ApproxJoin
Approximate Distributed Joins

Do Le Quoc, Istemi Ekin Akkus, Pramod Bhatotia, Spyros Blanas, Ruichuan Chen, Christof Fetzer, Thorsten Strufe
Join is a **critical** operation in big data analytics systems, but it is very **expensive**

**Reduce the overhead** of join operations using a **sampling-based** approach
Motivation

\[ R_1 \times R_2 = R_1 \boxtimes R_2 \]

\[
\begin{array}{c|c}
A_1 & B_0 \\
A_2 & B_1 \\
A_2 & B_2 \\
\vdots \\
A_2 & B_n \\
\end{array}
\quad \times \quad
\begin{array}{c|c}
A_2 & C_0 \\
A_1 & C_1 \\
A_1 & C_2 \\
\vdots \\
A_1 & C_m \\
\end{array}
\quad = \quad
\begin{array}{c|c|c}
A_1 & B_0 & C_1 \\
A_1 & B_0 & C_2 \\
\vdots \\
A_1 & B_0 & C_m \\
A_2 & B_1 & C_0 \\
A_2 & B_2 & C_0 \\
\vdots \\
A_2 & B_n & C_0 \\
\end{array}
\]
Motivation

Sampling over joins is a challenging task regarding the output quality.
Motivation

\[
R_1 \times R_2 = R_1 \Join R_2
\]

None-join items

Unnecessary data shuffle through cluster
State-of-the-art Systems

- **AQUA (SIGMOD’99)**
  - Sampling over joins (SIGMOD’99)
  - Requiring priori knowledge of inputs (statistical info, indices)

- **RippleJoin (SIGMOD’99), WanderJoin (SIGMOD’16)**
  - Using online aggregation approach for joins

- **SparkSQL (SIGMOD’15), SnappyData (SIGMOD’16)**
  - Using pre-existing samples to serve queries
State-of-the-art Systems

- AQUA (SIGMOD’99)
  - Sampling over joins (SIGMOD’99)
  - Requiring priori knowledge of inputs (statistical info, indices)
  - Designed for single node system

- RippleJoin (SIGMOD’99), WanderJoin (SIGMOD’16)
  - Using online aggregation approach for joins

- SparkSQL (SIGMOD’15), SnappyData (SIGMOD’16)
  - Do not support sampling over joins
  - Using pre-existing samples to serve queries
Outline

• Motivation
• Design
• Evaluation
ApproxJoin: System Overview

**Input datasets**

- $R_1$
- $R_2$
- $\vdots$
- $R_n$

**ApproxJoin**

- Filtering (Bloom filters)
- Sampling over distributed join

**Approximate Result**

- Reduce shuffled data size
- Achieve Low latency

\[
\text{SELECT SUM}(R_1.V + R_2.V + \ldots + R_n.V)
\text{FROM } R_1, R_2, \ldots, R_n
\text{WHERE } R_1.A = R_2.A = \ldots = R_n.A
\text{WITHIN } 120 \text{ seconds}
\text{OR}
\text{ERROR 0.05 CONFIDENCE 95%}
\]

192.68 ± 0.05 (95% confidence)
ApproxJoin: Core Idea

Input datasets:
- \( R_1 \)
- \( R_2 \)

Build bloom filter:
- \( BF(R_1) \)
- \( BF(R_2) \)

Filter out overlap items:
- \( R_1 \) \( \text{JoinBF} \)
- \( R_2 \) \( \text{JoinBF} \)
- \( \text{Sampling} \) \( \text{Join} \) \( \text{Result} \)

\[ \text{JoinBF} = BF(R_1) \& BF(R_2) \]
ApproxJoin: Filtering

\[ R_1 \bowtie R_2 \]

\[ \begin{align*}
R_1 & = \{A_1, A_2, A_3\} \\
R_2 & = \{A_1, A_2, A_4\} \\
\text{JoinBF} & = \{A_1, A_2\}
\end{align*} \]
ApproxJoin: Sampling

\[ \text{Sample}(R_1 \Join R_2) \]

CoGroup

Stratified Sampling
ApproxJoin: Implementation

Result

192.68 ± 0.05
(95% confidence)

Error-bound estimator

Aggregation engine (Apache Spark)

Stratified sampling during join operator

Sample sizes estimator (Cost-function)

Multi-way Bloom filter constructor

Cluster configuration

Input datasets (HDFS)

SELECT SUM(R_1.V + R_2.V + ... + R_n.V)
FROM R_1, R_2, ..., R_n
WHERE R_1.A = R_2.A = ... = R_n.A
WITHIN 120 seconds
OR ERROR 0.05 CONFIDENCE 95%
Outline

• Motivation
• Design
• Evaluation
Experimental Setup

• Evaluation questions
  • Latency vs overlap fraction
  • Shuffled data size vs overlap fraction
  • Latency vs sampling fraction

• Testbed
  • Cluster: 10 nodes
  • Datasets:
    • Synthesis: Poisson distribution datasets, TPC-H
    • CAIDA Network traffic traces; Netflix Prize

See the paper for more results!
Latency

Lower is better

ApproxJoin  Spark repartition join  Native Spark join

~2.6X and ~8X faster than Spark repartition join and native Spark join with overlap fraction of 1%
Shuffled Data Size

ApproxJoin  Spark repartition join  Native Spark join

~29X and ~26X lower shuffled data size compared to Spark repartition join and native Spark join with overlap fraction of 1%
Latency

(3X – 7X) faster than Spark with sampling after join
(1.01X – 1.3X) slower than Spark with sampling before join
Comparable accuracy to Spark with sampling after join
~42X more accurate than Spark with sampling before join
Outline

• Motivation
• Our work
• Conclusion
Conclusion

**ApproxJoin**: Approximate Distributed Joins

- **Transparent**: Supports applications w/ minor code changes
- **Practical**: Adaptive execution based on query budget
- **Efficient**: Employs sketch & sampling techniques

Thank you!