Recent Trends in Large Scale Data Intensive Systems

Barzan Mozafari

(presented by Ivo D. Dinov)

University of Michigan, Ann Arbor

Research Goals

Using statistics to build better data-intensive systems

1. **Faster**
   - How to query petabytes of data in seconds?
2. **More predictable**
   - How to predict the query performance?
   - How to design a predictable database in the first place?

Big Data

Online Media Websites, Sensor Data

Real-time Monitoring, Data Exploration

Big Data

Log Processing

Root-cause Analysis, A/B Testing

Problem

Problem: Analytical queries over massive datasets are becoming extremely slow and expensive

Goal: Support interactive, ad-hoc, exploratory analytics on massive datasets

Recent Trends in Large Data Processing

Computational Model: Embarrassingly parallel Map-Reduce

Software: fault tolerant Hadoop (OS for data centers)

Hardware: Commodity servers (lots of them!)

Realization: Moving towards declarative languages such as SQL

Trends in Interactive SQL Analytics

Impala
Presto
Stinger
Hive
Spark SQL
Redshift
HP Vertica
...

- Less I/O
  - Columnar formats / Compression
  - Caching Working Sets
  - Indexing
- Less Netw
  - Local C
- Faster P
  - Precomput
  - More CPUs/G

Good But Not Enough!
Because Data is Growing Faster than Moore's Law!

Data Growing Exponentially, faster than our ability to process it!

Estimated Global Data Volume*:
» 2011: 1.8 ZB => 2015: 7.9 ZB
(ZB = 10^{21} = 1 million PB = 1 billion TB)
World's information doubles every two years
Over next 10 years:
» # of servers will grow by 10x
» data managed by enterprise data centers by 50x
» # of "files" enterprise data center by 75x
» Kryder's law (storage) outpaces Moore's law (comput. power)**

* 2011 IDC Digital Universe Study
** Dinov et al., 2014

Outline

- **BlinkDB**: Approximate Query Processing
- Verdict: Database Learning

BlinkDB:

Query Petabytes of Data in a Blink Time!

Sameer Agarwal, **Barzan Mozafari**, Aurojit Panda, Henry Milner, Samuel Madde, Ion Stoica

100 TB & 1,000 cores

1-2 Hours

Hard Disks

→

25-30 Minutes

Memory

→

1 second

?

Target Workload

1. **Real-time latency** is valued over perfect **accuracy**

> "On a good day, I can run up to 6 queries in Hive."
> - Anonymous Data Scientist at facebook.

Target Workload

1. **Real-time latency** is valued over perfect **accuracy**: **≤ 10 sec for interactive experience**

> "On a good day, I can run up to 6 queries in Hive."
> - Anonymous Data Scientist at facebook

Target Workload

1. **Real-time latency** is valued over perfect **accuracy**: **≤ 10 sec for interactive experience**

2. Exploration is **ad-hoc**

Target Workload

1. **Real-time latency** is valued over perfect **accuracy**: **≤ 10 sec for interactive experience**
2. Exploration is **ad-hoc**
3. User defined functions (**UDF**) must be supported: **43.6% of Conviva's queries**
4. Data is **high-dimensional & skewed**: **+100 columns**

100 TB & 1,000 cores

1-2 Hours

25-30 Minutes

1 second

→

→ ?

Hard Disks

Memory

One can often make perfect decision without perfect answers

Approximation using Offline Samples

BlinkDB Interface

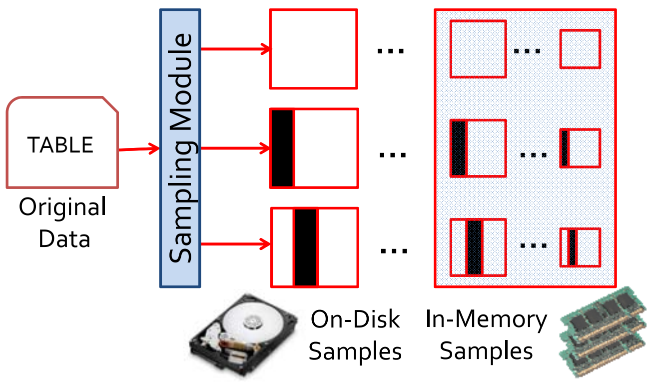
SELECT avg(sessionTime)
FROM Table
WHERE city='San Francisco'
WITHIN 1 SECONDS → 234.23 ± 15.32

BlinkDB Interface

SELECT avg(sessionTime)
FROM Table
WHERE city='San Francisco'
WITHIN 2 SECONDS → ~~234.23 ± 15.32~~ 239.46 ± 4.96

SELECT avg(sessionTime)
FROM Table
WHERE city='San Francisco'
ERROR 0.1 **CONFIDENCE** 95.0%

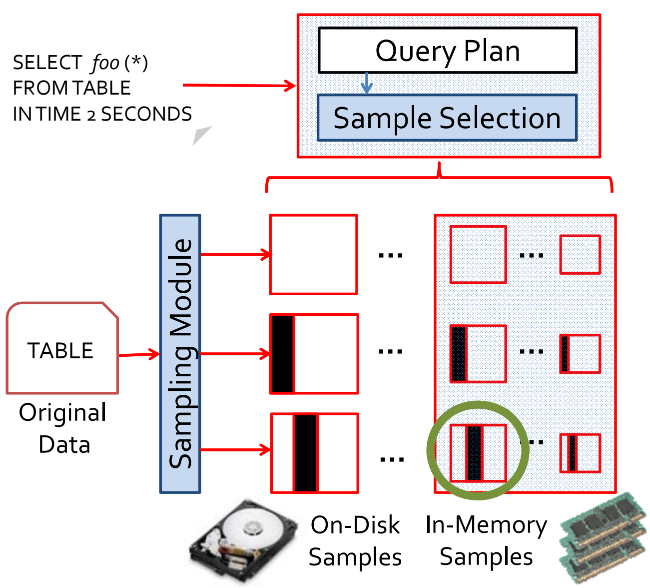
BlinkDB Architecture



Offline sampling:

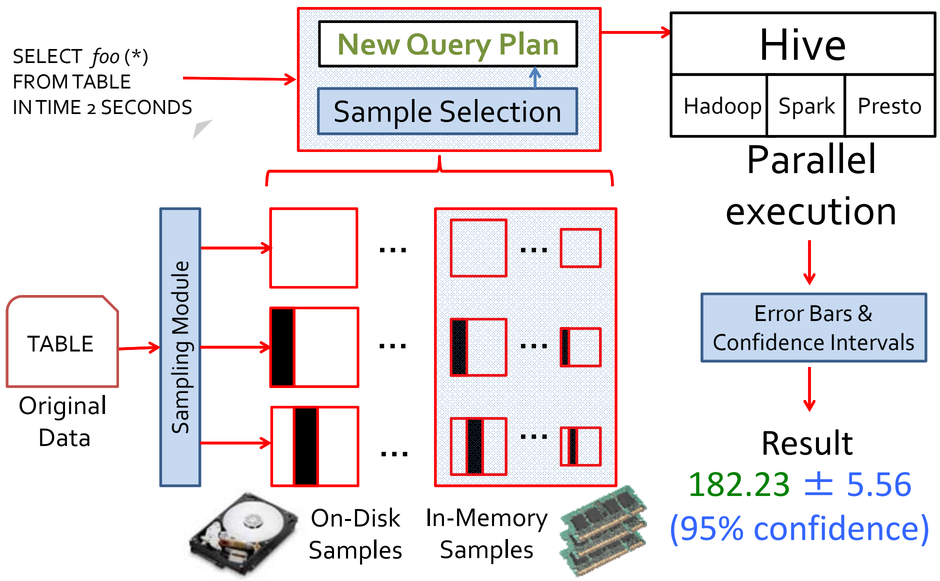
- » **Uniform**
- » **Stratified** on different sets of columns
- » Different sizes

BlinkDB Architecture



Predict **time** and **error** of the query for each sample type

BlinkDB Architecture



Main Challenges

1. How to accurately estimate the error?
2. What if the error estimate itself is wrong?
3. Given a storage budget, which samples to build & maintain to support a wide range of ad-hoc exploratory queries?
4. Given a query, what should be the optimal sample type and size that can be processed to meet its constraints?

Closed-Form Error Estimates

Central Limit Theorem (CLT)

1. Counts: $X_i = \begin{cases} 0 \\ 1 \end{cases} \sim B(n,p) \Rightarrow \Sigma = \sum_{i=1}^{n} X_i \sim N\big(np, np(1-p)\big)$
2. Total Sum: $\{Y_i\} \sim D(\mu, \sigma) \Rightarrow \Sigma = \sum_{i=1}^{n} Y_i \sim N(n\mu, n\sigma^2)$
3. Mean: $\{Z_i\} \sim D(\mu, \sigma) \Rightarrow \bar{x} = \frac{1}{n}\sum_{i=1}^{n} Z_i \sim N\left(\mu, \frac{\sigma^2}{n}\right)$
4. Variance: $\{U_i\} \sim D(\mu, \sigma) \Rightarrow \frac{(n-1)\bar{s}^2}{\sigma^2} \sim \chi^2(n-1)$,

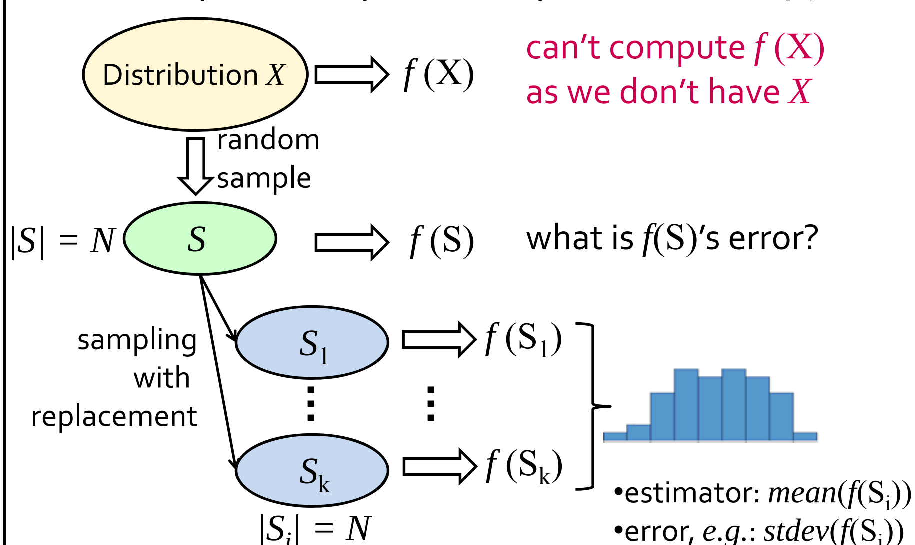
   with $\mu_{\bar{s}^2} = n-1$ and $\sigma^2{}_{\bar{s}^2} = n-1$.

What about more complex queries?

- UDFs, nested queries, joins, ...

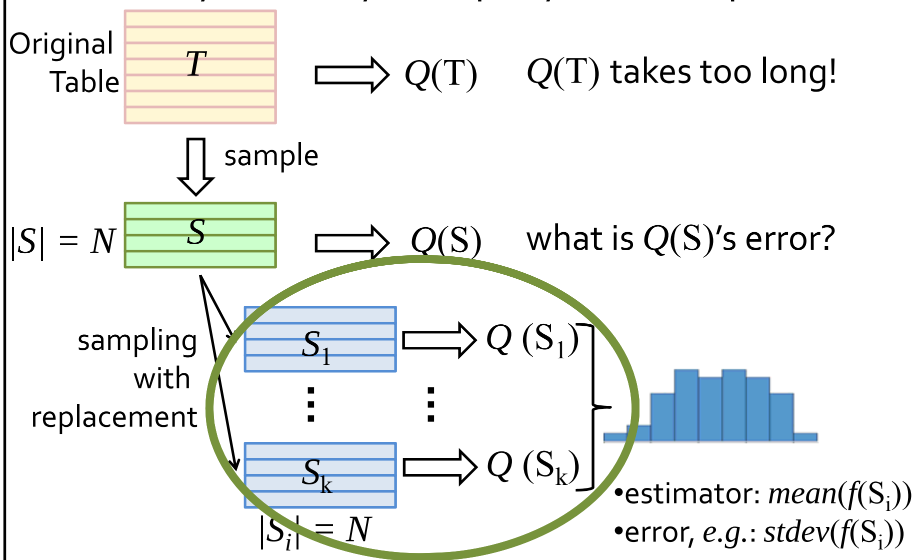
Bootstrap [Efron 1979]

Quantify accuracy of a sample estimator $f()$



Bootstrap

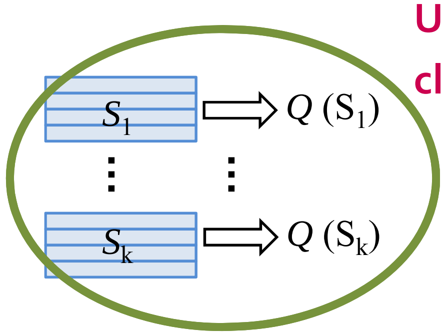
Quantify accuracy of a query on a sample table



Bootstrap

1. Bootstrap treats Q as a **black-box**
   - Can handle (almost) arbitrarily complex queries including UDFs!
2. **Embarrassingly Parallel**

Uses too many cluster resources



Error Estimation

1. CLT-based closed forms:

➢ **Fast** but **limited** to simple aggregates

2. Bootstrap (Monte Carlo simulation):

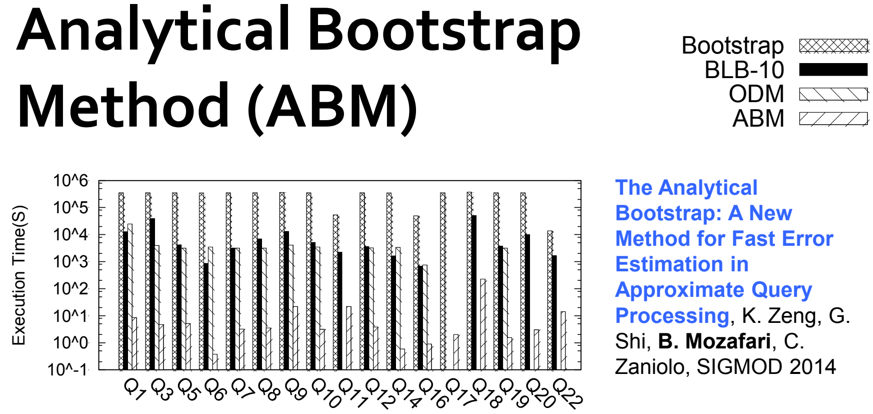
➢ **Expensive** but **general**

3. Analytical Bootstrap Method (ABM):

➢ **Fast** and **general**

✓ (some restrictions, e.g. no UDF, some self-joins, …)

Analytical Bootstrap Method (ABM)



The Analytical Bootstrap: A New Method for Fast Error Estimation in Approximate Query Processing, K. Zeng, G. Shi, **B. Mozafari**, C. Zaniolo, SIGMOD 2014

ABM is 2-4 orders of magnitude faster than simulation-based implementations of bootstrap

Bootstrap = (naïve) Bootstrap method
BLB = Bag of Little Bootstrap (BLB-10 = BLB on 10 cores)
ODM = On-Demand Materialization
ABM = Analytical Bootstrap Method

Main Challenges

1. How to accurately estimate the error?
2. What if the error estimate itself is wrong?
3. Given a storage budget, which samples to build & maintain to support a wide range of ad-hoc exploratory queries?
4. Given a query, what should be the optimal sample type and size that can be processed to meet its constraints?

Problem with Uniform Samples

ID	City	Age	Salary
1	NYC	22	50,000
2	Ann Arbor	25	120,242
3	NYC	25	78,212
4	NYC	67	62,492
5	NYC	34	98,341
6	Ann Arbor	62	78,453

Uniform Sample

ID	City	Age	Salary	Sampling Rate
3	NYC	25	78,212	1/3
5	NYC	34	98,341	1/3

SELECT avg(salary)
FROM table
WHERE city = 'Ann Arbor'

Problem with Uniform Samples

ID	City	Age	Salary
1	NYC	22	50,000
2	Ann Arbor	25	120,242
3	NYC	25	78,212
4	NYC	67	62,492
5	NYC	34	98,341
6	Ann Arbor	62	78,453

Larger Uniform Sample

ID	City	Age	Salary	Sampling Rate
3	NYC	25	78,212	2/3
5	NYC	34	98,341	2/3
1	NYC	22	50,000	2/3
2	Ann Arbor	25	120,242	2/3

SELECT avg(salary)
FROM table
WHERE city = 'Ann Arbor'

Stratified Samples

ID	City	Age	Salary
1	NYC	22	50,000
2	Ann Arbor	25	120,242
3	NYC	25	78,212
4	NYC	67	62,492
5	NYC	34	98,341
6	Ann Arbor	62	78,453

Stratified Sample on City

ID	City	Age	Salary	Sampling Rate
3	NYC	67	62,492	1/4
5	Ann Arbor	25	120,242	1/2

SELECT avg(salary)
FROM table
WHERE city = 'Ann Arbor'
AND age > 60

Target Workload

1. **Real-time latency** is valued over perfect **accuracy: ≤ 10 sec for interactive experience**
2. Exploration is **ad-hoc**
3. Columns queried together (i.e., **Templates**) are **stable** over time
4. User defined functions (**UDF**) must be supported: **43.6% of Conviva's queries**
5. Data is **high-dimensional & skewed**: **100+ columns**

Which Stratified Samples to Build?

For **n** columns, 2^n possible stratified samples

Modern data warehouses: **n ≈ 100-200**

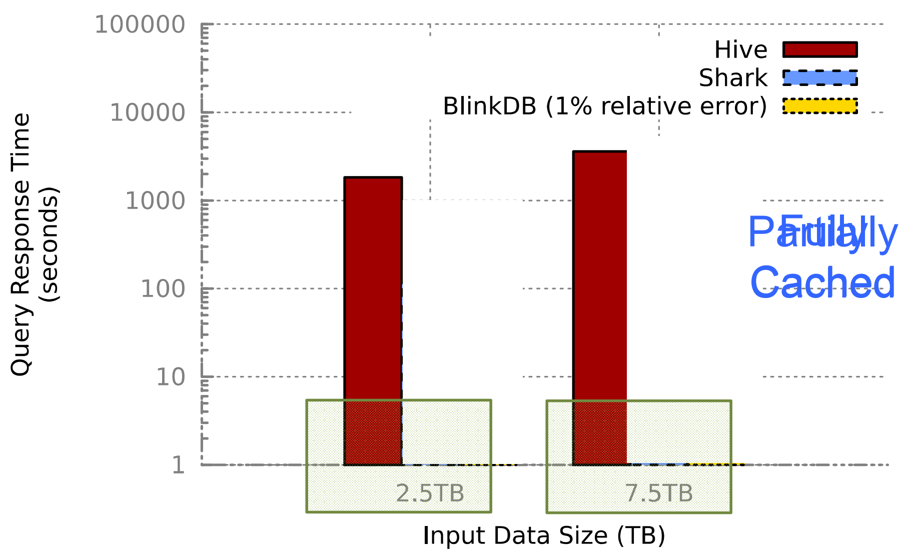
BlinkDB Solution: Choose the best set of samples by considering

1. Columns queried together
2. Data distribution
3. Storage costs

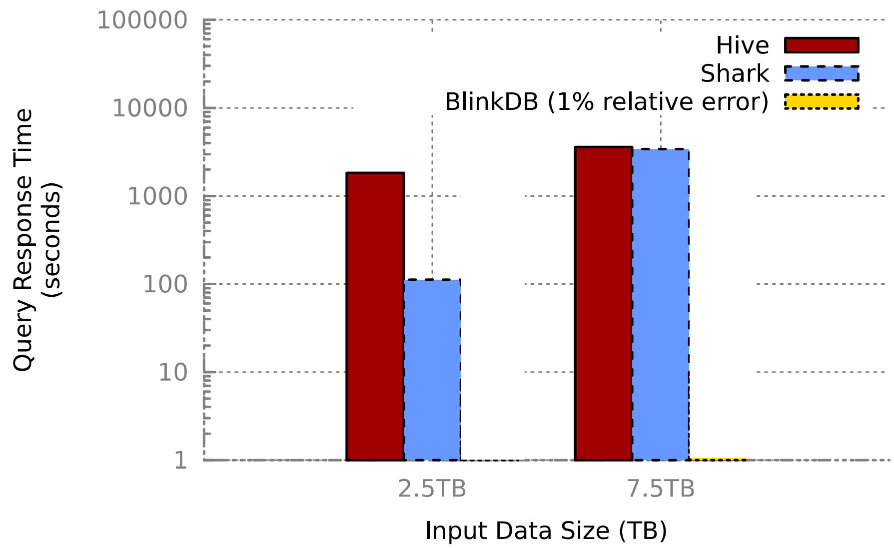
Experimental Setup

- **Conviva**: 30-day log of media accesses by Conviva users. Raw data 17 TB, partitioned this data across 100 nodes
- Log of 17,000 queries (a sample of 200 queries had 17 templates).
- 50% of storage budget: 8 Stratified Samples

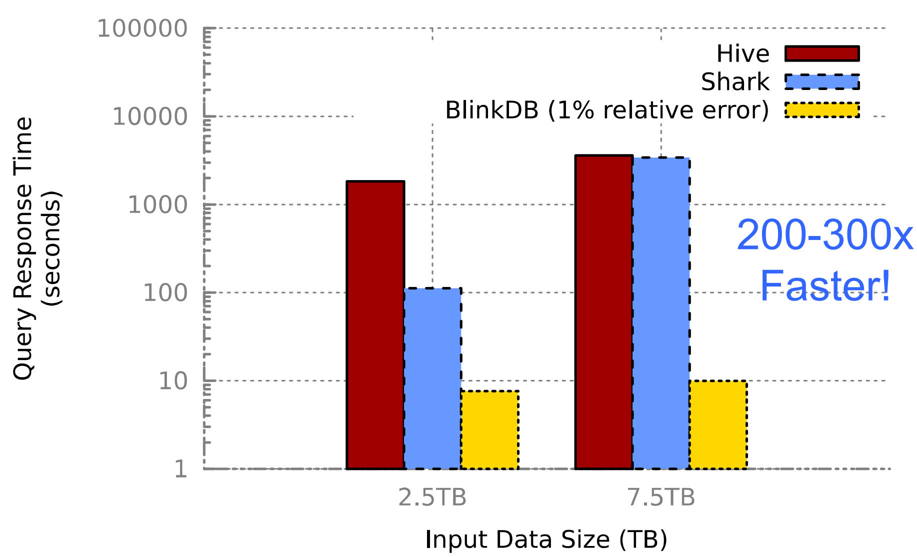
Sampling Vs. No-Sampling



BlinkDB: Evaluation



BlinkDB: Evaluation



Outline

- BlinkDB: Approximate Query Processing
- **Verdict**: Database Learning

Verdict:

DB Learning: A DB that Gets Faster as It Gets More Queries!

(Work In Progress)

Traditoinal Query Planning

1. ~~Efficiently access *all* relevant tuples~~
2. ~~Choose a single query plan out of many equivalent plans~~

Stochastic Query Planning

1. Access *a small fraction of* tuples
2. Pursue multiple plans (not necessarily equivalent)
3. Learn from past query results!

2. Pursue multiple, different plans

Q: Avg income per different health conditions

- Compute various approximations to re-calibrate the original estimate and boost accuracy

- Sampling-based estimates

Uniform / Stratified Samples

3. Learn from past queries

Time

Q1: select avg(salary)
from T where age>30

Q2: select avg(salary)
from T where job="prof"

Q3: select avg(commute)
from T where age=40

Each query is a new evidence

=> Use smaller samples

DB Learning:
DB gets smarter over time!

Verdict: A Next Generation AQP System

Verdict gets smarter over time as it learns from and uses past queries!

- In machine learning, models get smarter with more training data
- In DB learning, database gets smarter with more queries!

Verdict can use samples that are 10-100x smaller than BlinkDB, while guaranteeing (similar) accuracy

Conclusion

- Approximation is an important means to achieve interactivity in the big data age
- Ad-hoc exploratory queries on an optimal set of multi-dimensional stratified samples converges to lower errors 2-3 orders of magnitude faster than non-optimal strategies

Conclusion (cont.)

1. Once you open the door of approximations, there's no end to it!
2. Numerous new opportunities that wouldn't make sense for traditional DBs
   - **Pursuing non-equivalent plans!**
3. **DB Learning**: Databases can learn from past queries (not just reusing cached tuples!)