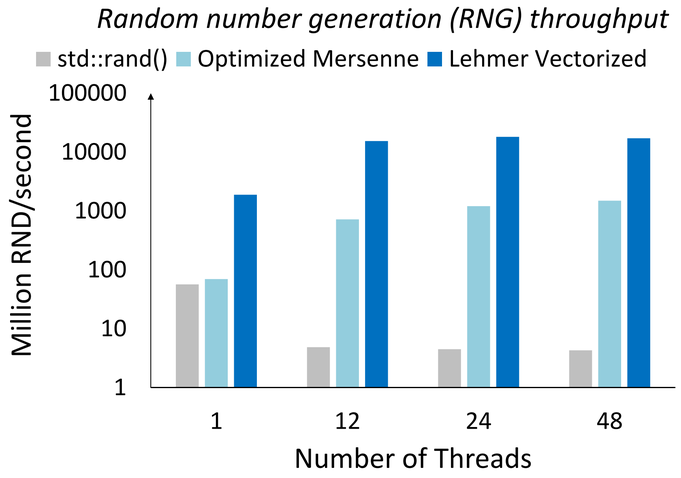
Goal: low overhead side-effect of query execution

Common operations and access patterns

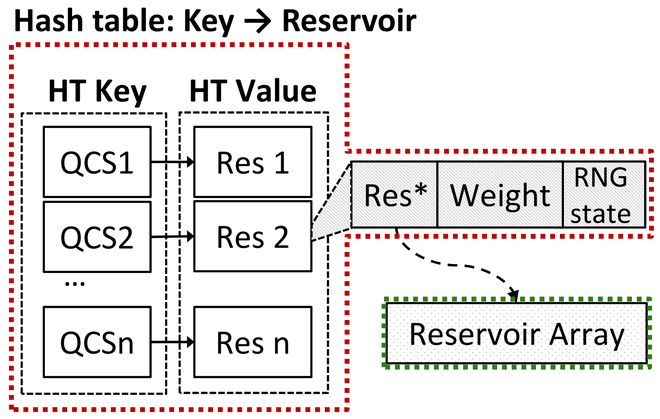
Per-thread RNG

In-register state

A few-instruction update



Hash table: Key → Reservoir



Decouple state from data

State: frequent access

Data: infrequent updates

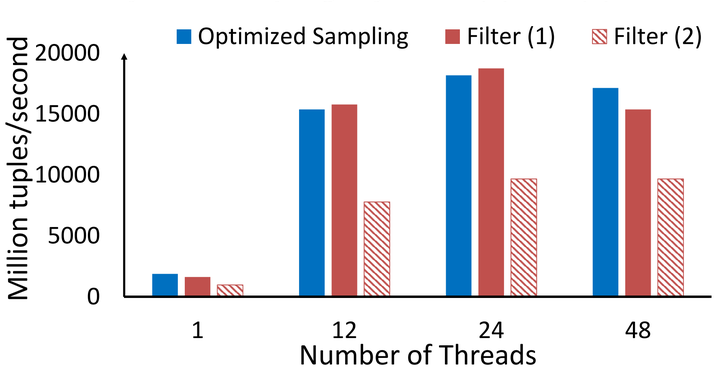
Better cache locality

Co-design the physical operators with algorithms

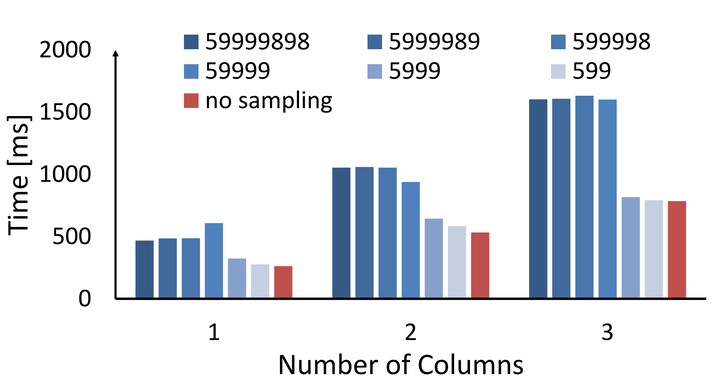
Exposing the bottlenecks: sampling inside an in-memory scale-up analytical engine

Setup: dual socket Intel Xeon Gold 5118 (2x12 cores), 384GB RAM **Data:** SSB with 600M (SF100) and 6B (SF1000) tuples in fact table, 1 binary column has ~2.3/23GB

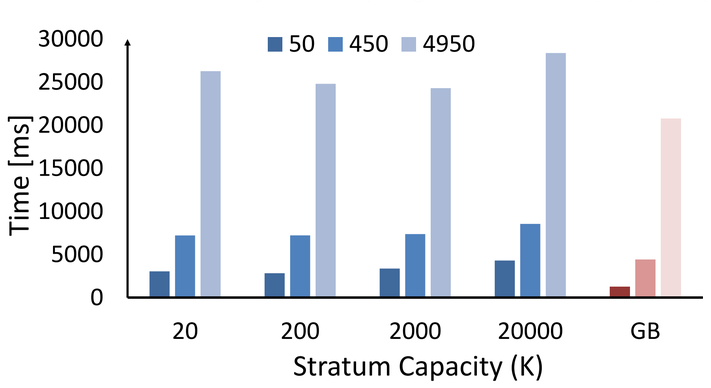
Rejection sampling *vs filter on (1) and (2) columns*



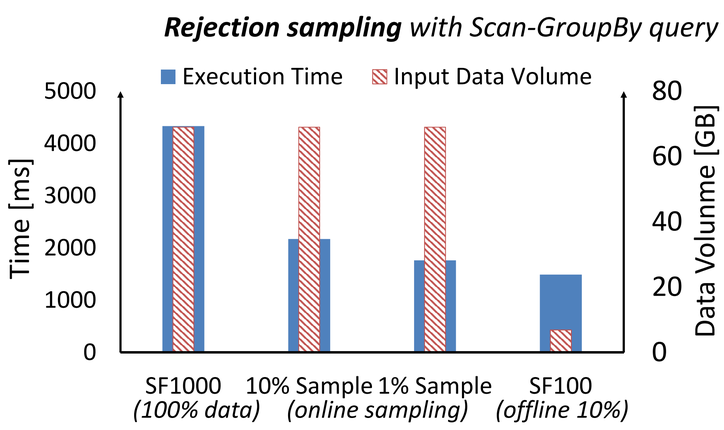
Reservoir Sampling *vs Scan-Reduce query*



Stratified Sampling *vs Scan-GroupBy query*



Rejection sampling *with Scan-GroupBy query*



Hardware- and algorithm-conscious sampling design

Minimize the overheads in the query processing pipeline

Reduce the overhead to 1-2.5x over matching operators

Sampling-based AQP is viable in modern engines

proteusdb.com