Relevance Matters: Capitalizing on Less

[Top-k Matching in Publish/Subscribe]

Mohammad Sadoghi           Hans-Arno Jacobsen

University of Toronto

April 3, 2012
1 Application Scenarios

2 Matching Problem

3 BE*-Tree (Boolean Expression-Tree)

4 Experimental Evaluation

5 Conclusions

6 BE*-Tree Adaptation Policy
Computational Advertising (A Billion-dollar Industry)
Computational Advertising (A Billion-dollar Industry)

Advertisement (BE):
(age < 32) \text{wt}=0.2
(credit-score > 630) \text{wt}=0.6
(num-visits > 4) \text{wt}=0.1
(price = 150) \text{wt}=0.1
Computational Advertising (A Billion-dollar Industry)

Advertisement (BE):
(age < 32) \( \text{wt}=0.2 \)
(credit-score > 630) \( \text{wt}=0.6 \)
(num-visits > 4) \( \text{wt}=0.1 \)
(price = 150) \( \text{wt}=0.1 \)

Subscriptions

User Profiles

Events

Events

Online User

Broker

User Profiles

Advertiser

Advertising Campaign

Advertiser

Sony

Amazon

Sears

Sony

Amazon

BE*-Tree

ICDE'12 4 / 25
Computational Advertising (A Billion-dollar Industry)

Advertisement (BE):
(age < 32)$_{wt=0.2}$
(credit-score > 630)$_{wt=0.6}$
(num-visits > 4)$_{wt=0.1}$
(price = 150)$_{wt=0.1}$

Events
(Num-visits=13)$_{wt=0.5}$
(age=25)$_{wt=0.1}$
(credit-score=647)$_{wt=0.2}$
(price<235)$_{wt=0.5}$

User Profiles

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

Online User

Broker

Advertise
Advertisement Campaign

(Relevant)

Adverser

Sears
Sony
Amazon
Computational Advertising (A Billion-dollar Industry)

Advertisement (BE):
- (age < 32) \( w_t = 0.2 \)
- (credit-score > 630) \( w_t = 0.6 \)
- (num-visits > 4) \( w_t = 0.1 \)
- (price = 150) \( w_t = 0.1 \)

Advertiser

Advertising Campaign

Subscriptions

User Profiles

Broker

Clickstream

Online User

Events

(Most Relevant)

Sears

Sony

Amazon

(Most Relevant)

Sony

Amazon

"BMW X3 2008" (price < 235) \( w_t = 0.2 \)

"BMW X3 2008" (price < 235) \( w_t = 0.2 \)
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)

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

1. Retrieve only the most relevant subscriptions for given a event.
2. Handle subscriptions with expressive operators (over continuous domains) that impose conditions only on a few dimensions, resulting 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 skewed workload distributions (self-adjusting mechanism), i.e., avoid structure deterioration.
1. Application Scenarios

2. Matching Problem

3. BE*-Tree (Boolean Expression-Tree)

4. Experimental Evaluation

5. Conclusions

6. BE*-Tree Adaptation Policy
Language and Data Model

- Both subscription and event are defined as Boolean expressions which are set of predicates.
Both subscription and event are defined as Boolean expressions which are set of predicates.

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

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

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


Language and Data Model

- Both subscription and event are defined as Boolean expressions which are set of predicates.
- A predicate $P$ is a quadruple consisting of an attribute uniquely representing a dimension in $n$-dimensional space, an operator, a range of values, and an assigned predicate weight denoted by $P(\text{attr}, \text{opt}, \text{val}, \text{wt})$.
- A predicate $P(x)$ either accepts or rejects an input $x$ such that $P : x \rightarrow \{\text{True}, \text{False}\}$, where $x \in \text{Dom}(P_{\text{attr}})$.
- Each predicate supports relational operators ($<$, $\leq$, $=$, $\neq$, $\geq$, $>$), set operators ($\in$, $\notin$), or the SQL BETWEEN operator.
Language and Data Model

- Both subscription and event are defined as Boolean expressions which are set of predicates.
- A predicate $P$ is a quadruple consisting of an attribute uniquely representing a dimension in $n$-dimensional space, an operator, a range of values, and an assigned predicate weight denoted by $P^{\text{attr}, \text{opt}, \text{val}, \text{wt}}$.
- A predicate $P(x)$ either accepts or rejects an input $x$ such that $P : x \rightarrow \{\text{True}, \text{False}\}$, where $x \in \text{Dom}(P^{\text{attr}})$.
- Each predicate supports relational operators ($<, \leq, =, \neq, \geq, >$), set operators ($\in, \notin$), or the SQL BETWEEN operator.
- Formally, a Boolean expression $\Omega$ is defined over an $n$-dimensional space as follows:

**Definition**

\[
\Omega = \{ P_1^{\text{attr}, \text{opt}, \text{val}, \text{wt}}(x) \land \ldots \land P_k^{\text{attr}, \text{opt}, \text{val}, \text{wt}}(x) \},
\]

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 (All Matches)

Stabbing Subscription

*Given an event* \( \omega \) *and a set of subscriptions* \( \Omega \), *find all subscriptions* \( \Omega_i \in \Omega \) *that are satisfied by* \( \omega \).

Definition

\[
\text{SQ}(\omega) = \{ \Omega_i \mid \forall P^\text{attr, opt, val, wt}_q(x) \in \Omega_i, \exists P^\text{attr, opt, val, wt}_o(x) \in \omega, P^\text{attr}_q = P^\text{attr}_o, \exists x \in \text{Dom}(P^\text{attr}_q), P_q(x) \land P_o(x) \}\.
\]
Top-k Matching Semantics (Relevant Matches)

Information Retrieval Vector Space (Monotonic) Scoring Function

We compute the score of a matched subscription $\Omega_i$ for a given event $\omega$ as follows:

$$\text{score}(\omega, \Omega_i) = \sum_{P_q(x) \in \Omega_i, P_o(x) \in \omega, P_q^\text{attr} = P_o^\text{attr}} P_q^\text{wt} \times P_o^\text{wt}.$$

Upper Bound Scoring Computation

We compute the upper bound score for an event $\omega$ w.r.t. an upper bound weighted summary $(\text{sum}_\text{wt})$ for a set of subscriptions $\Omega$ as follows:

$$\text{uscore}(\omega, \text{sum}_\text{wt}) = \sum_{P_o(x) \in \omega} P_o^\text{wt} \times \text{sum}_\text{wt}(P_o^\text{attr}),$$

where $\text{sum}_\text{wt}(\text{attr})$ returns the upper bound score of $\text{attr}$ over subscriptions $\Omega$ given by

$$\text{sum}_\text{wt}(\text{attr}) = \max_{\Omega_i \in \Omega, P_q(x) \in \Omega_i, P_q^\text{attr} = \text{attr}} P_q^\text{wt}.$$
1 Application Scenarios

2 Matching Problem

3 BE*-Tree (Boolean Expression-Tree)

4 Experimental Evaluation

5 Conclusions

6 BE*-Tree Adaptation Policy
To systematically explore the space in two iterative phases of partitioning and non-rigid clustering (i) to cope with the curse of dimensionality (ii) to support dynamic insertion and removal of subscriptions.
BE*-Tree Core Design (Partitioning – Global structuring)
BE*-Tree Core Design (Clustering – Local Structuring)
BE*-Tree Novel Features (Overlap-free Splitting Strategy)
BE*-Tree Novel Features (Overlap-free Splitting Strategy)
BE*-Tree Novel Features (Overlap-free Splitting Strategy)
BE*-Tree Novel Features (Overlap-free Splitting Strategy)
BE*-Tree Novel Features (Overlap-free Splitting Strategy)
BE*-Tree Novel Features (Overlap-free Splitting Strategy)

Reduce Overlap (R*-Tree)
Reduce Coverage (R-Tree)
BE*-Tree Novel Features (Overlap-free Splitting Strategy)

Left Buck

Reduce Overlap (R*-Tree)

Reduce Coverage (R-Tree)
BE*-Tree Novel Features (Overlap-free Splitting Strategy)

Left Buck

Right Buck

Reduce Overlap (R*-Tree)

Reduce Coverage (R-Tree)
BE*-Tree Novel Features (Overlap-free Splitting Strategy)

- **Left Buck**
- **Right Buck**
- **Overlap Buck**

Reduce Overlap (R*-Tree)
Reduce Coverage (R-Tree)
BE*-Tree Novel Features (Overlap-free Splitting Strategy)

Reduce Coverage (R-Tree)
Reduce Overlap (R*-Tree)
Descendant Repelling
BE*-Tree Novel Features (Overlap-free Splitting Strategy)
BE*-Tree Novel Features (Bi-directional Tree Expansion)

Insertion Sequence → Buck
BE*-Tree Novel Features (Bi-directional Tree Expansion)

Insertion Sequence

$t_1$

Buck
BE*-Tree Novel Features (Bi-directional Tree Expansion)
BE*-Tree Novel Features (Bi-directional Tree Expansion)

**Insertion Sequence**

- $t_1$
- $t_2$
- $t_3$

**Buck**
BE*-Tree Novel Features (Bi-directional Tree Expansion)
BE*-Tree Novel Features (Bi-directional Tree Expansion)

**Insertion Sequence**
- $t_1$
- $t_2$
- $t_3$

**Top-down Expansion**
- $t_1$
- $t_2$
- $t_3$
BE*-Tree Novel Features (Bi-directional Tree Expansion)
BE*-Tree Novel Features (Bi-directional Tree Expansion)

- **Insertion Sequence**
  - **Top-down Expansion**
  - **Bottom-up Expansion**

- **Dead Space**
- **Increased False Candidates**

Mohammad Sadoghi (University of Toronto)
BE*-Tree Novel Features (Bi-directional Tree Expansion)
BE*-Tree Novel Features (Bi-directional Tree Expansion)

- **Insertion**
  - Sequence
  - Buck
- **Top-down Expansion**
- **Bottom-up Expansion**
- **Increased False Candidates**
  - Dead Space

**Application Scenarios Language & Semantics Intuitions Evaluation Conclusions Adaptation**
BE*-Tree Novel Features (Hierarchical Top-k Matching)

Continuously refining upper bound score during BE*-Tree traversal.
BE*-Tree Novel Features (Hierarchical Top-k Matching)

Continuously refining upper bound score during BE*-Tree traversal.
BE*-Tree Novel Features (Top-k Pruning)
(1) **p-node pruning**: follow partitions in order of the highest upper bound score that are greater than the score of the least relevant subscription (realized through *p-node-max-heap*)
(2) l-node pruning: examine only leaf nodes with the upper bound scores greater than the score of the least relevant subscription (realized through $k$-min-heap)
1 Application Scenarios

2 Matching Problem

3 BE*-Tree (Boolean Expression-Tree)

4 Experimental Evaluation

5 Conclusions

6 BE*-Tree Adaptation Policy
Experimental Evaluation

Algorithms

1. **BE*-B**: *BE*-Tree (with Batching)
2. **BE***: *BE*-Tree
3. **BE**: *BE*-Tree (Sadoghi et al., SIGMOD’11)
4. **k-ind**: *k*-index (Whang et al. VLDB’09)
5. **SCAN**: Sequential Scan
# Workload Configurations

<table>
<thead>
<tr>
<th>Workload Size</th>
<th>Number of Dimensions</th>
<th>Dimension Cardinality</th>
<th>Number of Clusters</th>
<th>Cluster Size</th>
<th>Sub/Event Size</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>100K-1M</td>
<td>1M</td>
<td>100K</td>
<td>100K</td>
<td>100K</td>
<td>100K</td>
<td>1M</td>
<td>100-760K</td>
<td>50-250K</td>
<td>400k</td>
<td>150</td>
</tr>
<tr>
<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>250M</td>
<td>250M</td>
<td>250M</td>
<td>250M</td>
<td>250M</td>
<td>250M</td>
<td>250M</td>
<td>26^3</td>
<td>26^3</td>
<td>26^3</td>
<td>26^3</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>4</td>
<td>2-26,NA</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>—</td>
<td>—</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>25-800</td>
<td>100</td>
<td>—</td>
<td>—</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>5-66</td>
<td>8</td>
<td>35</td>
<td>8</td>
<td>35</td>
<td>8</td>
</tr>
<tr>
<td>15</td>
<td>15</td>
<td>15</td>
<td>15</td>
<td>15</td>
<td>13-81</td>
<td>15</td>
<td>—</td>
<td>—</td>
<td>16</td>
<td>43</td>
</tr>
<tr>
<td>0.1</td>
<td>0.1</td>
<td>0.1</td>
<td>0.1</td>
<td>0.1</td>
<td>0.1</td>
<td>0.1</td>
<td>0.01-9</td>
<td>—</td>
<td>—</td>
<td>0.01-9</td>
</tr>
</tbody>
</table>
Effect of Workload Size on Matching (Log Scale)

**Figure**: Varying Workload Size (Match Prob = 0.1%)
Effect of Matching Probability on Top-k Matching

Figure: Varying % of Matching Probability Predicates
1. Application Scenarios

2. Matching Problem

3. BE*-Tree (Boolean Expression-Tree)

4. Experimental Evaluation

5. Conclusions

6. BE*-Tree Adaptation Policy
Conclusions

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

1. support expressive set of predicate operators (e.g., range predicates) over continuous domain

2. handle subscriptions that impose conditions only on a small set of dimensions, resulting in a high degree of overlap

3. adapt to skew distribution and to sustain high matching rates in presence of frequent subscription updates through
   - a non-rigid clustering through bi-directional tree expansion and descendant-repelling and overlap-free splitting strategy (reduced BE-Tree matching time by up to 59%)

4. retrieve most relevant matches through a hierarchical top-k structure (reduced matching time by up to 77%)
Thank You,
Intuition Behind the Two-phase Space-cutting Technique

Required rules to avoid the **cascading split problem** and to ensure a robust strategy to alter between partitioning and clustering.

1. **Insertion rule:** expression is always inserted into the smallest bucket that encloses it, while respecting the descendant-repelling property.
2. **Forced split rule:** an overflowing non-leaf bucket is always split before switching back to the partitioning.
3. **Merge rule:** underflowing leaf bucket is merged with its parent only if the parent bucket is not partitioned yet.

**Invariance**

*Every expression $\Omega$ always inserted into the smallest bucket that encloses it, and a leaf bucket is always split first before it is partitioned.*
1 Application Scenarios

2 Matching Problem

3 BE*-Tree (Boolean Expression-Tree)

4 Experimental Evaluation

5 Conclusions

6 BE*-Tree Adaptation Policy
BE*-Tree Self-adjustment Mechanism

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).

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. **maximizing 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

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

(1)

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

(2)

\[
\text{Gain}(l_j) = \# \text{ subsumed pred}
\]  

(3)
BE*-Tree Example (Insertion)

**Insertion Sequence**

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

The c-node is the root node. The l-node's max capacity is 3.

**Example Insertion**

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

The sequence is inserted into the tree, maintaining the constraints.
BE*-Tree Example (Insertion)

**Insertion Sequence**

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

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

**c-node**

\{S_1, S_2, S_3, S_4\}
BE*-Tree Example (Insertion)

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

- **c-directory**
  - $[1, 4]$

- **c-node**
  - $a$

- **p-node**
  - $\{S_2, S_3\}$

- **l-node**
  - $\{S_1, S_4\}$

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

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

Insertion Sequence

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

Insertion Sequence
\[ \{ S_2, S_3, S_5, S_6 \} \]

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

**Insertion Sequence**

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

**c-node**
- \{S_1, S_4\}
- \{S_3, S_5, S_6\}

**l-node**
- \{S_2\}

**p-node**
- \{S_2\}

**c-directory**
- \[1, 4\]

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

**Insertion Sequence**

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

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

- \( \{ S_3, S_5, S_6 \} \)
- \( \{ S_1, S_4, S_7, S_8 \} \)
BE*-Tree Example (Insertion)

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

**l-node**

\{ S_8, S_9 \}

**c-node**

\{ S_1, S_7, S_8, S_9 \}

**l-node**

\{ S_3, S_5, S_6 \}
BE*-Tree Example (Insertion)

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

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

$\text{Insertion Sequence}\{S_2\}$
BE*-Tree Example (Matching)

Insertion Sequence

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

- **c-node**: \([1, 4]\)
- **p-node**: \([2, 4]\)
- **c-directory**: \([1, 4]\)
- **l-node's max cap**: 3

**BE*-Tree Example (Matching)**
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

- \( e_1 = [a=1, b=2, e=3] \)
- \( e_1 = [a=1] \)
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**

- \( e_1 = [a=1, b=2, e=3] \)
- \( e_1 = [a=1] \)

**l-node**

- \( l-node \)
  - \( l-node \)
  - \( l-node \)
  - \( l-node \)
  - \( l-node \)
  - \( l-node \)
  - \( l-node \)

**c-node**

- \( c-node \)
  - \( c-node \)
  - \( c-node \)
  - \( c-node \)
  - \( c-node \)
  - \( c-node \)
  - \( c-node \)

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

- \( l-node \)
  - \( l-node \)
  - \( l-node \)
  - \( l-node \)
  - \( l-node \)
  - \( l-node \)
  - \( l-node \)

- \( l-node \)
  - \( l-node \)
  - \( l-node \)
  - \( l-node \)
  - \( l-node \)
  - \( l-node \)
  - \( l-node \)

**p-node**

- \( p-node \)
  - \( p-node \)
  - \( p-node \)
  - \( p-node \)
  - \( p-node \)
  - \( p-node \)
  - \( p-node \)

**p-directory**

- \( p-directory \)
  - \( p-directory \)
  - \( p-directory \)
  - \( p-directory \)

**c-directory**

- \( c-directory \)
  - \( c-directory \)
  - \( c-directory \)
  - \( c-directory \)

**S**

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

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

- \( S_8 = [a<3] \)
  - \( S_9 = [a>1] \)

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

- \( S_8 = [a<3] \)
  - \( S_9 = [a>1] \)

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

- \( S_8 = [a<3] \)
  - \( S_9 = [a>1] \)
Effect of Workload Size on Top-k Matching (Log Scale)

Figure: Varying Workload Size (Match Prob = 0.1%)