Efficient Parallel kNN Joins for Large Data in MapReduce

Chi Zhang\textsuperscript{1}  Feifei Li\textsuperscript{2}  Jeffrey Jestes\textsuperscript{2}

\textsuperscript{1}Dept of Computer Science  Florida State University
\textsuperscript{2}School of Computing  University of Utah

April 4, 2012
Outline

1. Introduction

2. Background: \( k \text{NN Join} \)

3. Parallel \( k \text{NN Join} \) for Multi-dimensional Data Using MapReduce
   - Exact \( k \text{NN Join} \)
   - Approximate \( k \text{NN Join} \)

4. Experiments

5. Conclusions
Outline

1 Introduction

2 Background: $k$NN Join

3 Parallel $k$NN Join for Multi-dimensional Data Using MapReduce
   • Exact $k$NN Join
   • Approximate $k$NN Join

4 Experiments

5 Conclusions
**k Nearest Neighbor Join**

- **k nearest neighbor join (kNN join)**
  - Given two data sets $R$ and $S$, for every point $q$ in $R$, kNN join returns $k$ nearest points of $q$ from $S$. 

---

![Diagram showing k nearest neighbor join](image)
k Nearest Neighbor Join

- k nearest neighbor join (kNN join)
  - Given two data sets $R$ and $S$, for every point $q$ in $R$, kNN join returns $k$ nearest points of $q$ from $S$.

3-NN join for $q$

$$(q, p_1)$$
$$(q, p_3)$$
$$(q, p_4)$$
**k Nearest Neighbor Join**

- **k nearest neighbor join (kNN join)**
  - Given two data sets $R$ and $S$, for every point $q$ in $R$, kNN join returns $k$ nearest points of $q$ from $S$.

Find $k$NN in $S$ for all points in $R$
**k Nearest Neighbor Join**

- **k nearest neighbor join (kNN join)**
  - Given two data sets $R$ and $S$, for every point $q$ in $R$, kNN join returns $k$ nearest points of $q$ from $S$.

- Numerous applications: knowledge discovery, data mining, spatial databases, multimedia databases, etc.

![Diagram](image)
Data Growth

Exabytes Created

Source: IDC
Data sets are growing at an exponential rate.

- A single machine cannot handle large data efficiently.
- Parallel and distributed computing is the trend.
Data sets are growing at an exponential rate.

A single machine cannot handle large data efficiently.

Parallel and distributed computing is the trend.
Rise of Distributed and Parallel Computing

- Challenges:
  - Minimize communication and computation.
  - Achieve good load balance.

Chi Zhang, Feifei Li, Jeffrey Jestes
Efficient Parallel kNN Joins for Large Data in MapReduce
Outline

1. Introduction

2. Background: $k$NN Join

3. Parallel $k$NN Join for Multi-dimensional Data Using MapReduce
   - Exact $k$NN Join
   - Approximate $k$NN Join

4. Experiments

5. Conclusions
**kNN Join**

- **Exact kNN Join**
  - \( knn(r, S) \) = set of kNN of \( r \) from \( S \).
  - \( knnJ(R, S) = \{(r, knn(r, S))| \text{ for all } r \in R\} \).

- Approximate kNN Join
  - \( aknn(r, S) \) = approximate kNN of \( r \) from \( S \).
  - \( p = k \text{th NN of } r \) in \( knn(r, S) \).
  - \( p' = k \text{th NN for } r \) in \( aknn(r, S) \).
  - \( aknnJ(R, S) = \{(r, aknn(r, S))| \forall r \in R\} \).

---

Chi Zhang, Feifei Li, Jeffrey Jestes

EFFICIENT PARALLEL kNN JOINS FOR LARGE DATA IN MAPREDUCE
**kNN Join**

- **Exact kNN Join**
  - $knn(r, S) =$ set of kNN of $r$ from $S$.
  - $knnJ(R, S) = \{(r, knn(r, S))| $ for all $r \in R\}.$

- **Approximate kNN Join**
  - $aknn(r, S) =$ approximate kNN of $r$ from $S$.
    - $p = k$th NN of $r$ in $knn(r, S)$.
    - $p' = k$th NN for $r$ in $aknn(r, S)$
    - $aknn(r, S)$ is a $c$-approximation of $knn(r, S) : d(r, p) \leq d(r, p') \leq c \cdot d(r, p)$.
  - $aknnJ(R, S) = \{(r, aknn(r, S))| \forall r \in R\}.$
Outline

1 Introduction

2 Background: kNN Join

3 Parallel kNN Join for Multi-dimensional Data Using MapReduce
   - Exact kNN Join
   - Approximate kNN Join

4 Experiments

5 Conclusions
Exact $k$NN join: Block Nested Loop Join

- Block nested loop join (BNLJ) based method
Exact $k$NN join: Block Nested Loop Join

- Block nested loop join (BNLJ) based method
  1. Partition $R$ and $S$, each into $n$ equal-sized disjoint blocks.

![Diagram showing partitioning of sets $R$ and $S$ into blocks $R_1$, $R_2$, $S_1$, and $S_2$.]
Exact kNN join: Block Nested Loop Join

- Block nested loop join (BNLJ) based method
  1. Partition $R$ and $S$, each into $n$ equal-sized disjoint blocks.
  2. Perform (BNLJ) for each possible $R_i, S_j$ pairs of blocks

![Diagram showing partitioning of $R$ and $S$ into blocks $R_1$, $R_2$, $S_1$, and $S_2$.]
Exact $k$NN join: Block Nested Loop Join

- Block nested loop join (BNLJ) based method
  1. Partition $R$ and $S$, each into $n$ equal-sized disjoint blocks.
  2. Perform (BNLJ) for each possible $R_i, S_j$ pairs of blocks

\[
\begin{array}{c}
R \\
\downarrow \\
S \\
\end{array} \quad \rightarrow \quad \begin{array}{c}
R_1 \\
\downarrow \\
R_2 \\
\downarrow \\
S_1 \\
\downarrow \\
S_2 \\
\end{array}
\]
Exact $k$NN join: Block Nested Loop Join

- Block nested loop join (BNLJ) based method
  1. Partition $R$ and $S$, each into $n$ equal-sized disjoint blocks.
  2. Perform (BNLJ) for each possible $R_i, S_j$ pairs of blocks

Chi Zhang, Feifei Li, Jeffrey Jestes
Efficient Parallel $k$NN Joins for Large Data in MapReduce
Exact kNN join: Block Nested Loop Join

- Block nested loop join (BNLJ) based method
  1. Partition $R$ and $S$, each into $n$ equal-sized disjoint blocks.
  2. Perform (BNLJ) for each possible $R_i, S_j$ pairs of blocks

![Diagram showing the process of BNLJ](image)
Exact $k$NN join: Block Nested Loop Join

- Block nested loop join (BNLJ) based method
  1. Partition $R$ and $S$, each into $n$ equal-sized disjoint blocks.
  2. Perform (BNLJ) for each possible $R_i, S_j$ pairs of blocks.
  3. Get global $k$NN results from $n$ local $k$NN results for every record in $R$.
Exact $k$NN join: Block Nested Loop Join

Two-round MapReduce algorithm: Round 1

(1) Divide $R$ and $S$ into blocks
(2) Duplicate each blocks into 2 partitions
Two-round MapReduce algorithm: Round 1

1. Divide R and S into blocks
2. Duplicate each blocks into 2 partitions

(1) Divide R and S into blocks
(2) Duplicate each blocks into 2 partitions
Exact $k$NN join: Block Nested Loop Join

Two-round MapReduce algorithm: Round 2

File 1

\[(r_1, s_1, d_{1,1})\]
\[\vdots\]
\[(r_3, s_1, d_{3,1})\]

\[\text{partition by record } ids\]

File 2

\[(r_1, s_7, d_{1,8})\]
\[\vdots\]
\[(r_3, s_5, d_{3,5})\]

Mapper

\[(r_1, s_1, d_{1,1})\]
\[\vdots\]

\[(r_3, s_1, d_{3,1})\]
\[\vdots\]

\[(r_1, s_7, d_{1,8})\]
\[\vdots\]

\[(r_3, s_5, d_{3,5})\]
\[\vdots\]
exact $k$NN join: Block Nested Loop Join

Two-round MapReduce algorithm: Round 2

File 1

(r$_1$, s$_1$, d$_1$,1)

(r$_3$, s$_1$, d$_3$,1)

Mapper

partition by record $ids$

(r$_1$, s$_7$, d$_1$,8)

(r$_3$, s$_5$, d$_3$,5)

File 2

Mapper

Shuffle

sort $list(s, d(r, s))$

get top $k(=2)$ results for $r$

Reducer

DFS

(r$_1$, s$_1$, d$_1$,1)

(r$_3$, s$_1$, d$_3$,1)

(r$_1$, s$_7$, d$_1$,7)

(r$_3$, s$_5$, d$_3$,5)

(r$_3$, s$_6$, d$_3$,6)

DFS

Reducer

(r$_1$, s$_1$, d$_1$,1)

(r$_3$, s$_1$, d$_3$,1)

(r$_1$, s$_7$, d$_1$,7)

(r$_3$, s$_5$, d$_3$,5)

Chi Zhang, Feifei Li, Jeffrey Jestes

Efficient Parallel kNN Joins for Large Data in MapReduce
Exact $k$NN join: Block R-tree Join

- Use spatial index (R-tree) to improve performance
Exact $k$NN join: Block R-tree Join

- Use spatial index (R-tree) to improve performance
  - Build R-tree index for a block of $S$ in a bucket to speed up $k$NN computations.
  - Similar to BNLJ algorithm, only need to replace BNLJ with block R-tree join (BRJ) in the first round.
Exact $k$NN join: Block R-tree Join

- Use spatial index (R-tree) to improve performance
  - Build R-tree index for a block of $S$ in a bucket to speed up $k$NN computations.
  - Similar to BNLJ algorithm, only need to replace BNLJ with block R-tree join (BRJ) in the first round.

(1) Divide $R$ and $S$ into blocks
(2) Duplicate each blocks into 2 partitions
(3) Shuffle
(4) Reduce

Chi Zhang, Feifei Li, Jeffrey Jestes
Efficient Parallel $k$NN Joins for Large Data in MapReduce
Exact $k$NN join: Block R-tree Join

- Use spatial index (R-tree) to improve performance
  - Build R-tree index for a block of $S$ in a bucket to speed up $k$NN computations.
  - Similar to BNLJ algorithm, only need to replace BNLJ with block R-tree join (BRJ) in the first round.

(1) Divide $R$ and $S$ into blocks
(2) Duplicate each blocks into 2 partitions

Chi Zhang, Feifei Li, Jeffrey Jestes
Efficient Parallel kNN Joins for Large Data in MapReduce
Outline

1. Introduction
2. Background: kNN Join
   • Parallel kNN Join for Multi-dimensional Data Using MapReduce
     • Exact kNN Join
     • Approximate kNN Join
3. Experiments
4. Conclusions
Approximate $k$NN join

- Problems with exact $k$NN join solution

We search for approximate solutions.

Space-filling curve based methods ([YLK10], dubbed zkNN)

---

Chi Zhang, Feifei Li, Jeffrey Jestes

Efficient Parallel kNN Joins for Large Data in MapReduce
Approximate kNN join

- Problems with exact kNN join solution
  - Too much communication and computation ($n^2$ buckets required)

We search for approximate solutions.

Space-filling curve based methods ([YLK10], dubbed zkNN)

DFS

R

S

Mapper

(1) Divide $R$ and $S$ into blocks
(2) Duplicate each blocks into 2 partitions

Shuffle

$R_1$ $R_2$

$R_1$ $R_2$

$S_1$ $S_2$

$S_1$ $S_2$

$n^2$ buckets required, too much cost.

BRJ

Reducer

DFS

DFS

DFS

DFS

Chi Zhang, Feifei Li, Jeffrey Jestes

Efficient Parallel kNN Joins for Large Data in MapReduce
Approximate $k$NN join

- Problems with exact $k$NN join solution
  - Too much communication and computation ($n^2$ buckets required)
  - Find solution requiring $O(n)$ buckets.

![Diagram](#)

- $n^2$ buckets required, too much cost.

1. Divide $R$ and $S$ into blocks
2. Duplicate each blocks into 2 partitions

---

Chi Zhang, Feifei Li, Jeffrey Jestes

Efficient Parallel $k$NN Joins for Large Data in MapReduce
Approximate $k$NN join

- Problems with exact $k$NN join solution
  - Too much communication and computation ($n^2$ buckets required)
- Find solution requiring $O(n)$ buckets.
  - We search for approximate solutions.
  - Space-filling curve based methods ([YLK10], dubbed zkNN)

$n^2$ buckets required, too much cost.

Approximate $k$NN join: Z-order $k$NN join

- The idea of zkNN
  - Transform $d$-dimensional points to 1-D values using Z-value.
  - Map $d$-dimensional $k$NN join query to to 1-D range queries.
  - Multiple random shift copies are used to improve spatial locality.
    - In practice 2 copies is already good enough.
Approximate $k$NN join: $Z$-order $k$NN join

- The idea of zkNN
  - Transform $d$-dimensional points to 1-D values using $Z$-value.
  - Map $d$-dimensional $k$NN join query to 1-D range queries.
  - Multiple random shift copies are used to improve spatial locality.
    - In practice 2 copies is already good enough.

$\begin{array}{c|c|c}
 p_1 & & \\
\hline
 p_4 & p_5 & \\
\hline
 p_2 & & p_3 \\
\hline
 p_6 & & \\
\end{array}$

$\bullet$: points in $P$
Approximate $k$NN join: $Z$-order $k$NN join

- The idea of zkNN
  - Transform $d$-dimensional points to 1-D values using $Z$-value.
  - Map $d$-dimensional $k$NN join query to 1-D range queries.
  - Multiple random shift copies are used to improve spatial locality.
    - In practice 2 copies is already good enough.

\[
\begin{array}{|c|c|}
\hline
p_1 & p_{i,1} \\
\hline
p_4 & p_{i,4} & p_5 & p_{i,5} \\
\hline
p_2 & p_{i,2} & p_3 & p_{i,3} \\
\hline
p_6 & p_{i,6} & & \\
\hline
\end{array}
\]

\begin{itemize}
\item \textbullet{} : points in $P$
\item \textcircled{\textbullet{}} : points in $P_i$
\end{itemize}
Approximate $k$NN join: $Z$-order $k$NN join

- The idea of zkNN
  - Transform $d$-dimensional points to 1-D values using $Z$-value.
  - Map $d$-dimensional $k$NN join query to 1-D range queries.
  - Multiple random shift copies are used to improve spatial locality.
    - In practice 2 copies is already good enough.

![Diagram of zkNN join]

- Points in $P_i$ are transformed into $Z$-values and used for range queries in MapReduce.
Approximate $k$NN join: $Z$-order $k$NN join

- The idea of zkNN
  - Transform $d$-dimensional points to 1-D values using $Z$-value.
  - Map $d$-dimensional $k$NN join query to 1-D range queries.
  - Multiple random shift copies are used to improve spatial locality.
    - In practice 2 copies is already good enough.

\[
q_i = q + \mathbf{v}_i
\]
Approximate $k$NN join: $Z$-order $k$NN join

- The idea of zkNN
  - Transform $d$-dimensional points to 1-D values using $Z$-value.
  - Map $d$-dimensional $k$NN join query to 1-D range queries.
  - Multiple random shift copies are used to improve spatial locality.
    - In practice 2 copies is already good enough.

\[ q_i = q + v_i \]

\[ z^{-}(z_{qi}, k, P_i) \xrightarrow{z^{+}(z_{qi}, k, P_i)} B^{+}\text{-tree} \]

\[ Z_{P_i} \]
Approximate $k$NN join: $Z$-order $k$NN join

- The idea of zkNN
  - Transform $d$-dimensional points to 1-D values using $Z$-value.
  - Map $d$-dimensional $k$NN join query to 1-D range queries.
  - Multiple random shift copies are used to improve spatial locality.
    - In practice 2 copies is already good enough.

\[ q_i = q + v_i \]

\[ z^{-}(z_{q_i}, k, P_i) \]

\[ z^{+}(z_{q_i}, k, P_i) \]

\[ C_i(q) \]

\[ Z_{P_i} \]

\[ B^+\text{-tree} \]

\( p_1, p_2, p_3, p_4, p_5, p_6, p_{i,1}, p_{i,2}, p_{i,3}, p_{i,4}, p_{i,5}, p_{i,6} \)

\( z_{i,1}, z_{i,2}, z_{i,3}, z_{i,4}, z_{i,5}, z_{i,6} \)

\( q_i = q + v_i \)

\( z^{-}(z_{q_i}, k, P_i) \)

\( z^{+}(z_{q_i}, k, P_i) \)

\( C_i(q) \)

\( Z_{P_i} \)

\[ B^+\text{-tree} \]
Approximate kNN join: Z-order kNN join

- In our group’s previous work we derive the following guarantee for the zkNN join:

**Theorem**

*Given a query point* \( q \in \mathbb{R}^d \), a data set *\( P \subset \mathbb{R}^d \)*, and a small constant *\( \alpha \in \mathbb{Z}^+ \).* We generate \( (\alpha - 1) \) random vectors \( \{v_2, \ldots, v_\alpha\} \), such that for any *\( i \)*, \( v_i \in \mathbb{R}^d \), and shift *\( P \) by these vectors to obtain \( \{P_1, \ldots, P_\alpha\} \) (\( P_1 = P \)). Then, the zkNN join returns a constant approximation for \( \text{knn}(q, P) \) in expectation.*
Approximate $k$NN join: H-zkNNJ

- Apply zkNN for join in MapReduce (H-zkNNJ)
- Partition based algorithm
  - Partitioning policy:
    - To achieve linear communication and computation costs (to the number of blocks $n$ in each input data set)
Approximate $k$NN join: H-zkNNJ

- Apply zkNN for join in MapReduce (H-zkNNJ)
- Partition based algorithm
  - Partitioning policy:
    - To achieve linear communication and computation costs (to the number of blocks $n$ in each input data set)
  - Partitioning by $z$-values:
Approximate \( k \)NN join: H-zkNNJ

- Apply zkNN for join in MapReduce (H-zkNNJ)
- Partition based algorithm
  - Partitioning policy:
    - To achieve linear communication and computation costs (to the number of blocks \( n \) in each input data set)
- Partitioning by \( z \)-values:

![Diagram]

Chi Zhang, Feifei Li, Jeffrey Jestes
Efficient Parallel \( k \)NN Joins for Large Data in MapReduce
Approximate $k$NN join: H-zkNNJ

- Apply zkNN for join in MapReduce (H-zkNNJ)
- Partition based algorithm
  - Partitioning policy:
    - To achieve linear communication and computation costs (to the number of blocks $n$ in each input data set)
  - Partitioning by $z$-values:
Approximate $k$NN join: H-zkNNJ

- Apply zkNN for join in MapReduce (H-zkNNJ)
- Partition based algorithm
  - Partitioning policy:
    - To achieve linear communication and computation costs (to the number of blocks $n$ in each input data set)
  - Partitioning by $z$-values:
    - Partition input data sets $R_i$ and $S_i$ into \{${R_i, 1}, \ldots, {R_i, n}$\} and \{${S_i, 1}, \ldots, {S_i, n}$\} using \((n - 1)\) $z$-values \{${z_i, 1}, \ldots, {z_i, n}$\}

\[
\begin{align*}
&S_i, 1 \quad R_i, 1 \\
&S_i, 2 \quad R_i, 2 \\
&S_i, 3 \quad R_i, 3
\end{align*}
\]

Chi Zhang, Feifei Li, Jeffrey Jestes
Efficient Parallel $k$NN Joins for Large Data in MapReduce
Approximate $k$NN join: H-zkNNJ

- Apply zkNN for join in MapReduce (H-zkNNJ)
- Partition based algorithm
  - Partitioning policy:
    - To achieve linear communication and computation costs (to the number of blocks $n$ in each input data set)
  - Partitioning by $z$-values:
    - Partition input data sets $R_i$ and $S_i$ into $\{R_{i,1}, \ldots, R_{i,n}\}$ and $\{S_{i,1}, \ldots, S_{i,n}\}$ using $(n - 1)$ $z$-values $z_{i,1}, \ldots, z_{i,n}$

small neighborhood search!!!

Chi Zhang, Feifei Li, Jeffrey Jestes

Efficient Parallel kNN Joins for Large Data in MapReduce
Approximate $k$NN join: H-zkNNJ

- Apply zkNN for join in MapReduce (H-zkNNJ)
- Partition based algorithm
  - Partitioning policy:
    - To achieve linear communication and computation costs (to the number of blocks $n$ in each input data set)
  - Partitioning by $z$-values:
    - Partition input data sets $R_i$ and $S_i$ into $\{R_{i,1}, \ldots, R_{i,n}\}$ and $\{S_{i,1}, \ldots, S_{i,n}\}$ using $(n-1)$ $z$-values $\{z_{i,1}, \ldots, z_{i,n}\}$
Approximate $k$NN join: H-zkNNJ

- Apply zkNN for join in MapReduce (H-zkNNJ)
- Partition based algorithm
  - Partitioning policy:
    - To achieve linear communication and computation costs (to the number of blocks $n$ in each input data set)
  - Partitioning by $z$-values:
    - Partition input data sets $R_i$ and $S_i$ into $\{R_{i,1}, ..., R_{i,n}\}$ and $\{S_{i,1}, ..., S_{i,n}\}$ using $(n-1)$ $z$-values $\{z_{i,1}, ..., z_{i,n}\}$

\[
\begin{align*}
\text{Partitioning by } z\text{-values:} & \\
\text{Partition input data sets } R_i & \text{ and } S_i \text{ into } \{R_{i,1}, ..., R_{i,n}\} \text{ and } \\
\text{} & \{S_{i,1}, ..., S_{i,n}\} \text{ using } (n-1) \text{ } z\text{-values } \{z_{i,1}, ..., z_{i,n}\}
\end{align*}
\]
Approximate $k$NN join: H-zkNNJ

- Apply zkNN for join in MapReduce (H-zkNNJ)
- Partition based algorithm
  - Partitioning policy:
    - To achieve linear communication and computation costs (to the number of blocks $n$ in each input data set)
  - Partitioning by $z$-values:
    - Partition input data sets $R_i$ and $S_i$ into $\{R_{i,1}, \ldots, R_{i,n}\}$ and $\{S_{i,1}, \ldots, S_{i,n}\}$ using $(n - 1)$ $z$-values $\{z_{i,1}, \ldots, z_{i,n}\}$
Approximate $k$NN join: H-zkNNJ

- Choice of partitioning values.
  - Each block of $R_i$ and $S_i$ shares the same boundary so we only search a small neighborhood and minimize communication.
  - Goal: load balance.
Choice of partitioning values.
- Each block of $R_i$ and $S_i$ shares the same boundary so we only search a small neighborhood and minimize communication.
- Goal: load balance.
- Evenly partition $R_i$ or $S_i$. 
Approximate $k$NN join: H-zkNNJ

- Choice of partitioning values.
  - Each block of $R_i$ and $S_i$ shares the same boundary so we only search a small neighborhood and minimize communication.
  - Goal: load balance.
  - Evenly partition $R_i$ or $S_i$.
    - Evenly partition $R_i \rightarrow O\left(\frac{|R_i|}{n} \log |S_i|\right)$
    - Evenly partition $S_i \rightarrow O(|R_i| \log |S_i|)$
Computation of partitioning values.
- Quantiles can be used for evenly partitioning a data set $D$.
- Sort a data set $D$ and retrieve its $(n - 1)$ quantiles (expensive).
Approximate $k$NN join: H-$zk$NNJ

- Computation of partitioning values.
  - Quantiles can be used for evenly partitioning a data set $D$.
  - Sort a data set $D$ and retrieve its $(n-1)$ quantiles (expensive).
- We propose sampling based method to estimate quantiles.
  - We proved that both estimations are close enough (within $\epsilon N$) to the original ranks with a high probability ($1-e^{-2/\epsilon}$).
Approximate $k$NN join: H-zkNNJ

- $H – zkNNJ$ algorithm can be implemented in 3 rounds of MapReduce.
  - Round 1: construct random shift copies for $R$ and $S$, $R_i$ and $S_i$, $i \in [1, \alpha]$, and generate partitioning values for $R_i$ and $S_i$
Approximate $k$NN join: H-zkNNJ

- $H$–zkNNJ algorithm can be implemented in 3 rounds of MapReduce.

- Round 1: construct random shift copies for $R$ and $S$, $R_i$ and $S_i$, $i \in [1, \alpha]$, and generate partitioning values for $R_i$ and $S_i$. 

```
$\begin{array}{c}
\text{Map} \\
$\begin{array}{c}
R \\
\text{shift by } v_i \\
\text{compute } z\text{-value} \\
S
\end{array}
\end{array}$

$\begin{array}{c}
\text{DFS} \\
R_i \\
\text{sample of } i\text{th shift} \\
\widehat{R}_i
\end{array}$

$\begin{array}{c}
S_i \\
\text{sample of } i\text{th shift} \\
\widehat{S}_i
\end{array}$

$\begin{array}{c}
\text{shuffle } & \text{sort} \\
\widehat{R}_i & \widehat{S}_i
\end{array}$

$\begin{array}{c}
\text{DFS} \\
estimator 1 \\
estimator 2 \\
\text{Reduce}
\end{array}$
```

Chi Zhang, Feifei Li, Jeffrey Jestes
Approximate $k$NN join: H-zkNNJ

- $H$–zkNNJ algorithm can be implemented in 3 rounds of MapReduce.
  - Round 2: partition $R_i$ and $S_i$ into blocks and compute the candidate points for $\text{knn}(r, S)$ for any $r \in R$.

```
\[\begin{array}{c}
| R_i \downarrow | R_{i,1} | R_{i,2} | \ldots | R_{i,n} \\
| Si \downarrow | S_{i,1} | S_{i,2} | \ldots | S_{i,n} \\
\end{array}\]
```

- partition by $R_i$’s ranges
- partition by $S_i$’s ranges

Map
Approximate \( k \text{NN join: H-zkNNJ} \)

- \( H – zkNNJ \) algorithm can be implemented in 3 rounds of MapReduce.
  - Round 2: partition \( R_i \) and \( S_i \) into blocks and compute the candidate points for \( knn(r, S) \) for any \( r \in R \).

Retrieve \( C_i(r) \) for all \( r \in R_{i,j}, j \in [1, n] \)

Chi Zhang, Feifei Li, Jeffrey Jestes

Efficient Parallel kNN Joins for Large Data in MapReduce
Approximate $k$NN join: H-zkNNJ

- $H – zkNNJ$ algorithm can be implemented in 3 rounds of MapReduce.
  
  - Round 3: determine $\text{knn}(r, C(r))$ of any $r \in R$ from the $(r, C_i(r))$ emitted by round 2.

Retrieve $C_i(r)$ for all $r \in R_{i,j}$, $j \in [1, n]$
Outline

1 Introduction

2 Background: \( k \text{NN Join} \)

3 Parallel \( k \text{NN Join for Multi-dimensional Data Using MapReduce} \)
   - Exact \( k \text{NN Join} \)
   - Approximate \( k \text{NN Join} \)

4 Experiments

5 Conclusions
We implement the following methods in Hadoop 0.20.2:

- **Exact Methods:**
  - The baseline solution is denoted $H$-$BNLJ$,
  - The improvement to the baseline solution is denoted $H$-$BRJ$.

- **Approximate Methods:**
  - Our three-round solution is denoted by $H$-$z$-$kNNJ$, (meaning “Hadoop $z$-value $k$NN Join”).
Experiments: setup

Experiments are performed in a heterogeneous Hadoop cluster with 17 machines:

1. 9 machines with 2GB of RAM and an Intel Xeon 1.86GHz CPU
2. 6 machines with 4GB of RAM and an Intel Xeon 2GHz CPU
   - One is reserved for the master (running JobTracker and NameNode).
3. 2 machines with 6GB of RAM and an Intel Xeon 2.13GHz CPU
   - All machines are directly connected to a 1000Mbps switch.
   - Each slave node has 300GB hard drive space and 1GB of RAM for Hadoop daemon.
   - The chunk size of DFS is set to 128MB.
Experiments: setup

- Experiments are performed in a heterogeneous Hadoop cluster with 17 machines:
  1. 9 machines with 2GB of RAM and an Intel Xeon 1.86GHz CPU
  2. 6 machines with 4GB of RAM and an Intel Xeon 2GHz CPU
     - One is reserved for the master (running JobTracker and NameNode).
  3. 2 machines with 6GB of RAM and an Intel Xeon 2.13GHz CPU
- All machines are directly connected to a 1000Mbps switch.
- Each slave node has 300GB hard drive space and 1GB of RAM for Hadoop daemon.
- The chunk size of DFS is set to 128MB.
Experiments: datasets

- **OpenStreet Map dataset:**
  - the road-networks for 50 states in U.S.
  - 160 million records.
  - preprocessed to remove duplications
  - each record consists of a 4 bytes integer id, two 4 bytes real type coordinates representing latitude and longitude, and a description information.
  - the coordinates has a positive real domain (0,100000).
  - stored in text format, 6.6GB.
Experiments: datasets

- **OpenStreet Map dataset:**
  - the road-networks for 50 states in U.S.
  - 160 million records.
  - preprocessed to remove duplications
  - each record consists of a 4 bytes integer id, two 4 bytes real type coordinates representing latitude and longitude, and a description information.
  - the coordinates has a positive real domain (0,100000).
  - stored in text format, 6.6GB.

- **Large synthetic Random-Cluster datasets:**
  - data sets have varying dimensionality (up to 30).
  - each record has a 4-byte id and float type $d$-dimensional coordinates.
Data set configurations

- $(MXN)$ represents a data set configuration containing $M$ records of $R$ and $N$ record of $S$ (in 10s of millions).
Experiments: configurations and defaults

- Data set configurations
  - $(MXN)$ represents a data set configuration containing $M$ records of $R$ and $N$ record of $S$ (in 10s of millions).

- Default values for OpenStreet dataset:

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Definition</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>$(MXN)$</td>
<td>data set configuration</td>
<td>$(4 \times 4)$</td>
</tr>
<tr>
<td>$k$</td>
<td># of nearest neighbor</td>
<td>10</td>
</tr>
<tr>
<td>$\alpha$</td>
<td># of shift copies</td>
<td>2</td>
</tr>
<tr>
<td>$\epsilon$</td>
<td>the error rate of sampling</td>
<td>0.003</td>
</tr>
<tr>
<td>$\gamma$</td>
<td>the physical number of machines</td>
<td>16</td>
</tr>
</tbody>
</table>

Values for R-Cluster dataset: $(2 \times 2)$ is set to be the default data set configuration.
Experiments: configurations and defaults

- Data set configurations
  - $(MXN)$ represents a data set configuration containing $M$ records of $R$ and $N$ record of $S$ (in 10s of millions).

- Default values for OpenStreet dataset:

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Definition</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>$(MXN)$</td>
<td>data set configuration</td>
<td>(4x4)</td>
</tr>
<tr>
<td>$k$</td>
<td># of nearest neighbor</td>
<td>10</td>
</tr>
<tr>
<td>$\alpha$</td>
<td># of shift copies</td>
<td>2</td>
</tr>
<tr>
<td>$\epsilon$</td>
<td>the error rate of sampling</td>
<td>0.003</td>
</tr>
<tr>
<td>$\gamma$</td>
<td>the physical number of machines</td>
<td>16</td>
</tr>
</tbody>
</table>

- Values for R-Cluster dataset:
  - (2x2) is set to be the default data set configuration.
Experiments: Approximation quality

$H$-$zkNNJ$: Hadoop $z$-value $k$NN Join

Approximation ratio

$k$ values

OpenStreet

Chi Zhang, Feifei Li, Jeffrey Jestes

Efficient Parallel kNN Joins for Large Data in MapReduce
Experiments: Approximation quality

$H-zkNNJ$: Hadoop $z$-value $k$NN Join

Recall (Precision) vs $k$ values

OpenStreet

Chi Zhang, Feifei Li, Jeffrey Jestes

Efficient Parallel $k$NN Joins for Large Data in MapReduce
Experiments: Approximation quality

$H$-$zkNNJ$: Hadoop z-value $k$NN Join

Approximation ratio vs Dimensionality for $R$-Cluster
Experiments: Approximation quality

$H$-zkNNJ: Hadoop z-value $k$NN Join

Recall (Precision) vs. Dimensionality

Chi Zhang, Feifei Li, Jeffrey Jestes

Efficient Parallel $k$NN Joins for Large Data in MapReduce
Experiments: Running time and communication cost

\( H-zkNNJ \): Hadoop z-value kNN Join
\( H-BRJ \): Hadoop Block R-tree Join

\[ |R| \times |S|: 10^7 \times 10^7 \]

Chi Zhang, Feifei Li, Jeffrey Jestes

Efficient Parallel kNN Joins for Large Data in MapReduce
Experiments: Running time and communication cost

$H$-zkNNJ: Hadoop z-value kNN Join

$H$-BRJ: Hadoop Block R-tree Join

Chi Zhang, Feifei Li, Jeffrey Jestes

Efficient Parallel kNN Joins for Large Data in MapReduce
Experiments: Effect of $d$

$H$-zkNNJ: Hadoop $z$-value $k$NN Join

$H$-BRJ: Hadoop Block R-tree Join

Chi Zhang, Feifei Li, Jeffrey Jestes
Efficient Parallel $k$NN Joins for Large Data in MapReduce
Experiments: Effect of $d$

$H$-$zkNNJ$: Hadoop z-value kNN Join

$H$-$BRJ$: Hadoop Block R-tree Join

Chi Zhang, Feifei Li, Jeffrey Jestes

Efficient Parallel kNN Joins for Large Data in MapReduce
Conclusions

- We study efficient methods to perform $k$NN joins in MapReduce.
  - Exact (H-BRJ) and approximate (H-zkNNJ) algorithms are proposed.
  - H-zkNNJ performs orders of magnitude better than other methods with excellent approximation quality.
- We plan to investigate $k$NN joins on very high dimensions in the future.
The End

Thank You

Q and A
Approximate \( k \)NN join: \( Z \)-order \( k \)NN join

- zkNN algorithm

**Algorithm 1**: zkNN\((q, P, k, \alpha)\)

1. generate \( \{v_2, \ldots, v_\alpha\} \), \( v_1 = \overrightarrow{0} \), \( v_i \) is a random vector in \( \mathbb{R}^d \);
2. \( P_i = P + v_i \) \( (i \in [1, \alpha]; \ \forall p \in P, \text{ insert } p + v_i \text{ in } P_i) \);
3. for \( i = 1, \ldots, \alpha \) do
   4. let \( q_i = q + v_i \), \( C_i(q) = \emptyset \), and \( z_{q_i} \) be \( q_i \)'s \( z \)-value;
   5. insert \( z^-\left(z_{q_i}, k, P_i\right) \) into \( C_i(q) \);
   6. insert \( z^+\left(z_{q_i}, k, P_i\right) \) into \( C_i(q) \);
   7. for any \( p \in C_i(q) \), update \( p = p - v_i \);
8. \( C(q) = \bigcup_{i=1}^\alpha C_i(q) = C_1(q) \cup \cdots \cup C_\alpha(q) \);
9. return \( \text{knn}(q, C(q)) \).
Experiments: Approximation quality

$H$-zkNNJ: Hadoop $z$-value $k$NN Join

Approximation ratio

$|R| \times |S|: 10^7 \times 10^7$

- 4x4
- 6x6
- 8x8
- 12x12
- 16x16

OpenStreet
Experiments: Approximation quality

$H$-zkNNJ: Hadoop z-value kNN Join

Chi Zhang, Feifei Li, Jeffrey Jestes
Efficient Parallel kNN Joins for Large Data in MapReduce
Experiments: Approximation quality

$H$-zkNNJ: Hadoop z-value $k$NN Join

Recall (Precision)

OpenStreet

Chi Zhang, Feifei Li, Jeffrey Jestes

Efficient Parallel kNN Joins for Large Data in MapReduce
Experiments: Approximation quality

$H$-$zkNNJ$: Hadoop $z$-value $k$NN Join

Recall (Precision)

$k$ values

OpenStreet

Chi Zhang, Feifei Li, Jeffrey Jestes

Efficient Parallel kNN Joins for Large Data in MapReduce
Experiments: Effect of $\varepsilon$

![Graph showing the effect of $\varepsilon$ on time (seconds).]
Experiments: Effect of $\varepsilon$

\begin{align*}
\text{Standard deviation} & \quad \varepsilon (\times 10^{-3}) \\
10^2 & \quad 0.6 \quad 1 \quad 3 \quad 10 \quad 100
\end{align*}

\[ \varepsilon \times 10^{-3} \]

- R blocks
- S blocks
Experiments: Evaluation of H-BNLJ

Chi Zhang, Feifei Li, Jeffrey Jestes
Efficient Parallel kNN Joins for Large Data in MapReduce
Experiments: Evaluation of H-BNLJ

Chi Zhang, Feifei Li, Jeffrey Jestes

Efficient Parallel kNN Joins for Large Data in MapReduce
Experiments: Speedup

Chi Zhang, Feifei Li, Jeffrey Jestes

Efficient Parallel kNN Joins for Large Data in MapReduce
Experiments: Speedup

- H-zkNNJ
- H-BRJ

Chi Zhang, Feifei Li, Jeffrey Jestes
Efficient Parallel kNN Joins for Large Data in MapReduce
Experiments: Running time and communication cost

$H$-zkNNJ: Hadoop z-value kNN Join

$H$-BRJ: Hadoop Block R-tree Join

\[ |R| \times |S| : 10^7 \times 10^7 \]
Experiments: Running time and communication cost

*H-zkNNJ*: Hadoop z-value kNN Join

*H-BRJ*: Hadoop Block R-tree Join

Chi Zhang, Feifei Li, Jeffrey Jestes

Efficient Parallel kNN Joins for Large Data in MapReduce
Experiments: Running time and communication cost

$H$-zkNNJ: Hadoop $z$-value $k$NN Join

$H$-BRJ: Hadoop Block R-tree Join

Chi Zhang, Feifei Li, Jeffrey Jestes

Efficient Parallel $k$NN Joins for Large Data in MapReduce
Experiments: Running time and communication cost

\( H-zkNNJ \): Hadoop z-value kNN Join
\( H-BRJ \): Hadoop Block R-tree Join

\[ \text{Data shuffled (GB)} \]

\[ |R| \times |S| : 10^7 \times 10^7 \]
Experiments: Effect of $d$

$H$-zkNNJ: Hadoop z-value kNN Join

$H$-BRJ: Hadoop Block R-tree Join

Chi Zhang, Feifei Li, Jeffrey Jestes

Efficient Parallel kNN Joins for Large Data in MapReduce
Experiments: Effect of $d$

$H$-zkNNJ: Hadoop $z$-value $k$NN Join
$H$-BRJ: Hadoop Block R-tree Join

Chi Zhang, Feifei Li, Jeffrey Jestes

Efficient Parallel kNN Joins for Large Data in MapReduce
Experiments: Effect of $d$

$H$-$zkNNJ$: Hadoop $z$-value $k$NN Join
$H$-$BRJ$: Hadoop Block R-tree Join

![Graph showing the approximation ratio vs. dimensionality for $R$-Cluster.](image)
Experiments: Effect of $d$

H-zkNNJ: Hadoop $z$-value $k$NN Join
H-BRJ: Hadoop Block R-tree Join

Recall (Precision)
Dimensionality
R-Cluster

Chi Zhang, Feifei Li, Jeffrey Jestes
Efficient Parallel $k$NN Joins for Large Data in MapReduce
Experiments: Effect of $k$

![Bar chart showing the time (seconds $\times 10^3$) for different $k$ values with phases zPhase1, zPhase2, and zPhase3.]

Chi Zhang, Feifei Li, Jeffrey Jestes
Efficient Parallel kNN Joins for Large Data in MapReduce
Experiments: Effect of $k$

![Bar chart showing the effect of $k$ values on the time taken for RPhase1 and RPhase2. The x-axis represents $k$ values (10, 20, 40, 60, 80), and the y-axis represents time (seconds $\times 10^3$). The chart indicates that as $k$ increases, the time taken for both RPhase1 and RPhase2 increases. The time for RPhase1 is generally lower than RPhase2.](image-url)
Experiments: Effect of $k$

![Graph showing the effect of $k$ on time](image)

- H-zkNNJ
- H-BRJ

Time (seconds $\times 10^3$)

$k$ values

Chi Zhang, Feifei Li, Jeffrey Jestes

Efficient Parallel kNN Joins for Large Data in MapReduce
Experiments: Effect of $k$

Data shuffled (GB)

$k$ values

- H-zkNNJ
- H-BRJ

Chi Zhang, Feifei Li, Jeffrey Jestes

Efficient Parallel kNN Joins for Large Data in MapReduce
Experiments: Effect of number of shifts $\alpha$

- $\alpha$ values:
  - 2
  - 3
  - 4
  - 5
  - 6

- Time (seconds)

- $10^3$ to $10^5$

- H-zkNNJ, H-BRJ
Experiments: Effect of number of shifts $\alpha$

![Graph showing data shuffled (GB) vs. $\alpha$ values for H-zkNNJ and H-BRJ.](image)
Experiments: Effect of number of shifts $\alpha$

![Graph showing the approximation ratio for different $\alpha$ values.](image-url)
Experiments: Effect of number of shifts $\alpha$

![Graph showing the effect of number of shifts $\alpha$ on recall (precision).](image)