BlinkDB: queries with bounded errors and bounded response times on very large data

Authors: Sameer Agarwal, Aurojit Panda, Barzan Mozafari, Samuel Madden, Ion Stoica

8th ACM European Conference on Computer Systems

15 April 2013

Presenter: Bozhou Lu, Dominic DiResta, Karen Sun, Zhong Zhang

Georgia Tech

Background & Motivation

BlinkDB - Motivation

- Problem
  - Present query results near real time for large data
  - Unpredictable queries
  - Uneven data distribution

- Why matters
  - Growing need for faster responsiveness in data analytics applications
    - Update ads based on real time search history
    - Compare prices of financial securities in real time

BlinkDB - Background

- Example
  - Computing average over 10 TB data
  - 30 - 45 minutes on Hadoop
  - 5 - 10 minutes in memory

  - BlinkDB - seconds with statistical error bound

BlinkDB - Background

- Within an error range

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

- Within a time range

```
SELECT COUNT(*), RELATIVE ERROR AT 95% CONFIDENCE
FROM Sessions
WHERE Genre = ‘western’
GROUP BY OS
WITHIN 5 SECONDS
```

Related Work

BlinkDB - Existing Solutions

- 1. Sampling
  - Execute queries on a subset of data
  - Limitation: require assumptions about the query types and data distributions
- 2. Sketch-based Solution
  - Reorganize the data for quick and approximate aggregate operations (count, sum)
  - Limitation: require assumptions about the query types and data distributions

BlinkDB - Existing Solutions

- 3. Online Aggregation (OLA)
  - Offers preliminary results and allows users to stop the query early
  - Fewer assumptions on the data
  - Limitation: poor performance on rare tuples

- Current
  - BlinkDB is no longer maintained

Overview

Overview

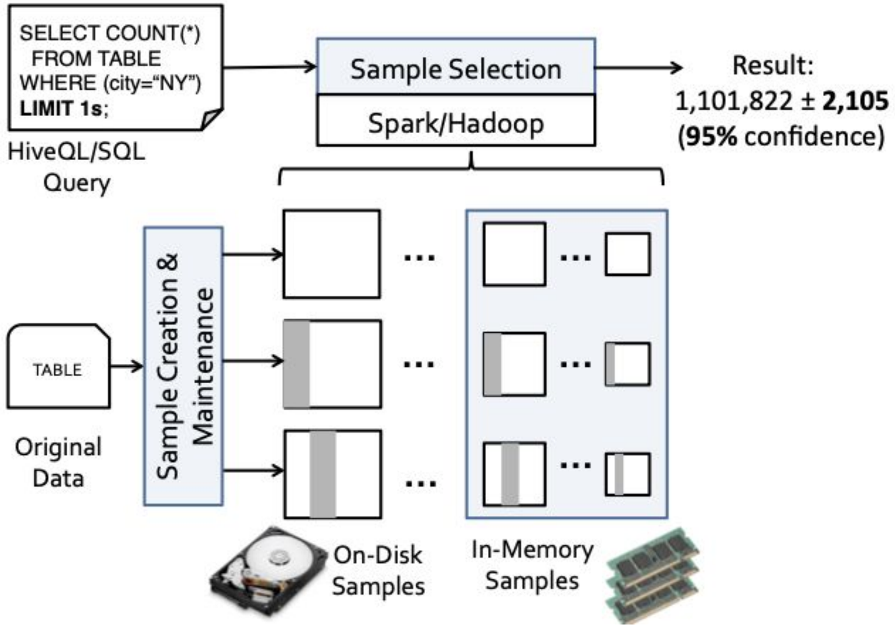
- BlinkDB would answer the queries over huge data quickly in the absence of perfect answers.
- Develop a multi-dimensional, multi-granular stratified sampling strategy
- Cast the decision of what stratified samples to build as an optimization problem:
  - The skew of the data distribution
  - Query templates
  - The storage overhead of each sample
- Develop a run-time dynamic sample selection strategy

Settings and Assumptions

- Workload characteristics
  - the query templates remain fairly stable over time
    - To optimize the creation of samples
- Queries with joins
  - BlinkDB supports two types of joins
    - arbitrary joins
    - joins with a join-key
- Closed-form aggregates
  - focus on a small set of aggregation operators: COUNT, SUM, MEAN, MEDIAN/QUANTILE
  - estimate the error of certain aggregate functions using standard closed-form error estimations.
- Offline Sampling
  - a sample may not represent the full dataset accurately
  - periodically replace samples with new ones in the background to maintain accuracy without significant performance costs

Architecture

- BlinkDB builds on the **Apache Hive framework** and has two major components.
- **Offline sampling module** that creates and maintains samples over time
  - Offline Sample Creation
  - Sample Maintenance
  - Storage Optimization
- **Run-time sample selection module** that creates an Error-Latency Profile for ad-hoc queries
  - ELP (Error-Latency Profile)



Example

- creates several multidimensional and multi-resolution samples based on past query templates and the data distribution
- In the example, decides to create two sample families of stratified samples: one on City, and another one on (OS,URL).
- creates several instances of each sample family, each with a different size, or resolution.
- For every query, at run time, BlinkDB selects the appropriate sample family and the appropriate sample resolution to answer the query based on the user specified error or response time bounds.

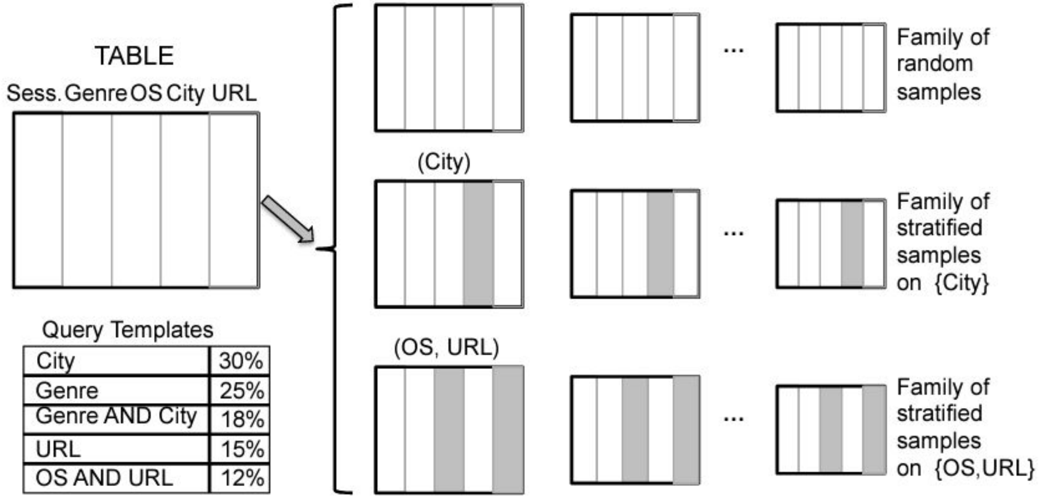


TABLE
Sess. Genre OS City URL
(City)
(OS, URL)
...
Family of random samples
Family of stratified samples on {City}
Family of stratified samples on {OS,URL}

Query Templates

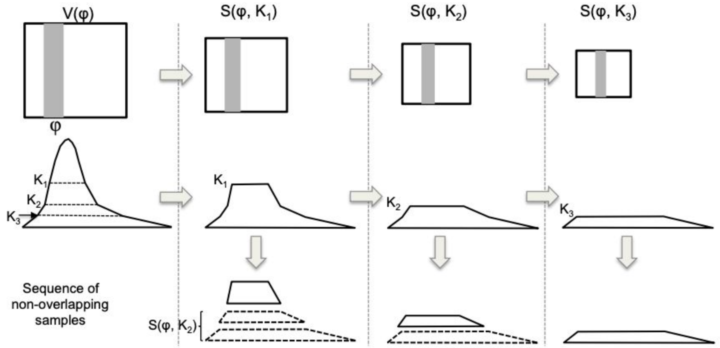
City	30%
Genre	25%
Genre AND City	18%
URL	15%
OS AND URL	12%

Sample Creation

Multi-resolution Stratified Samples

- **Why stratified samples are needed**
  - It both provides faster convergence of answer estimates and avoids missing subgroups in results.
- **How it is done**
  - BlinkDB creates stratified samples based on subsets of columns that are frequently queried together.
  - It constructs smaller samples from the larger ones, and thus need an amount of storage equivalent to maintaining only the largest sample.

$$SFam(\phi) = \{S(\phi, K_i) \mid 0 \le i < m\},$$



Optimization Framework

$$G = \sum_{i=1}^{m} w_i \cdot y_i \cdot \Delta(\phi_i^T)$$

- **Problem formation**
  - Non-uniformity (skew) of the data
    - The greater the skew for a set of columns, the more important it is to have a stratified sample on those columns.
    - Non-uniformity metric
  - Workload
    - Ues a query workload defined as a set of query templates and their weights (normalized frequency or importance)
  - Storage cost
    - We use Store(φ) to denote the storage cost (say, in MB) of building a sample family on a set of columns φ.

$$\sum_{j=1}^{\alpha} Store(\phi_j) \cdot z_j \leq \mathbb{S}$$

- **Scaling the Solution**
  - To reduce the number of candidate column-sets found by the optimization framework, we restrict the candidate subsets to only those that have appeared together at least in one of the query templates.

Run Time

Sample Selection

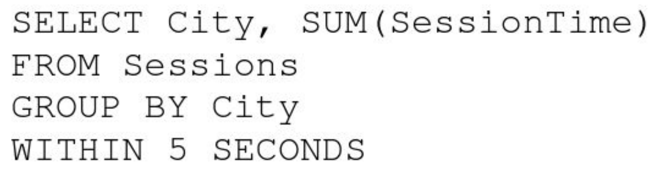
- **Selecting the Sample Family**
  - Tried to select the stratified sample that contains the union of the columns appeared in each predicates
  - If not found, run Q in parallel on the smallest sample of all sample families then select based on:
    - the number of rows selected by Q
    - the number of rows read by Q (i.e., number of rows in that sample)

- **Selecting the Sample size**
  - Construct Error-Latency Profile
    - The ELP characterizes the rate at which the error decreases (and the query response time increases) with increasing sample sizes, and is built simply by running the query on smaller samples to estimate the selectivity and project latency and error for larger samples.

Query Answer for Uniform Sample

Uniform Sample: (EASY:)

- suppose we take a uniform sample with 40% of the rows of the original table.
- scale the final sums of the session times by 1/0.4 = 2.5 in order to produce an unbiased estimate of the true answer

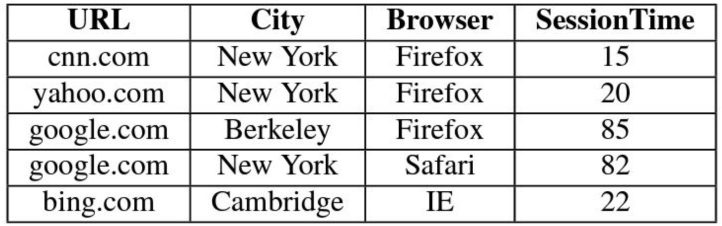


```
SELECT City, SUM(SessionTime)
FROM Sessions
GROUP BY City
WITHIN 5 SECONDS
```

Query Answer for Stratified Sample

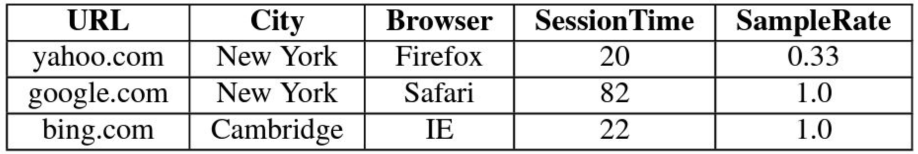
Stratified Sample:

- Strategy for Uniform Sample doesn't work
- different values were sampled with different rates
- to produce unbiased answers, BlinkDB keeps track of the effective sampling rate applied to each row



URL	City	Browser	SessionTime
cnn.com	New York	Firefox	15
yahoo.com	New York	Firefox	20
google.com	Berkeley	Firefox	85
google.com	New York	Safari	82
bing.com	Cambridge	IE	22

Table 3: Sessions Table.



URL	City	Browser	SessionTime	SampleRate
yahoo.com	New York	Firefox	20	0.33
google.com	New York	Safari	82	1.0
bing.com	Cambridge	IE	22	1.0

Table 4: A sample of Sessions Table stratified on Browser column.

Evaluation

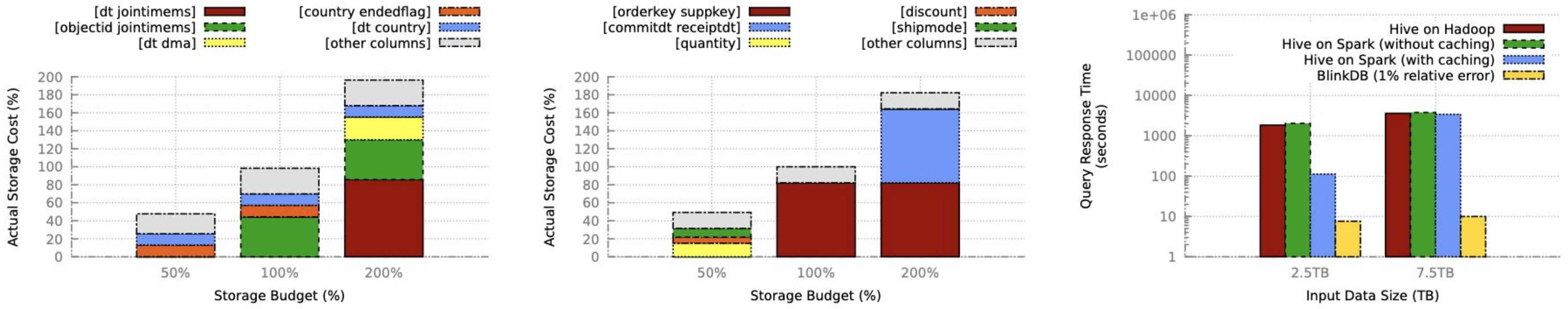
Hypothesis

- Hypothesis: Sampling provides higher query response times compared to no sampling when querying on a large dataset, even on simple queries
- Hypothesis: When querying rare subgroups, multi-dimensional, multi-resolution stratified samples attain higher accuracy and convergence over random uniform sampling and single-column stratified sampling approaches
- Hypothesis: BlinkDB accurately selects a sample for a desired query response time with its runtime selection strategy for sample family and sample size

BlinkDB vs. No Sampling

- Performance of BlinkDB versus frameworks executing on complete data were evaluated on 100 node EC2 cluster using two workloads
- Conviva Inc. Workload
  - 17TB dataset about video streams viewed by Internet users
- TPC-H Benchmark Workload
  - 1TB dataset for query benchmarking to ensure generality of results
- Evaluated response time of a simple query executed on Hive on Hadoop MapReduce, Hive on Spark, and BlinkDB
  - Query on 2.5 & 7.5TB subset: *average* of user session time, filter on *dt* (date column), and GROUP BY on *city* column
  - Utilized 1% error bound at 95% confidence for BlinkDB
- Results: For both datasets, BlinkDB answered query in a few seconds compared to thousands of seconds for other frameworks

BlinkDB vs. No Sampling



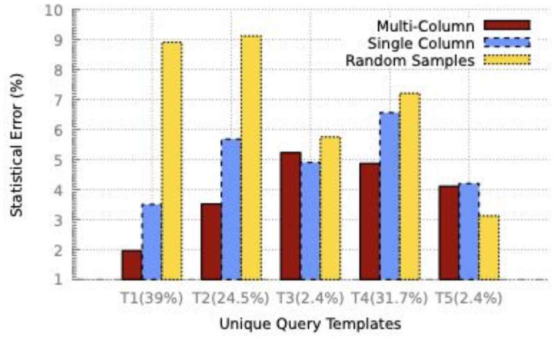
(a) Sample Families (Conviva) (b) Sample Families (TPC-H) (c) BlinkDB Vs. No Sampling

Figure 6: 6(a) and 6(b) show the relative sizes of the set of stratified sample(s) created for 50%, 100% and 200% storage budget on Conviva and TPC-H workloads respectively. 6(c) compares the response times (in log scale) incurred by Hive (on Hadoop), Shark (Hive on Spark) – both with and without input data caching, and BlinkDB, on simple aggregation.

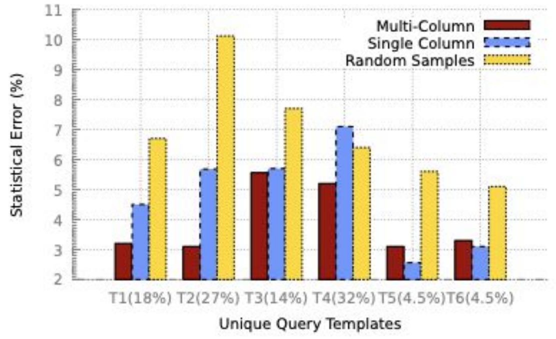
Multi-Dimensional Stratified Sampling

- Error and convergence were measured for BlinkDB's stratified-sampling strategy, simple random sampling, and one-dimensional stratified sampling
- The three samples were generated on the Conviva and TPC-H datasets with 50% storage constraint
  - Columns to stratify chosen by BlinkDB optimization framework
  - Random sample contained 50% of entire data chosen uniformly at random
- Error measurement: For set of 40 queries, average statistical error at 95% confidence from query running 10 seconds recorded for each sample
- Convergence measurement: Multiple queries run to determine latency for achieving an error bound with 95% confidence
- Results: Across both data sets, BlinkDB's multidimensional stratified sampling approach had lower error and faster convergence compared to the other sampling methods

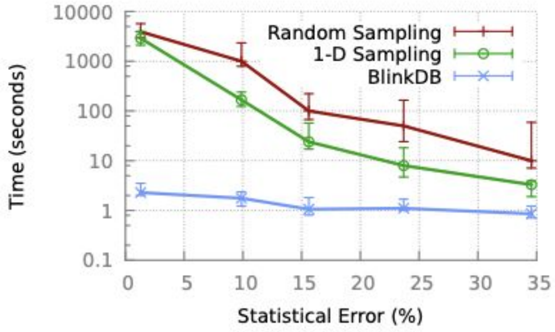
Multi-Dimensional Stratified Sampling



(a) Error Comparison (Conviva)



(b) Error Comparison (TPC-H)



(c) Error Convergence (Conviva)

Figure 7: 7(a) and 7(b) compare the average statistical error per template when running a query with fixed time budget for various sets of samples. 7(c) compares the rates of error convergence with respect to time for various sets of samples.

Time/Accuracy Guarantees

- Effectiveness of BlinkDB at meeting a time and error bounds specified by the user was evaluated
- Time-bounded evaluation: sample of 20 Conviva queries, each ran 10 times, for time bounds from 1 to 10 seconds
- Error-bounded evaluation: error was recorded for same set of queries, with specified error bounds ranging from 2% to 32%
- Results: BlinkDB consistently selects a sample that meets a specified time bound and the measured error. Measured error is almost always less than or equal to requested error
- Scaling up: BlinkDB's ability to scale to different cluster sizes was also measured for two sets of query workloads: high selective queries and number crunching queries

Time/Accuracy Guarantees

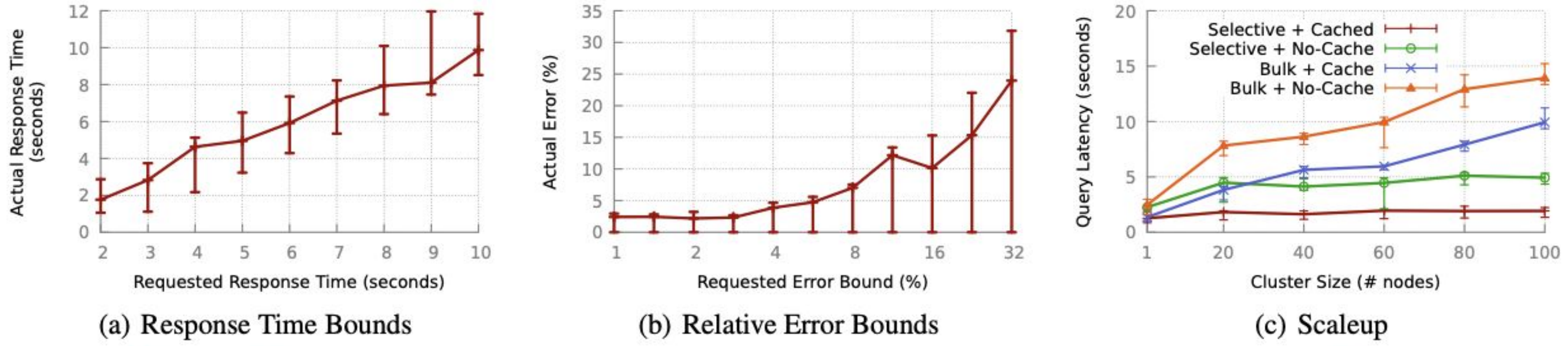


Figure 8: 8(a) and 8(b) plot the actual vs. requested response time and error bounds in BlinkDB. 8(c) plots the query latency across 2 different query workloads (with cached and non-cached samples) as a function of cluster size

Conclusion

BlinkDB

- A parallel, sampling- based approximate query engine that provides support for ad-hoc queries with error and response time constraints. Based on two key ideas:
  - a multi-dimensional, multi-granularity sampling strategy that builds and maintains a large variety of samples
  - a run-time dynamic sample selection strategy that uses smaller samples to estimate query selectivity and choose the best samples for satisfying query constraints.
- Evaluation results on real data sets and on deployments of up to 100 nodes demonstrate the effectiveness of BlinkDB at handling a variety of queries with diverse error and time constraints, answering a range of queries within 2 seconds on 17 TB of data with 90-98% accuracy.

Challenges and Further Work

Challenges

- Maintaining stratified samples over time
- Potential issues in adapting to new, unforeseen query types.

Further Work

- Explore more adaptive sampling techniques and enhanced algorithms for optimizing sample selection and maintenance.

Study Questions

1. Why does BlinkDB use stratified sampling over other sampling methods like random uniform sampling?

1. How BlinkDB balances the trade-off between query accuracy and response time?