# BlinkDB

S. Agarwal, B. Mozafari, A. Panda, H. Milner,
S. Madden, I. Stoica
UC Berkeley MIT Conviva Inc.

Presented by Jacky

# Outline

- Background & Problem
- Approach
- System Architecture
- Creating samples
- Selecting samples
- Evaluation and results

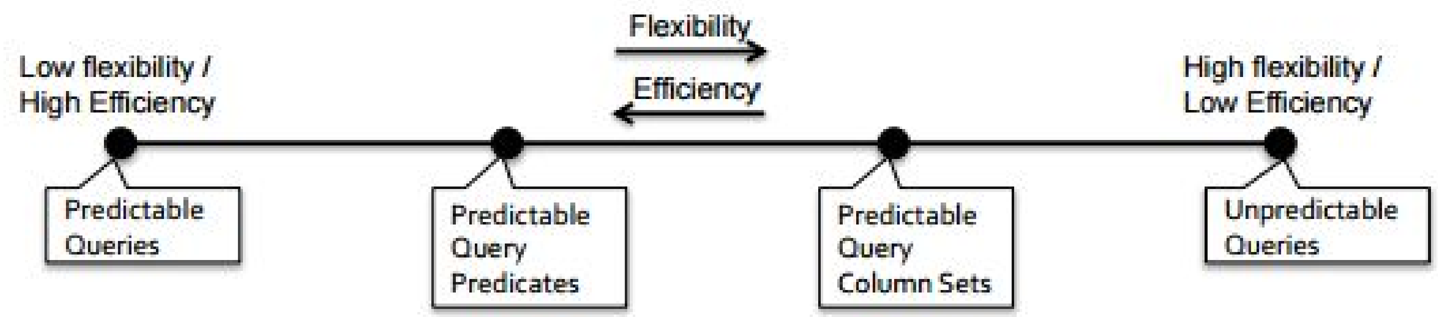
# Background & Problem

- Data analytics comes in different forms:
  - Web clicks
  - Online transactions
  - Download records
  - etc.
- Large volume of data with many features

- Problem: New applications requires near real-time responses.
  - Update ads on a website based on social network trends
  - Determine the subset of poor users experience based on some features

# Existing Solutions

- Traditional DB method: sequential scans on large fraction of the database
  => not feasible
- Other techniques:
  - Sampling
  - Sketches
  - Online aggregation



# BlinkDB

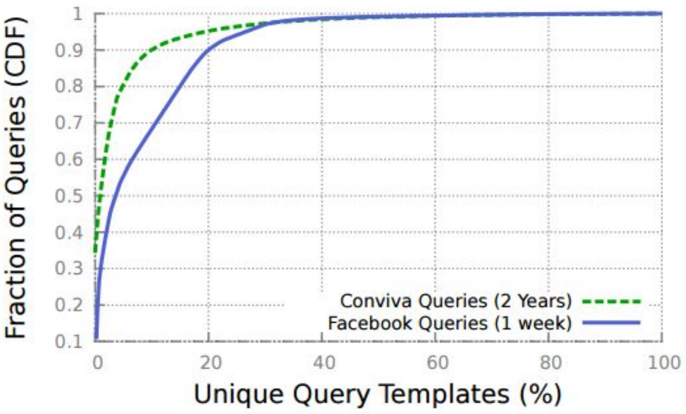
- BlinkDB is a massively parallel approximate query engine
- It trades off query accuracy for response time and memory requirement
- Queries over multiple terabytes of data can be answered in seconds with error bounds.
- Assumptions:
  - The sets of columns used by aggregated queries are stable over time.
  - These sets of columns are referred as “query column sets” QCSs

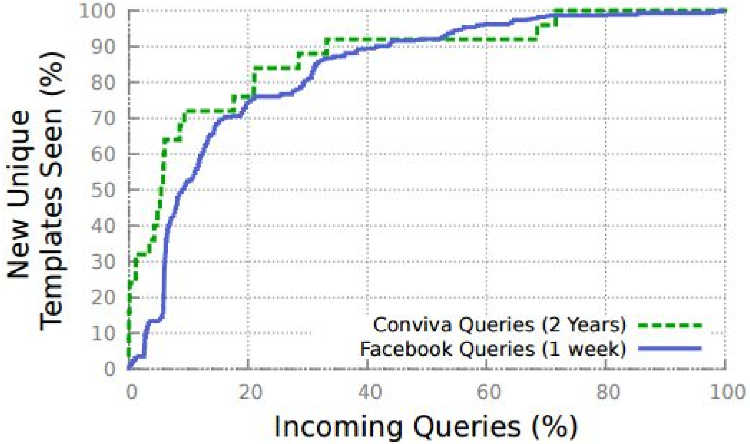
```
SELECT COUNT(*)
FROM Sessions
WHERE Genre = ‘western’
GROUP BY OS
ERROR WITHIN 10% AT CONFIDENCE 95%
```

```
SELECT COUNT(*)
FROM Sessions
WHERE Genre = ‘western’
GROUP BY OS
WITHIN 5 SECONDS
```

# Is QCSs are stable over time?

- From Conviva, analyze over 18K queries from 30 days
- From Facebook, analyze over 69K queries from 7 days

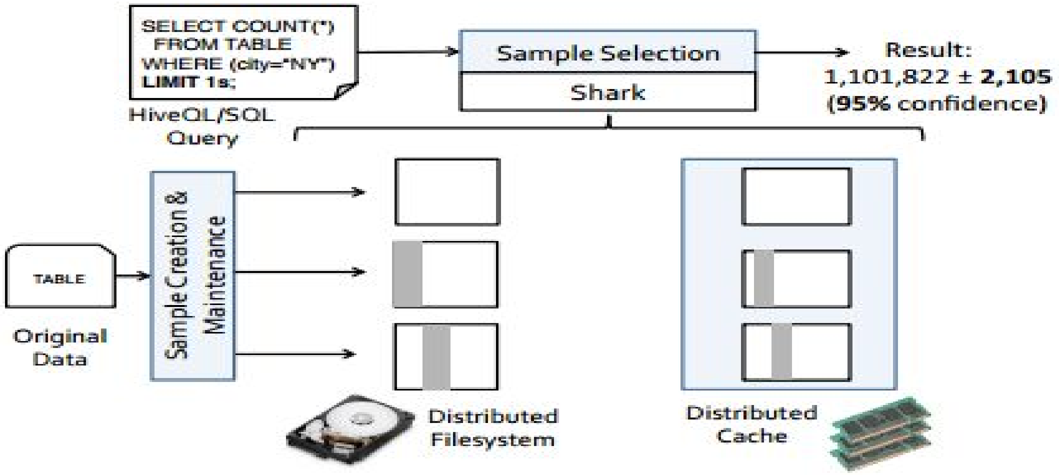



- QCSs are relatively stable over time, which suggests that the past history is a good predictor for the future workload.

# System Architecture

- Two major components:
  - Sample creation: builds and maintains a set of multi-dimensional stratified samples
  - Sample selection: use a dynamic sample selection strategy that selects an appropriate sized sample based on a query's response time and accuracy requirements



# Sample creation

- Uniform sampling often does not work well for a query that require filter or group operations
- Stratified sampling ensure that rare subgroups are sufficiently represented
- Sample creation module takes into account:
  - The frequency of rare subgroups in the data
  - The column sets in the past queries
  - The storage overhead of each sample

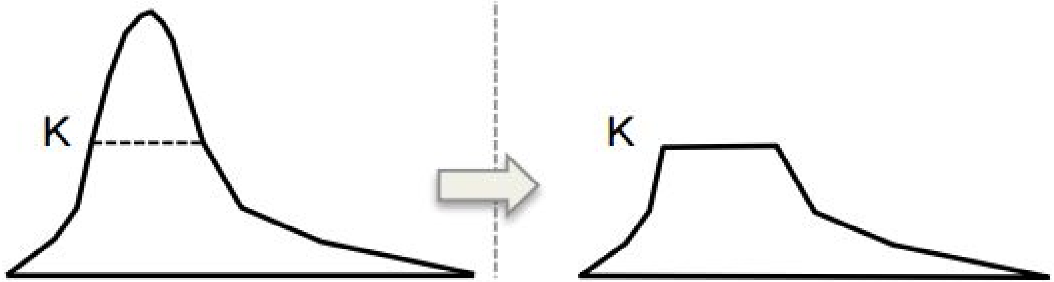
# Creating stratified samples

- For a specified QCS, $\varphi$, on the table T
- Limited by time bound ,t, and error bound ,e
- The maximum number of rows that can be accessed during t is n.
- If $t\uparrow$ $n\uparrow$
- if $e\uparrow$ $n\downarrow$
- Let $D(\varphi)$ be the set of unique values x on columns in $\varphi$
- Let $T_x$ be the rows in T that has values x on columns $\varphi$
- We will choose a sample, S, to represent T with $|S|=n$
- For each group $T_x$, there will be $S_x$ in S that is a subset of $T_x$.
- The aggregate calculation for each $S_x$ will subject to error that will depend on its size.

# Creating stratified samples

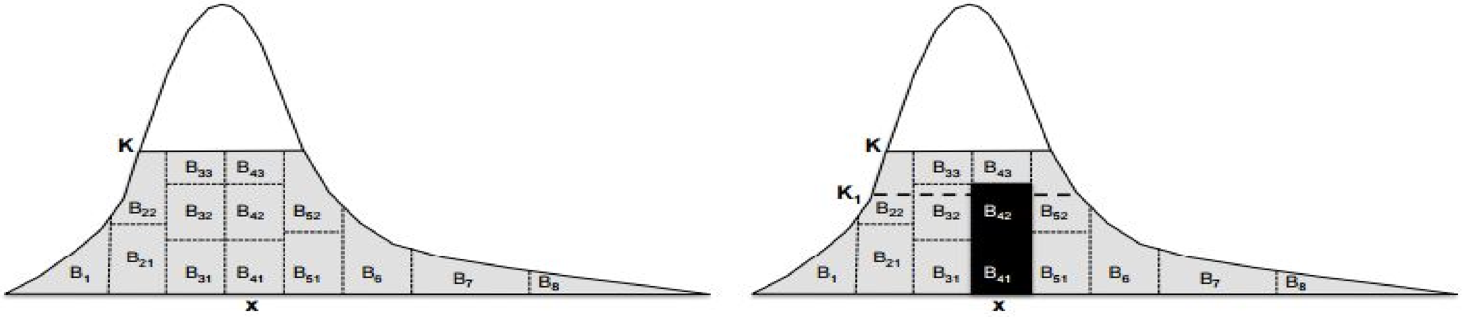
- Since error decreases as sample size increases, the best choice simply assigns equal sample size to each group
- And the assignment of sample sizes is deterministic

1. Compute group counts: sample cap of each group to be $K=\lfloor n/|D(\varphi)| \rfloor$
2. Take samples: take a sample $S(\varphi,K)$ which is a stratified sample associated with $\varphi$, where frequency of every group x in $\varphi$ is capped by K. If $|T_x|>K$, aggregate operators have standard error inversely proportional to sqrt(K)



# Sample storage layout

- The rows of stratified sample S(φ,K) are stored sequentially according to the order of columns in φ




- The sample are divided into blocks and distributed in HDFS
- Furthermore, storing sample in blocks allow we to compare different stratified sample using a small K-value (i.e. $K_1$).

# Storage overhead

- It turns out that the storage required by sample S(φ,K) is only 2.4% of the original table for $K=10^4$, 5.2% for $K=10^5$, and 11.4% for $K=10^6$
- We want to build several multidimensional stratified sample
- But we can build $n^2$-1 stratified sample
- We need to find a subset from these possible stratified sample that maximize the weighted sum of coverage on the QCSs

# Optimize a mixed integer linear program (MILP)

$$G = \sum_j p_j \cdot y_j \cdot \Delta(q_j, M) \tag{1}$$

subject to

$$\sum_{i=1}^{m} |S(\phi_i, K)| \cdot z_i \leq \mathbb{C} \tag{2}$$

and

$$\forall\ j: \quad y_j \leq \max_{i:\phi_i \subseteq q_j \cup i:\phi_i \supset q_j} (z_i \min 1, \frac{|D(\phi_i)|}{|D(q_j)|}) \tag{3}$$

where $0 \leq y_j \leq 1$ and $z_i \in \{0, 1\}$ are variables.

# Sample selection

- Given a query Q, the goal is to select one (or more) samples at run-time that meet the time bond or error constraints.
- Then use Error Latency Profile (ELP) to determine the optimal sample and the sample size
- ELP a heuristic that enables quick evaluation of different query plans in BlinkDB to pick the one that can best satisfy a query’s error/response time constraints.

# Selecting a sample

- If $q \subseteq \varphi_i$, BlinkDB will pick the $S(\varphi_i,K)$ with the smallest number of columns in $\varphi_i$
- Or else, from all the samples currently in-memory, select the samples that gives the highest selectivity from running the query.
    - Selectivity is the ratio of the number of rows selected by Q, to the number of rows read by Q (in the sample)

# Selecting the right sample size

- Construct an ELP for the query
- ELP characterizes the rate at which the error decreases (and the query response time increases) with increasing sample size.
- ELPs can be build by running the query on smaller subsample of the potential samples to estimate the selectivity, projects latency and error
- The sample that gives the optimal Error profile and Latency profile is chosen

(a) dt, country

(b) dt, dma

(c) dt, ended_flag

# Error Profile

- An error profile is created for all queries with error constraints.
- The error profile tries to predict the size of the smallest sample to satisfies Q’s error constraint
- Variance and confidence intervals are estimated using standard closed-form formulas from statistics
- Also estimates the query selectivity, sample variance, and the input data distribution by running the query on a number of small sample subsets.
- The number of rows required to meet Q’s error constraints is calculated using standard closed form statistical error estimates.

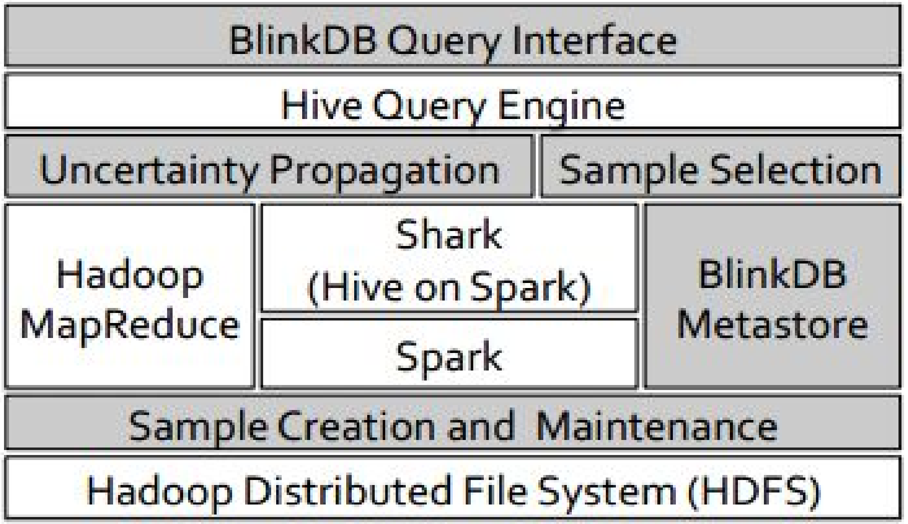
# Latency Profile

- A latency profile is created for all queries with response time constraints.
- The latency profile tries to predict the size of the largest sample to satisfies Q's time constraint
- The value of n depends on the physical placement of input data, the query structure and complexity, and the degree of parallelism.
- To simplify, BlinkDB predicts n by assuming the latency scales linearly with input size.

# Bias correction

- Running a query on a non-uniform sample can be statistical bias if the different groups are picked at different frequencies.
- For example, all rows from a rare subgroup would be in the sample while the popular subgroup will only have a small fraction of rows represented.
- BlinkDB keeps track of the effective sampling rate for each group associated with each sample in the sample table schema.
- BlinkDB uses the effective sampling rate  to weight different subgroups to produce an unbiased result.

# BlinkDB implementation stack

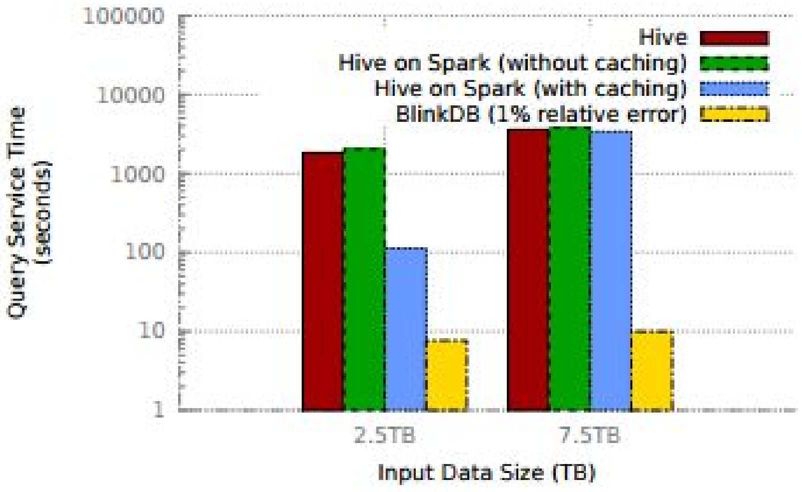


# Evaluation

- With a 100 node EC2 cluster
    - Each node with 8 CPU cores (2.66GHz), 68.4 GB of RAM, 800 GB of disk
    - Total: 75 TB of distributed disk storage and 6 TB if distributed RAM
- TPC-H benchmarks
    - 1TB of data
    - 22 benchmark queries
- Real-world analytic workload from Conviva Inc
    - 17TB of information about video streams viewed by internet users.
    - Provided a query log consists of 19296 queries

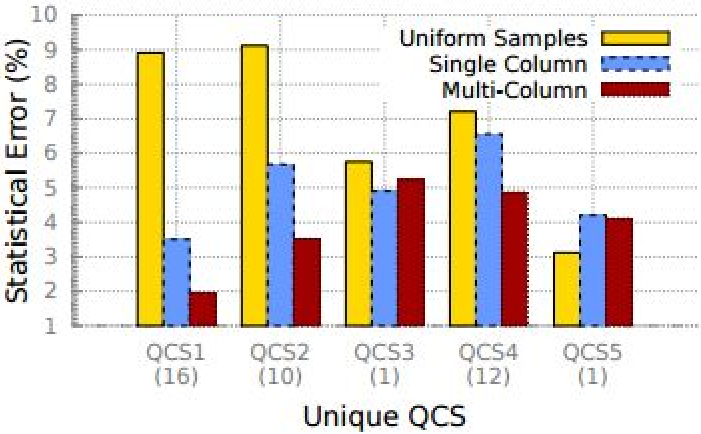
# Result

- BlinkDB versus no sampling

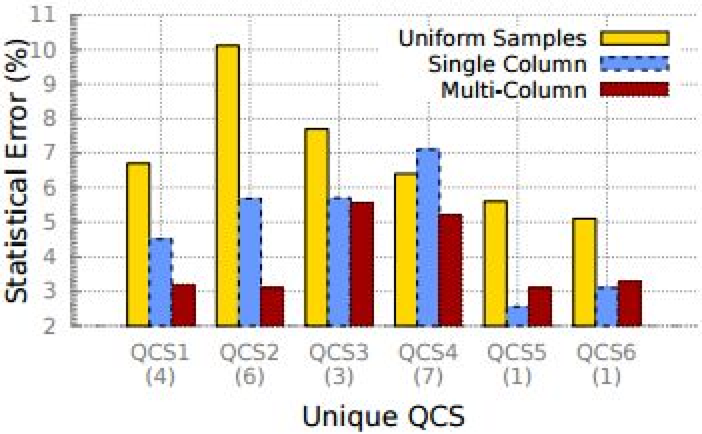


# Result

- Multi-dimensional stratified samples versus others



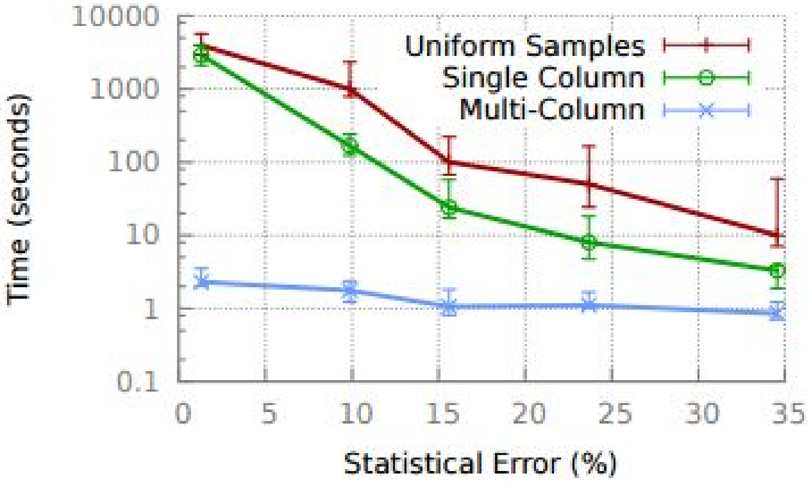

(a) Error Comparison (Conviva)




(b) Error Comparison (TPC-H)

# Result

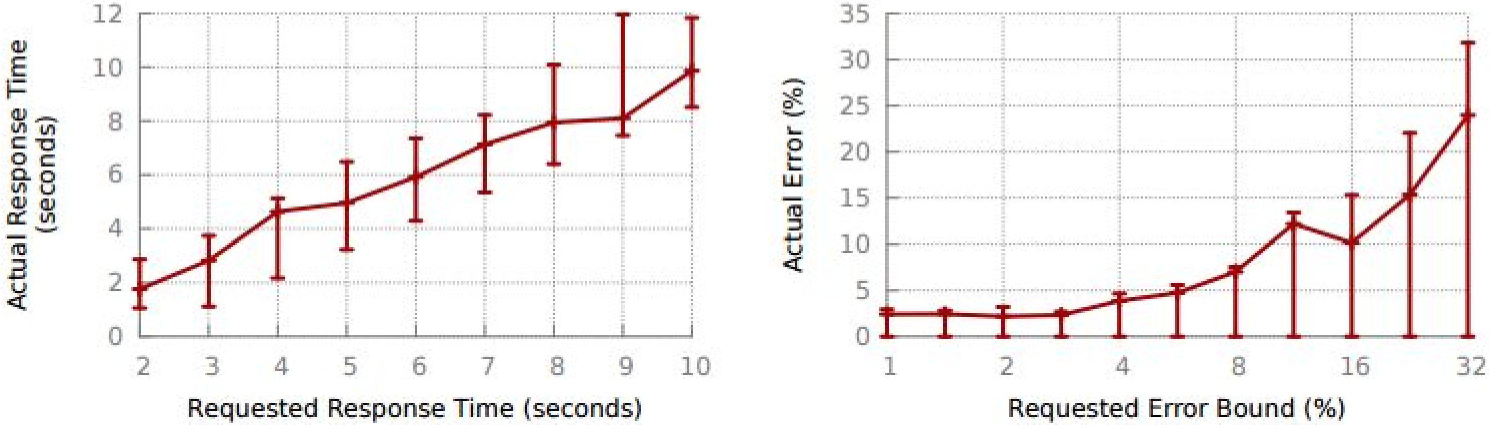
- Convergence properties




(c) Error Convergence (Conviva)

# Result

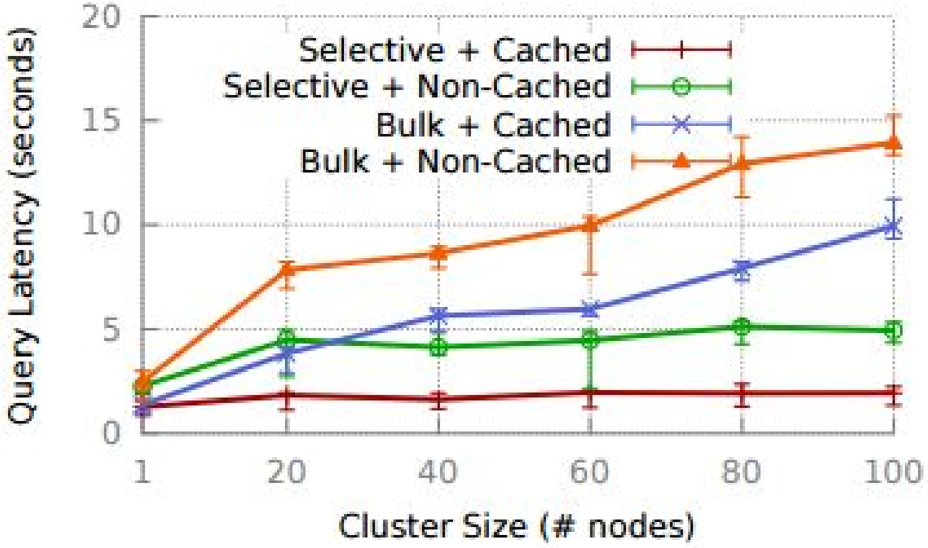
- Time & Accuracy guarantees




(a) Response Time Bounds (b) Relative Error Bounds

# Result

- Scaling up



# Result

- On a 100 node cluster, BlinkDB can answer queries on up to 17 TBs of data in less than 2 seconds (over 200x faster than Hive), within an error of 2-10%
- Two orders of magnitude faster than running the same queries on Hive/Hadoop

# Summary

- This parallel sampling based approximate approach supports ad-hoc queries with error and response time constraints.
- This approximate approach provides users with a result with error bonds but it gives a faster response time

# Thanks, Q&A