BE-Tree: An Index Structure to Efficiently Match Boolean Expressions over High-dimensional Space

Mohammad Sadoghi       Hans-Arno Jacobsen

University of Toronto

June 15, 2011
1 Application Scenarios

2 Matching Problem

3 BE-Tree (Boolean Expression-Tree)

4 BE-Tree Adaptation Policies

5 Experimental Evaluation

6 Conclusions
Computational Advertising (A Billion-dollar Industry)

Advertisers and Agencies

PARTNER WITH YAHOO!
Vast, engaged audiences, deep consumer insights and superior targeting for solutions that help you meet your goals.

Yahoo! Advertising

Aol Advertising.

Original content

Yahoo! hotjobs

Search now. Get hired.

Enter Keyword(s)

City & Province (CAN)

Select a Job Category

Search Jobs
Computational Advertising (A Billion-dollar Industry)

Advertisement (BE):
- age < 32
- credit-score > 630
- num-visits > 4
- price = 150

Advertiser

Advertised:
- Sears
- Sony
- Amazon

Advertiser

Subscriptions
Computational Advertising (A Billion-dollar Industry)

Advertising Campaign

Broker

User Profiles

Online User

Clickstream

Events

Subscriptions

Advertisement (BE):
- age < 32
- credit-score > 630
- num-visits > 4
- price = 150

Advertisements:
- Sears
- Sony
- Amazon

Events:
- num-visits=13
- age=25
- credit-score=647
- price<235

"BMW X3 2008"
- car=BMW
- model=X3
- year=2008

Mohammad Sadoghi (University of Toronto)
Computational Advertising (A Billion-dollar Industry)

Application Scenarios
Language & Semantics
Intuitions
Adaptation
Evaluation
Conclusions

Advertisement (BE): age < 32
credit-score > 630
num-visits > 4
price = 150

Subscriptions

Sony
Amazon

Advertiser

Advertising Campaign

Broker

Targeted Ads

User Profiles

Online User

num-visits=13
age=25
credit-score=647
price<235

Events

Sears

Clickstream

“BMW X3 2008”
car=BMW
model=X3
year=2008

Events
Application Scenarios

1. Computational advertising (targeted advertising)
2. Computational finance (algorithmic trading)
3. Intrusion detection (deep packet inspection)
4. Real-time data analysis (data analytics)
5. Emerging mobile applications in co-spaces (location-based services)
6. Approximate string matching (data cleansing)
Application Scenarios

1. Computational advertising (targeted advertising)
2. Computational finance (algorithmic trading)
3. Intrusion detection (deep packet inspection)
4. Real-time data analysis (data analytics)
5. Emerging mobile applications in co-spaces (location-based services)
6. Approximate string matching (data cleansing)

Common Denominator

To continuously evaluate a set of predefined patterns/specifications (subscriptions) over incoming events.
Challenges Derived from Application Scenarios

Key matching problem challenges addressed in this work
Challenges Derived from Application Scenarios

**Key matching problem challenges addressed in this work**

1. Enable an expressive matching semantics that supports constraints on both the subscription and the event.

2. Handle subscriptions with expressive operators that impose conditions only on a small set of dimensions, which results in a high degree of overlap among subscriptions.

3. Scale to large collections of subscriptions with thousands of dimensions.

4. Sustain high matching rates of events in presence of frequent insertions and deletions of subscriptions.

5. Adapt to temporal changes of workload distribution (self-adjusting mechanism), i.e., avoid structure deterioration.
1. Application Scenarios

2. Matching Problem

3. BE-Tree (Boolean Expression-Tree)

4. BE-Tree Adaptation Policies

5. Experimental Evaluation

6. Conclusions
Both subscription and event are defined as Boolean expressions which are collections of predicates.
Language and Data Model

- Both subscription and event are defined as Boolean expressions which are collections of predicates.
- A predicate $P$ is a triple consisting of an attribute uniquely representing a dimension in $n$-dimensional space, an operator, and a set of values, denoted by $P(\text{attr}, \text{opt}, \text{val})$. 
Both subscription and event are defined as Boolean expressions which are collections of predicates.

A predicate $P$ is a triple consisting of an attribute uniquely representing a dimension in $n$-dimensional space, an operator, and a set of values, denoted by $P^{(\text{attr}, \text{opt}, \text{val})}$.

A predicate $P(x)$ either accepts or rejects an input $x$ such that $P : x \rightarrow \{\text{True}, \text{False}\}$, where $x \in \text{Dom}(P^{\text{attr}})$. 
Both subscription and event are defined as Boolean expressions which are collections of predicates.

A predicate $P$ is a triple consisting of an attribute uniquely representing a dimension in $n$-dimensional space, an operator, and a set of values, denoted by $P_{(attr,opt,val)}$.

A predicate $P(x)$ either accepts or rejects an input $x$ such that $P : x \rightarrow \{\text{True, False}\}$, where $x \in \text{Dom}(P_{attr})$.

Each predicate supports relational operators ($<, \leq, =, \neq, \geq, >$), set operators ($\in, \notin$), or the SQL BETWEEN operator.
Both subscription and event are defined as Boolean expressions which are collections of predicates.

A predicate $P$ is a triple consisting of an attribute uniquely representing a dimension in $n$-dimensional space, an operator, and a set of values, denoted by $P^{(\text{attr, opt, val})}$.

A predicate $P(x)$ either accepts or rejects an input $x$ such that $P: x \rightarrow \{\text{True, False}\}$, where $x \in \text{Dom}(P^{\text{attr}})$.

Each predicate supports relational operators ($<$, $\leq$, $=$, $\neq$, $\geq$, $>$), set operators ($\in$, $\notin$), or the SQL BETWEEN operator.

Formally, a Boolean expression $e$ is defined over an $n$-dimensional space as follows:

**Definition**

\[
e = \left\{ P_1^{(\text{attr, opt, val})} \land \cdots \land P_k^{(\text{attr, opt, val})} \right\},
\]

where $k \leq n$; $i, j \leq k$, $P_i^{\text{attr}} = P_j^{\text{attr}}$ iff $i = j$. 

Mohammad Sadoghi (University of Toronto)
Matching Semantics

Given an event $e$ and a set of subscriptions $S$, find all subscriptions $s_i \in S$ satisfied by $e$.

Definition

$$SQ(e) = \{ s_i | \forall P_{attr}, opt, val q(x) \in s_i, \exists P_{attr}, opt, val o(x) \in e, P_{attr} q = P_{attr} o, \exists x \in \text{Dom}(P_{attr} q), P_{q}(x) \land P_{o}(x) \}$$
Matching Semantics

Stabbing Subscription

Given an event $e$ and a set of subscriptions $S$, find all subscriptions $s_i \in S$ satisfied by $e$. 
Matching Semantics

Stabbing Subscription

*Given an event e and a set of subscriptions S, find all subscriptions \( s_i \in S \) satisfied by e.*

**Definition**

\[
SQ(e) = \left\{ s_i \mid \forall P_q^{\text{attr, opt, val}}(x) \in s_i, \exists P_o^{\text{attr, opt, val}}(x) \in e, \right. \\
\left. P_{q}^{\text{attr}} = P_{o}^{\text{attr}}, \exists x \in \text{Dom}(P_{q}^{\text{attr}}), P_q(x) \land P_o(x) \right\}
\]
1 Application Scenarios

2 Matching Problem

3 BE-Tree (Boolean Expression-Tree)

4 BE-Tree Adaptation Policies

5 Experimental Evaluation

6 Conclusions
Key Principles

Most Important Observation

To utilize the underlying high-dimensional, discrete, finite, and multi-valued attribute space that is assumed in many practical scenarios.
Key Principles

Most Important Observation

To utilize the underlying high-dimensional, discrete, finite, and multi-valued attribute space that is assumed in many practical scenarios.

Most Important Design Feature

To systematically explore the space in two iterative phases of space partitioning and space clustering (i) to cope with the curse of dimensionality (ii) to support dynamic insertion and removal of subscriptions independent of their orders.
Key Principles

Most Important Observation

*To utilize the underlying high-dimensional, discrete, finite, and multi-valued attribute space that is assumed in many practical scenarios.*

Most Important Design Feature

*To systematically explore the space in two iterative phases of space partitioning and space clustering (i) to cope with the curse of dimensionality (ii) to support dynamic insertion and removal of subscriptions independent of their orders.*

The two-phase space-cutting technique consists of

1. **Space partitioning:** global structuring to determine the best splitting dimension
2. **Space clustering:** local structuring for each partition to determine the best grouping of expressions with respect to the expressions’ range of values for each dimension
The Complete Structure

- Space Clustering
- Space Partitioning

O(1)

O(klogN)

O(logN)

Mohammad Sadoghi (University of Toronto)
The Complete Structure
The Partition Directory

Maintaining information for dimensions

dim_i

l-node

c-node

p-node

c-directory

p-directory

dim_j

p-node

c-directory

p-node

c-directory

Mohammad Sadoghi (University of Toronto)
The Complete Structure
The Cluster Directory

Maintaining information for the range of values

p-node

buckets

c-directory

[1, N]

[1, N/2] [N/2, N]

[1, N/4] [N/4, N/2] [3N/4, N]

[1] ...

[N/2]

c-node

c-node

c-node

c-node

c-node

l-node

l-node

l-node

l-node

l-node
Intuition Behind the Two-phase Space-cutting Technique

Required rules to avoid the cascading split problem and to ensure a deterministic property for altering between space partitioning and clustering.
Intuition Behind the Two-phase Space-cutting Technique

Required rules to avoid the cascading split problem and to ensure a deterministic property for altering between space partitioning and clustering.

1. **Insertion rule:** expression is always inserted into the smallest bucket that encloses it.

2. **Forced split rule:** non-atomic bucket (i.e., a multi-valued, divisible bucket) is always split before switching back to space partitioning.

3. **Merge rule:** underflowing leaf bucket is merged with its parent only if the parent bucket is not partitioned yet.
Intuition Behind the Two-phase Space-cutting Technique

Required rules to avoid the cascading split problem and to ensure a deterministic property for altering between space partitioning and clustering.

1. **Insertion rule:** expression is always inserted into the smallest bucket that encloses it.

2. **Forced split rule:** non-atomic bucket (i.e., a multi-valued, divisible bucket) is always split before switching back to space partitioning.

3. **Merge rule:** underflowing leaf bucket is merged with its parent only if the parent bucket is not partitioned yet.

**Invariance**

Every expression always resides in the smallest bucket that encloses it, and a non-atomic leaf bucket is never partitioned.
1. Application Scenarios
2. Matching Problem
3. BE-Tree (Boolean Expression-Tree)
4. BE-Tree Adaptation Policies
5. Experimental Evaluation
6. Conclusions
## Ranking Objective

A novel ranking objective that directly reduces the matching cost as opposed to a ranking that is biased towards either the least or the most popular dimension(s).
A novel ranking objective that directly reduces the matching cost as opposed to a ranking that is biased towards either the least or the most popular dimension(s).

The matching cost consists of:

1. **minimizing false candidate computations**: reduce the number of predicate evaluations before an unsatisfied predicate is reached, namely, penalizing multiple search paths and penalizing paths that produce many false candidates (**Loss function**).

2. **minimizing true candidate computations**: promote the evaluation of the common predicate among matched expressions exactly once (**Gain function**).
Self-adjustment Mechanism (Cost-based Ranking Model)

Definition

\[ \text{Rank}(n_i) = \begin{cases} 
\beta \text{Gain}(n_i) - \gamma \text{Loss}(n_i) & \text{if } n_i \text{ is a l-node} \\
\sum_{l_j \in c(n_i)} \text{Rank}(l_j) - \gamma \text{Loss}(n_i) & \text{otherwise} 
\end{cases} \]

where \(0 \leq \beta, \gamma \leq 1\) \(\text{(2)}\)

\[ \text{Loss}(n_i) = \sum_{e' \in \text{window}} m(n_i) \#	ext{discarded pred eval for } e' \mid \text{window}(n_i) \mid \text{(3)} \]

\[ \text{Gain}(l_j) = \alpha s \text{Gain}_s(l_j) + \alpha c \text{Gain}_c(l_j), \quad 0 \leq \alpha s, \alpha c \leq 1 \text{ (4)} \]

\[ \text{Gain}_s(l_j) = \#\text{subsumed pred}, \quad \text{Gain}_c(l_j) = \#\text{covered pred} \text{ (5)} \]
Self-adjustment Mechanism (Cost-based Ranking Model)

**Definition**

\[
\text{Rank}(n_i) = \begin{cases} 
\beta \text{Gain}(n_i) - \gamma \text{Loss}(n_i) & \text{if } n_i \text{ is a } l\text{-node} \\
\left( \sum_{l_j \in c(n_i)} \text{Rank}(l_j) \right) - \gamma \text{Loss}(n_i) & \text{otherwise}
\end{cases}
\]

where \( 0 \leq \beta, \gamma \leq 1 \)
Self-adjustment Mechanism (Cost-based Ranking Model)

**Definition**

\[
R\text{ank}(n_i) = \begin{cases} 
\beta \text{Gain}(n_i) - \gamma \text{Loss}(n_i) & \text{if } n_i \text{ is a l-node} \\
(\sum_{l_j \in c(n_i)} R\text{ank}(l_j)) - \gamma \text{Loss}(n_i) & \text{otherwise}
\end{cases}
\]

, where \(0 \leq \beta, \gamma \leq 1\) \hspace{1cm} (2)

\[
\text{Loss}(n_i) = \sum_{e' \in \text{window}_m(n_i)} \frac{\# \text{discarded pred eval for } e'}{|\text{window}(n_i)|}
\]

\hspace{1cm} (3)
Self-adjustment Mechanism (Cost-based Ranking Model)

Definition

\[ \text{Rank}(n_i) = \begin{cases} \beta \text{Gain}(n_i) - \gamma \text{Loss}(n_i) & \text{if } n_i \text{ is a l-node} \\ \left( \sum_{l_j \in c(n_i)} \text{Rank}(l_j) \right) - \gamma \text{Loss}(n_i) & \text{otherwise} \end{cases} \]

, where \( 0 \leq \beta, \gamma \leq 1 \) \hspace{1cm} (2)

\[ \text{Loss}(n_i) = \sum_{e' \in \text{window}_m(n_i)} \frac{\# \text{discarded pred eval for } e'}{|\text{window}(n_i)|} \] \hspace{1cm} (3)

\[ \text{Gain}(l_j) = \alpha_s \text{Gain}_s(l_j) + \alpha_c \text{Gain}_c(l_j), \quad 0 \leq \alpha_s, \alpha_c \leq 1 \] \hspace{1cm} (4)

\[ \text{Gain}_s(l_j) = \# \text{subsumed pred}, \quad \text{Gain}_c(l_j) = \# \text{covered pred} \] \hspace{1cm} (5)
BE-Tree Example (Insertion)

Insertion Sequence
\[ S_1 = [a>0, b=2, e>4] \]
\[ S_2 = [c<3, f>1, m<2] \]
\[ S_3 = [d=1, f<3] \]
\[ S_4 = [a=3] \]
\[ S_5 = [d>3, f>2] \]
\[ S_6 = [d=4, f<4] \]
\[ S_7 = [a<4, b=1] \]
\[ S_8 = [a<3] \]
\[ S_9 = [a>1] \]

Insertion Sequence
\( \{ S_1, S_2, S_3 \} \)

l-node's max \( \text{cap} \) = 3
BE-Tree Example (Insertion)

Insertion Sequence

\[ S_1 = [a>0, b=2, e>4] \]
\[ S_2 = [c<3, f>1, m<2] \]
\[ S_3 = [d=1, f<3] \]
\[ S_4 = [a=3] \]

\[ S_5 = [d>3, f>2] \]
\[ S_6 = [d=4, f<4] \]
\[ S_7 = [a<4, b=1] \]
\[ S_8 = [a<3] \]
\[ S_9 = [a>1] \]

\[ \text{c-node} \]
\[ \text{l-node} \]
\[ \{S_1, S_2, S_3, S_4\} \]

\[ \text{l-node's max } \text{cap } = 3 \]
**BE-Tree Example (Insertion)**

**Insertion Sequence**

\[ S_1 = [a>0, b=2, e>4] \]

\[ S_2 = [c<3, f>1, m<2] \]

\[ S_3 = [d=1, f<3] \]

\[ S_4 = [a=3] \]

\[ S_5 = [d>3, f>2] \]

\[ S_6 = [d=4, f<4] \]

\[ S_7 = [a<4, b=1] \]

\[ S_8 = [a<3] \]

\[ S_9 = [a>1] \]

**BE-Tree Diagram**

- **c-node**: \([1, 4]\)
- **p-directory**: \(a\)
- **l-node**: \(\{S_2, S_3\}\)
- **l-node's max cap** = 3

Mohammad Sadoghi (University of Toronto)

SIGMOD'11
BE-Tree Example (Insertion)

Insertion Sequence

$S_1 = [a>0, b=2, e>4]$

$S_2 = [c<3, f>1, m<2]$

$S_3 = [d=1, f<3]$

$S_4 = [a=3]$

$S_5 = [d>3, f>2]$

$S_6 = [d=4, f<4]$

l-node

$\{S_1, S_4\}$

c-node

$\{S_2, S_3, S_5, S_6\}$

p-node

$\{S_2\}$

c-node

$[1, 4]$
BE-Tree Example (Insertion)

Insertion Sequence

- $S_1 = [a>0, b=2, e>4]$
- $S_2 = [c<3, f>1, m<2]$
  - $S_3 = [d=1, f<3]$
  - $S_4 = [a=3]$
- $S_5 = [d>3, f>2]$
- $S_6 = [d=4, f<4]$

l-node's max $\text{cap} = 3$

- $\{S_2\}$
- $\{S_3, S_5, S_6\}$
- $\{S_1, S_4\}$
BE-Tree Example (Insertion)

Insertion Sequence

$S_1 = [a > 0, b = 2, e > 4]$
$S_2 = [c < 3, f > 1, m < 2]$

$S_3 = [d = 1, f < 3]$
$S_4 = [a = 3]$
$S_5 = [d > 3, f > 2]$
$S_6 = [d = 4, f < 4]$
$S_7 = [a < 4, b = 1]$
$S_8 = [a < 3]$

l-node
{S_1, S_4, S_7, S_8}

Insertion Sequence
l-node's max cap = 3

BE-Tree
p-directory

a

l-node

{S_2}

Mohammad Sadoghi (University of Toronto)
BE-Tree Example (Insertion)

**Insertion Sequence**

\[ S_1 = [a > 0, b = 2, e > 4] \]

\[ S_2 = [c < 3, f > 1, m < 2] \]

\[ S_3 = [d = 1, f < 3] \]

\[ S_4 = [a = 3] \]

\[ S_5 = [d > 3, f > 2] \]

\[ S_6 = [d = 4, f < 4] \]

\[ S_7 = [a < 4, b = 1] \]

\[ S_8 = [a = 3] \]

**l-node's \( \text{max cap} = 3 \)**
BE-Tree Example (Insertion)

Insertion Sequence

\[ S_1 = [a>0, b=2, e>4] \]
\[ S_2 = [c<3, f>1, m<2] \]
\[ S_3 = [d=1, f<3] \]
\[ S_4 = [a=3] \]
\[ S_5 = [d>3, f>2] \]
\[ S_6 = [d=4, f<4] \]
\[ S_7 = [a<4, b=1] \]
\[ S_8 = [a<3] \]
\[ S_9 = [a>1] \]

l-node

l-node's \( \text{max cap} = 3 \)

Mohammad Sadoghi (University of Toronto)
BE-Tree Example (Insertion)

Insertion Sequence

\[ S_1 = [a>0, b=2, e>4] \]
\[ S_2 = [c<3, f>1, m<2] \]
\[ S_3 = [d=1, f<3] \]
\[ S_4 = [a=3] \]
\[ S_5 = [d>3, f>2] \]
\[ S_6 = [d=4, f<4] \]
\[ S_7 = [a<4, b=1] \]
\[ S_8 = [a<3] \]
\[ S_9 = [a>1] \]

**BE-Tree Structure**

- **p-directory**
  - \( S_1 = [a>0, b=2, e>4] \)
  - \( S_2 = [c<3, f>1, m<2] \)
  - \( S_3 = [d=1, f<3] \)
  - \( S_4 = [a=3] \)
  - \( S_5 = [d>3, f>2] \)
  - \( S_6 = [d=4, f<4] \)
  - \( S_7 = [a<4, b=1] \)
  - \( S_8 = [a<3] \)
  - \( S_9 = [a>1] \)

- **l-node**
  - \( \{ S_2 \} \)
  - \( \{ S_3, S_5, S_6 \} \)

- **l-node's max cap = 3**

- **c-node**
  - \( \{ S_8, S_9 \} \)
  - \( \{ S_4 \} \)
  - \( \{ S_1, S_7 \} \)
BE-Tree Example (the Notion of a Key)

**Insertion Sequence**

- $S_1 = [a>0, b=2, e>4]$
- $S_2 = [c<3, f>1, m<2]$
  - $S_3 = [d=1, f<3]$
  - $S_4 = [a=3]$
  - $S_5 = [d>3, f>2]$
  - $S_6 = [d=4, f<4]$
  - $S_7 = [a<4, b=1]$
  - $S_8 = [a<3]$
  - $S_9 = [a>1]$

**Example Structure**

- **c-node**
  - $[1, 4]$
  - $[2, 4]$
  - $[1, 4]$
- **l-node**
  - $\{S_2\}$
- **p-node**
  - $\{S_8, S_9\}$
  - $\{S_4\}$
- **l-node**
  - $\{S_3, S_5, S_6\}$

**l-node’s max cap = 3**
**BE-Tree Example (Matching)**

**Insertion Sequence**
- $S_1 = [a>0, b=2, e>4]$
- $S_2 = [c<3, f>1, m<2]$
- $S_3 = [d=1, f<3]$
- $S_4 = [a=3]$
- $S_5 = [d>3, f>2]$
- $S_6 = [d=4, f<4]$
- $S_7 = [a<4, b=1]$
- $S_8 = [a<3]$
- $S_9 = [a>1]$

**Event**
- $e_1 = [a=1, b=2, e=3]$
- $e_1 = [a=1]$

**Event**
- $e_1 = [a=1, b=2, e=3]$
- $e_1 = [a=1]$

**BE-Tree Diagram**

- **l-node**
  - $\{S_2\}$
- **c-node**
  - $a$
  - $d$
- **p-node**
  - $\{S_8, S_9\}$
  - $\{S_4\}$
  - $\{S_3, S_5, S_6\}$

**l-node's max cap = 3**
BE-Tree Example (Matching)

**Insertion Sequence**

- $S_1 = [a>0, b=2, e>4]$
- $S_2 = [c<3, f>1, m<2]$
  - $S_3 = [d=1, f<3]$
  - $S_4 = [a=3]$
  - $S_5 = [d>3, f>2]$
  - $S_6 = [d=4, f<4]$
  - $S_7 = [a<4, b=1]$
  - $S_8 = [a<3]$
  - $S_9 = [a>1]$

**Event**

- $e_1 = [a=1, b=2, e=3]$
- $e_1 = [a=1]$

**Event e_1 = [a=1]**

- $e_1 = [a=1]$

**Event e_1 = [a=1]**

- $e_1 = [a=1]$

**l-node's max cap = 3**

- l-node
  - {S_2}
  - {S_8, S_9}
  - {S_3, S_5, S_6}

**Insertion Sequence**

- $S_1 = [a>0, b=2, e>4]$
- $S_2 = [c<3, f>1, m<2]$
  - $S_3 = [d=1, f<3]$
  - $S_4 = [a=3]$
  - $S_5 = [d>3, f>2]$
  - $S_6 = [d=4, f<4]$
  - $S_7 = [a<4, b=1]$
  - $S_8 = [a<3]$
  - $S_9 = [a>1]$

- $e_1 = [a=1, b=2, e=3]$
- $e_1 = [a=1]$

**l-node's max cap = 3**

- l-node
  - {S_2}
  - {S_8, S_9}
  - {S_3, S_5, S_6}

**l-node's max cap = 3**

- l-node
  - {S_2}
  - {S_8, S_9}
  - {S_3, S_5, S_6}
1. Application Scenarios

2. Matching Problem

3. BE-Tree (Boolean Expression-Tree)

4. BE-Tree Adaptation Policies

5. Experimental Evaluation

6. Conclusions
Experimental Evaluation

Algorithms

1. **BE**: *BE-Tree* (our index structure)
2. **BE-B**: *BE-Tree* (batch version)
3. **GR**: IBM *Gryphon* (Aguilera et al., PODC’99)
4. **P**: *Propagation* Algorithm (Fabret et al. SIGMOD’01)
5. **k-ind**: *k-index* (Whang et al. VLDB’09)
6. **SIFT**: *Counting* Algorithm (Yan et al. TODS’94)
7. **SCAN**: *Sequential Scan*
### Workload Configurations

**Table:** Synthetic and Real Workload Properties

<table>
<thead>
<tr>
<th>Workload Size</th>
<th>Number of Dimensions</th>
<th>Dimension Cardinality</th>
<th>Predicate Selectivity</th>
<th>Dimension Selectivity</th>
<th>Sub/Event Size</th>
<th>% Equality Pred</th>
<th>Match Prob</th>
<th>DBLP (Author)</th>
<th>DBLP (Title)</th>
<th>Match Prob (Author)</th>
<th>Match Prob (Title)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Size</td>
<td>100K-1M</td>
<td>1M</td>
<td>100K</td>
<td>100K</td>
<td>100K</td>
<td>1M</td>
<td>1M</td>
<td>100-760K</td>
<td>50-250K</td>
<td>400k</td>
<td>150</td>
</tr>
<tr>
<td>Number of Dim</td>
<td>400</td>
<td>50-1400</td>
<td>400</td>
<td>400</td>
<td>400</td>
<td>400</td>
<td>400</td>
<td>677</td>
<td>677</td>
<td>677</td>
<td>677</td>
</tr>
<tr>
<td>Cardinality</td>
<td>48</td>
<td>48-150K</td>
<td>48</td>
<td>2-10</td>
<td>48</td>
<td>48</td>
<td>48</td>
<td>26</td>
<td>26</td>
<td>26</td>
<td>26</td>
</tr>
<tr>
<td>Avg. Sub Size</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>5-66</td>
<td>7</td>
<td>7</td>
<td>8</td>
<td>35</td>
<td>8</td>
<td>30</td>
</tr>
<tr>
<td>Avg. Event Size</td>
<td>15</td>
<td>15</td>
<td>15</td>
<td>15</td>
<td>13-81</td>
<td>15</td>
<td>15</td>
<td>8</td>
<td>35</td>
<td>16</td>
<td>43</td>
</tr>
<tr>
<td>Pred Avg. Range Size %</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>6-50</td>
<td>—</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>—</td>
<td>—</td>
<td>12</td>
</tr>
<tr>
<td>% Equality Pred</td>
<td>0.3</td>
<td>0.3</td>
<td>0.3</td>
<td>0.3</td>
<td>1.0</td>
<td>0.3</td>
<td>0.3</td>
<td>1.0</td>
<td>1.0</td>
<td>0.3</td>
<td>0.3</td>
</tr>
<tr>
<td>Match Prob %</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0.01-9</td>
<td>—</td>
<td>—</td>
<td>0.01-9</td>
</tr>
</tbody>
</table>
Workload Configurations

Table: Synthetic and Real Workload Properties

<table>
<thead>
<tr>
<th></th>
<th>Size</th>
<th>Number of Dimensions</th>
<th>Dimension Cardinality</th>
<th>Predicate Selectivity</th>
<th>Dimension Selectivity</th>
<th>Sub/Event Size</th>
<th>% Equality Pred</th>
<th>Match Prob %</th>
<th>DBLP (Author)</th>
<th>DBLP (Title)</th>
<th>Match Prob (Author)</th>
<th>Match Prob (Title)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Workload Size</td>
<td>100K-1M</td>
<td>1M</td>
<td>100K</td>
<td>100K</td>
<td>100K</td>
<td>1M</td>
<td>1M</td>
<td>1M</td>
<td>100-760K</td>
<td>50-250K</td>
<td>400k</td>
<td>150</td>
</tr>
<tr>
<td>Number of Dim</td>
<td>400</td>
<td>50-1400</td>
<td>400</td>
<td>400</td>
<td>400</td>
<td>400</td>
<td>400</td>
<td>400</td>
<td>677</td>
<td>677</td>
<td>677</td>
<td>677</td>
</tr>
<tr>
<td>Cardinality</td>
<td>48</td>
<td>48-150K</td>
<td>48</td>
<td>2-10</td>
<td>2-10</td>
<td>2-10</td>
<td>2-10</td>
<td>2-10</td>
<td>26</td>
<td>26</td>
<td>26</td>
<td>26</td>
</tr>
<tr>
<td>Avg. Sub Size</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>5-66</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>8</td>
<td>35</td>
<td>8</td>
<td>30</td>
</tr>
<tr>
<td>Avg. Event Size</td>
<td>15</td>
<td>15</td>
<td>15</td>
<td>15</td>
<td>13-81</td>
<td>15</td>
<td>15</td>
<td>15</td>
<td>8</td>
<td>35</td>
<td>16</td>
<td>43</td>
</tr>
<tr>
<td>Pred Avg. Range Size %</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>6-50</td>
<td>—</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>—</td>
<td>—</td>
<td>12</td>
<td>12</td>
</tr>
<tr>
<td>% Equality Pred</td>
<td>0.3</td>
<td>0.3</td>
<td>0.3</td>
<td>0.3</td>
<td>0.3</td>
<td>0.3</td>
<td>0.3</td>
<td>0.3</td>
<td>1.0</td>
<td>1.0</td>
<td>0.3</td>
<td>0.3</td>
</tr>
<tr>
<td>Match Prob %</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0.01-9</td>
<td>1</td>
<td>0.01-9</td>
<td>—</td>
<td>—</td>
<td>0.01-9</td>
<td>0.01-9</td>
</tr>
</tbody>
</table>

The results in this paper were verified by the SIGMOD repeatability committee.
Effect of Workload Size on Matching (Log Scale)

Figure: Varying Workload Size

(a) Uniform: Workload Size

(b) Zipf: Workload Size
Effect of Dimension Cardinality on Matching (Log Scale)

Figure: Varying Dimension Cardinality

(a) Uniform: Dimension Cardinality
(b) Zipf: Dimension Cardinality
Effect of Predicate Expressiveness on Matching (Log Scale)

Figure: Varying % of Equality Predicates
1. Application Scenarios

2. Matching Problem

3. BE-Tree (Boolean Expression-Tree)

4. BE-Tree Adaptation Policies

5. Experimental Evaluation

6. Conclusions
The main benefits in the overall design of BE-Tree are to
Conclusions

The main benefits in the overall design of BE-Tree are to

1. support expressive set of predicate operators (e.g., range predicates) through a novel predicate transformation
The main benefits in the overall design of BE-Tree are to

1. support expressive set of predicate operators (e.g., range predicates) through a novel predicate transformation

2. handle subscriptions that impose conditions only on a small set of dimensions (of a high-dimensional space), resulting in a high degree of overlap, through the two-phase space-cutting technique that identifies dense subspaces and copes with the curse of dimensionality
Conclusions

The main benefits in the overall design of \textit{BE-Tree} are to

1. support expressive set of predicate operators (e.g., range predicates) through a novel predicate transformation

2. handle subscriptions that impose conditions only on a small set of dimensions (of a high-dimensional space), resulting in a high degree of overlap, through the two-phase space-cutting technique that identifies dense subspaces and copes with the curse of dimensionality

3. adapt to temporal changes of the workload distribution and to sustain high matching rates in presence of frequent subscription updates through
   1. a cascading-split-free approach when inserting and removing expressions
   2. a deterministic clustering and a partition-clustering alteration strategy that are independent of the insertion sequence
   3. a novel cost-based technique to optimize the matching cost and recycle ineffective buckets
Thank You,
Effect of Sub/Event Size on Matching (Log Scale)

Varying Sub/Event Size; Sub=100K
(a) Uniform: Sub/Event Size
(b) Zipf: Sub/Event Size

Figure: Varying Subscription/Event Size
Effect of Matching Probability on Matching (Log Scale)

Varying Match (%); Sub=1M
(a) Uniform: % Matching Prob
(b) Zipf: % Matching Prob

Figure: Varying % of Matching Probability