Structuring the network: ordering nodes

Gossip-based distributed slicing
[JK,P2P 2006] [FGJKR,ICDCS 2007]
Gossip-based ordered slicing

- Several applications running on the same infrastructure
- Need for network partitioning
  - Decentralized
  - Robust
  - Handle dynamics
  - Customizable

Structure: slices
Gossip-based approach to “slice” the network
Objective

- Targeted applications
  - Desktop grids
  - Testbed platforms (PlanetLab)
  - Telco applications
- Resource assignment
- Objective
  - Create and maintain partitions (slices) as subsets of the network in a fully decentralized manner
  - Ordered nature: along a single attribute (memory, bandwidth, computing power)
- Use a gossip-based approach to estimate to which partition a node belongs
  - Scalable
  - Robust
  - Based on local knowledge
Random slices
Random slices

1 0.21

8 0.67

2 0.11

9 0.43

3 0.52

10 0.98

4 0.22

6 0.55

5 0.87

7 0.09
Random slices
Random slices

1 0.21
8 0.67
7 0.09
9 0.43
2 0.11
6 0.55
10 0.98
4 0.22
5 0.87
3 0.52
Ordered slices: System model

- Problem
  - Automatically assign a node to a slice
  - Along an attribute metric

- System model
  - Crash-only nodes
  - Each node $i$ has
    - an attribute $\chi_i$
    - a random number $r_i$
    - a view of $c$ entries (peer sampling)
    - a time stamp
  - Each node belongs to one slice
Ordered slicing algorithm: basic operation

<table>
<thead>
<tr>
<th></th>
<th>Node a</th>
<th></th>
<th>Node b</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_i$</td>
<td>12 34 22 35 56 78 98 2 37 13</td>
<td></td>
<td>12 34 22 35 56 78 98 2 37 13</td>
<td></td>
</tr>
<tr>
<td>$r_i$</td>
<td>0.6 0.4 0.3 0.45 0.87 0.77 0.21 0.65 0.98 0.12</td>
<td></td>
<td>0.6 0.4 0.65 0.45 0.87 0.77 0.21 0.3 0.98 0.12</td>
<td></td>
</tr>
</tbody>
</table>

Table 1: Ordered slicing algorithm

<table>
<thead>
<tr>
<th></th>
<th>Node a</th>
<th></th>
<th>Node b</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_i$</td>
<td>2 12 13 22 34 35 37 56 78 98</td>
<td></td>
<td>2 12 13 22 34 35 37 56 78 98</td>
<td></td>
</tr>
<tr>
<td>$r_i$</td>
<td>0.12 0.21 0.3 0.4 0.45 0.6 0.65 0.77 0.87 0.98</td>
<td></td>
<td>0.12 0.21 0.3 0.4 0.45 0.6 0.65 0.77 0.87 0.98</td>
<td></td>
</tr>
</tbody>
</table>
Ordered slicing algorithm

Wait (t)

\( p \gets \text{random element from view} \)

\( \text{buffer} \gets \text{view} \cup \{(\text{my}@, \text{my\_stamp}, x_q, r_q)\} \)

Send buffer to \( p \)

Receive buffer(\( p \))

\( \text{view} \gets \text{freshest} \ c \text{ entries from} \)

\( \{\text{buffer}(p) \cup \text{view}\} \)

\( i \gets \text{peer such that} \ (x_i - x_q)(r_i - r_q) < 0 \)

Send \( (x_q, r_q) \) to \( i \)

\( r_q = r_i \)

Active thread at peer \( q \)

Receive buffer(\( q \)) from \( q \)

\( \text{buffer} \gets \text{view} \cup \{(\text{my}@, \text{my\_stamp}, x_p, r_p)\} \)

Send buffer to \( q \)

\( \text{view} \gets \text{freshest} \ c \text{ entries from} \)

\( \{\text{buffer}(p) \cup \text{view}\} \)

Passive thread at peer \( p \)
Ordered slicing algorithm: maintenance

- New nodes discovered using the random peer sampling
- Random number ensures uniform spread
- Once the order stabilizes: each node knows which slice it belongs to
- Example
  - A peer with a number <0.5 knows in the first 50% of the nodes according to the metric
- Slice creation and maintenance
Disorder measure

\[ \sigma(t, r_{i_1(t)}, \ldots, r_{i_N(t)}) = \sigma(t) = 1/N \sum_{j=1}^{N} (j - i_j(t))^2 \]
Analogy with average

- Weight conserving property

\[
\frac{1}{N} \sum_{j=1}^{N} j - i_j(t) = \frac{1}{N} \sum_{j=1}^{N} j - 1/N \sum_{j=1}^{N} i_j(t) = 0
\]

- The swapping does not influence this value (=0) but always reduces the disorder value
Exponential decrease of the disorder
Swaps over time
Simulation set-up

- PeerSim simulator
- Configurations
  - Network size: 30,000 – 100,000 – 300,000 node overlay
  - Views: c=20, 40 and 80
  - Unreliable communication channels
  - Churn
  - Age bias: peer selection based on age similarity
- Metric: disorder
Churn

- 0.1% churn in each cycle
- 1% churn in each cycle

Graphs showing disorder (\(\sigma\)) over cycles for different values of \(N\) and \(c\).
Age-based technique

Young nodes disordered
Old nodes protected
Example

- Quick stabilization
- Relatively well-defined slices
  - constant churn
  - massive failure
  - massive join
- Stabilizes as soon as churn stops
Ordered slicing

- Issue when failures are correlated to the attribute values
- Fix the uniformity requirement
- [Fernandez & al, ICDCS 2007]
  - Infer slice from a sample of attributes
  - Gossip-based propagation
Gossip-based aggregation

[Jelasity et al, ACM TOCS 2005]
Aggregation in large networks

- Basic building block for maintenance and monitoring
- Aggregation: global system property
  - System size estimation
  - Total free storage
  - Maximum load
  - Average uptime
  - Hotspots...
Aggregation protocol

- Compute global property from local interactions
- Proactive
- Adaptive: accounts for dynamics
System model

- N nodes, assigned unique Ids
- Communicate through message exchanges
- Each node has a partial view
- Each node holds a numeric value
- Push-pull communication
Basic aggregation protocol

- Passive and active threads: symmetric communication
- Local state: Current estimate of global aggregate
- `Getneighbor()`: returns a uniform random sample
- `Update()`: compute a new local state based on the current local state and the received one
- Example: average
  - `Update(s_p, s_q)` returns `(s_p+s_q)/2`
  - Mass conservation: Simple redistribution of the initial sum among the two nodes
  - Decreases the variance over the set of all estimates
  - Exponential convergence
Gossip framework instantiation

- Style of interaction: push-pull
- Local state $S$: Current estimate of global aggregate
- Method `SelectPeer()`: Single random neighbor
- Method `Update()`: Numerical function defined according to desired global aggregate (arithmetic/geometric mean, min, max, etc.)
Peer selection

- **Perfect matching:**
  - Optimal strategy
  - Cannot be implemented without global knowledge
  - Intuition:
    - N/2 pairs created without overlap,
    - Iterate on the perfect matching
    - Utility of exchanges maximized

- **Random matching:**
  - Intuition:
    - For all nodes, the same sampling probability applies at each time step
    - Each pair has the same probability of being selected regardless of having been selected before
    - Much slower convergence
Peer selection

- **Gossip-based matching**
  - In each execution of average: each node is guaranteed to be a member of at least one pair.
  - Distribution improves upon fully random
  - Better convergence than random although not optimal

\[
\rho = \frac{E(\sigma^2_{i+1})}{E(\sigma^2_i)} \approx \frac{1}{2 \sqrt{e}} \approx 0.303
\]
Practical protocol

- Adaptive
  - Restart the protocol every epoch
  - Messages tag with epoch identifier

- Synchronization
  - When a node receives a message tagged with epoch $j > i$, stops computation of $i$ and starts participating to $j$

- Multiple instances
Other functions

- Minimum and maximum
- Generalized means
- Variance
- Counting
- Sum and products
Peer counting

- Observation: one node has value 1, all others have value 0, global average is 1/N
- No need for leader election
- Several concurrent instances (leader Id)
  - Each node maintains a map M associating a leader and the average
Peer counting (2)

- Nodes $n_i$ and $n_j$ perform an exchange
- The new map $M$ merges $M_i$ and $M_j$

$$M = \{(l, e/2 \mid e = M_i(\ell) \land \ell \not\in D(M_j)) \cup \{(l, e/2 \mid e = M_j(\ell) \land \ell \not\in D(M_i)) \cup \{(l, (e_i + e_j)/2 \mid e_i = M_i(\ell) \land e_j = M_j(\ell))\}$$

$D(M) =$ domain set of $M$
$e_i:$ current estimate of node $n_i$

If the average estimate for a leader is known to only one node, the other node is considered to have an estimate of 0.
Robustness of network size estimation

1000 nodes crash at the beginning of each cycle
Robustness of network size estimation

20% of messages are lost
Structuring overlays with gossip

T-Man [Jelasity et al, Computer networks 2009]
System model

- $N$ nodes connected
- Node descriptor
  - IP@
  - Profile: additional information relevant for the overlay (e.g. ID, geographical location, available resources, etc)
- Highly dynamic network: crashes and leaves treated uniformly
- Communication: unpredictable delays
- All nodes have access to a peer sampling service
Problem statement

- Building desirable overlay form scratch
- Overlay representation: ranking function
  - N nodes
  - Target view size of size K
  - Ranking function
    - Node x, \{y_1, \ldots, y_j\} ≤ N
    - Outputs an ordered list of the j nodes
Example

- K=2, node profile is a real number in [0,M]
- Ranking d(a,b) = |a-b|
- Output: circular structure
T-man protocol

- Gossip-based protocol: periodic exchange of node descriptors
- Constant improvement of partial views
- Local state S: current neighbor set

Parameters
- View propagation: push-pull
- Peer selection: single random neighbor
- View selection: ranking function defined according to desired topology
T-man protocol

(Active thread)

Loop:

Wait(Δ)
P selectPeer(Ψ, rank (MyDescriptor, view))
buffer rank(p, buffer)
Send first m entries of buffer to p
Receive buffer_p from p
View merge(buffer_p, view)

(Passive thread)

Loop:

Receive buffer_q from q
Buffer merge(view, {myDescriptor})
Buffer rank(q, buffer)
Send first m entries of buffer to q
View merge(buffer_q, view)

Rank(); ranking method
Δ: cycle length
Ψ: peer sampling parameter
m: message size

November 2012
T-man: torus

Cycle 3  Cycle 5  Cycle 8  Cycle 15
Structuring the network

- Peers optimize their view using the view of their close neighbours
- Ranking function

\[ R(x, \{y_1, \ldots, y_m\}) \text{ ranks } y_j \text{ strictly lower than } y_i \text{ if } y_i \text{ precedes strictly } y_j \text{ in all possible rankings} \]

- Peer selection
  - Rank nodes in the view according to \( R \)
  - Returns a random sample from the first half
- Data exchange
  - Rank the elements in the (view+buffer) according to \( R \)
  - Returns the first \( c \) elements
- Data processing
  - Keep the \( c \) closest
Gossip-based topology management

- **Line**: $d(a,b) = |a-b|$
- **Ring**: interval $[0,N]$, $d(a,b) = \min(N-|a-b|,|a-b|)$
- **Mesh and torus**: $d=$Manhattan distance
- **Sorting problems**: any other application dependent metric
T-man properties

- Generate a large number of structured topologies
- Exponential convergence (logarithmic in the number of nodes)
- Irrespective of the initial topology
- Exact structure
Exponential convergence - time

November 2012
Back to peer counting
Improved peer counting

[Montresor, Ghodsi P2P 2009]

- Disadvantage of previous approach: estimate is initially very inaccurate
- Combination of average and T-man

Step 1
- Each node is assigned a random value $v_p$ from $I = \{0, \ldots, M-1\}$
- Distance $(x, y) = (y-x) \mod M$

Step 2
- Node are organized in a ring using T-man
Improved peer counting (2)

Step 3
- each node initiates and internal variable $a_p$ with its distance to its successor.
- in the absence of failure $\sum_p(a_p) = M$

Step 4
- the average protocol is run on the $a_p$ values
- at the end of the execution each node knows an approximation of $M/N$
- each node can easily derive $N$
Evaluation

Metrics
- Error: quality of the estimate (%)
- Convergence time: in s
- Overhead: amount of traffic
Scalability

---

Overhead per node (kB)

<table>
<thead>
<tr>
<th>Size</th>
<th>1.0</th>
<th>3.0</th>
<th>5.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Error (%)</td>
<td>10</td>
<td>20</td>
<td>30</td>
</tr>
</tbody>
</table>

Convergence Time (s)

<table>
<thead>
<tr>
<th>Size</th>
<th>2</th>
<th>4</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Error (%)</td>
<td>0</td>
<td>0.2</td>
<td>0.4</td>
</tr>
</tbody>
</table>

While epidemic protocols for peer counting have been proposed without periodic restarting, this is the subject of future work.

---

Convergence Time (s)

<table>
<thead>
<tr>
<th>Size</th>
<th>2</th>
<th>20</th>
<th>50</th>
</tr>
</thead>
<tbody>
<tr>
<td>Error (%)</td>
<td>10</td>
<td>20</td>
<td>30</td>
</tr>
</tbody>
</table>

---

IV. Discussion and Conclusions

- The accuracy of the estimation protocols increases with the size of the network.
- The overhead per node decreases as the network size grows.
- The convergence time increases with the network size.

References


Nodes were exceedingly inaccurate.

The failure of the initiator or its neighbors nodes could easily complicate the network.

Sensitivity to failures; in the first phases of the computation, the initiator.

The network size was initialized all nodes to 0, apart from one node set to 1 (the initiator).

The idea was to use continuous – i.e. able to continuously provide the estimate of the network size, without periodic restarting. This is the subject of future work.

This is because no node is more important than another.

It's important to note that the initial estimates have the same expected error.

The network size estimation is an important improvement over state-of-the-art epidemic protocols for peer counting.

Epidemic protocols for peer counting have been proposed by Jelasity et al. in 2005 (11); the idea was to use distributed schemes for size estimation in large and dynamic networks: A comparative study.

The system was evaluated in terms of scalable to arbitrary internet end hosts.

Performance Distributed Computing (HPDC'06), vol. 2735, Berkeley, CA, 2003, pp. 68–79.

These schemes were compared to local and global sampling, and other epidemic-based fast aggregations.

The system was also evaluated for accuracy under churn.

The accuracy of the system was measured under message losses.

Evaluation of the system was done in Peer-to-Peer Systems (HotP2P'08) conference.


Clustering peers with gossip
Clustering similar peers

- Vicinity: Introducing application-dependent proximity metric [VvS, EuroPar 2005]
- Two-layered approach
  1. Biased gossip reflecting some application semantic
  2. Unbiased peer sampling service
System model

- Semantic view of $l$ semantic neighbours
- Semantic proximity function $S(P,Q)$.
  - The higher the value of $S(P,Q)$, the “closer” the nodes.
  - The objective is to fill P’s semantic view to optimize

$$
\sum_{i=1}^{l} S(P, Q_i)
$$
Gossiping framework

- **Target selection**
  - Close peers
  - All nodes are examined: create a “small-world” like structure so that new nodes are discovered.
Gossip parameter setting

- Clustering protocol
  - Peer selection
    - tail “oldest timestamp”
  - Data exchange
    - aggressively biased,
    - select the $g$ items the closest from semantic and random views
  - Data processing
    - select the $l$ closest peers (buffer, semantic and random views)
- Peer sampling service
Evaluation

- Edonkey trace: 12,000 peers around 900,000 files
- Proximity function: overlap
  \[ S(F_p, F_q) = |F_p \cap F_q| \]
- \( l = 10 \)
- Simulations
- Hit ratio: search on 1 file randomly chosen
Convergence of semantic views (quality)

Figure 4. (a) Convergence of sem. views' quality. (b) Evolution of semantic views' quality for a sudden change in all users' interests at cycle 550.

4 Performance Evaluation

4.1 Convergence Speed

To evaluate the convergence speed of our algorithm, we test how quickly it finds nodes having files in common. The proximity function's objective is for each node to discover the $\ell$ peers that have the most common files with it. Therefore, a good metric of the progress towards this goal is the average number of common files between a node and each one of its semantic neighbors. From our traces, we measured that in the optimal organization, this metric has a value of 3.88.

Figure 4 (a) shows this metric as a function of the cycle for four distinct configurations. In favor of comparison fairness, the cache size and gossip length are 50 and 3, respectively, in each layer, for all configurations. The only exception is the first configuration, which has a single layer. In this case, the cache size and gossip length are 100 and 6, respectively. All experiments start with each node knowing 5 random other ones, simply to ensure initial connectivity in a single connected cluster.

In the first configuration, RANDOM VICINITY is running stand-alone. The progress of the semantic views' quality is rather steep in the first 100 cycles, but as nodes gradually concentrate on their very own neighborhood, getting to know new, possibly better peers becomes rare, and progress slows down.

In the second configuration, a two-layered approach consisting of RANDOM VICINITY and CYCLON is running. The slow start compared to stand-alone VICINITY is a reflection of the smaller VICINITY cache (3 as opposed to 6). However, the two-layered approach's advantage becomes apparent later, when CYCLON keeps feeding the RANDOM VICINITY layer with new, uniform randomly selected nodes, maintaining a higher progress rate, and outperforming stand-alone VICINITY in the long run.

In the third configuration, BIASED VICINITY demonstrates its contribution, as progress is significantly faster in the initial phase of the experiment. This is to be expected, since the items sent over in each BIASED VICINITY communication, are the ones that have been selected as the semantically closest to the recipient.
Epidemic-Style Management of Semantic Overlays for Content-Based Searching

![Semantic Hit Ratio Graph]

- **Figure 5.** Semantic Hit Ratio, for gossip lengths 1, 3, and 5 in each layer.

For gossip length 3, the average amount of data transferred to and from a node in one cycle is 38,400 bytes, while for gossip length 1, it is just 12,800. Considering the gossip period $T$ equal to 1 minute, this translates to an average bandwidth of 640 and 213 bytes per second, respectively. With gossip lengths 3, the system adapts faster to changes, but if bandwidth is of high concern, gossip length 1 can provide good results. Note that with a period of 1 minute, in the first 8 minutes we reach 85% of the optimal semantic hit ratio, having roughly 30% of all requests handled by the semantic neighbors.

We consider such a bandwidth consumption to be rather small, if not negligible compared to the bandwidth used for the actual file downloads. It is, in fact, a small price to pay for relieving the default search mechanism from about 35% of the search load.

**Discussion**

To the best of our knowledge, all earlier work on implicit building of semantic overlays relies on using heuristics to decide which of the peers that served a node recently are likely to be useful again in future queries [11–13]. However, all these techniques inhibit a weakness that challenges their applicability to the real world. They all assume a static network, free of node departures, which is a rather strong assumption considering the highly dynamic nature of file-sharing communities. Also, it is not clear how they perform in the presence of dynamic user preferences.

Regarding proximity-based P2P clustering, our work comes close to T-Man [8]. However, a key difference is that T-Man assumes continuous proximity metrics. That is, every node can point any other node to the right direction. This is not true in the problem we faced, i.e. in the case of completely unrelated peers. We dealt with it by harnessing CYCLOP’s randomness. This renders our solution more generic. Moreover, T-Man assumes a preconstructed almost random graph to start with. We make no such assumptions.
Improving routing: Kleinberg-like peer sampling
Motivation

- **Small-world overlay networks**
  - Neighbour set: Close + shortcuts
  - Theoretical analysis: Asymptotic bounds on routing performance (random versus Kleinberg’s shortcuts)

- **Epidemic-based overlay networks**
  - Decentralized overlay building and maintenance using gossip-based protocols
  - Practical systems: efficient routing

**Epidemic-based small-world networks**
Clustering protocols: close neighbours
Peer sampling service: shortcuts
Motivation

![Graph]

- 1 random shortcut
- 1 Kleinberg’s shortcut
- 10 random shortcuts
- 10 Kleinberg’s shortcuts

Average number of hops vs. Number of peers
Small world overlay network

- Neighbour set
  - Local contacts
  - Shortcuts
- Shortcut selection
  - Random [Watts & Strogatz 1998]
  - Greedy routing $O \left( n^{1/3} \right)$
  - Harmonic distribution [Kleinberg 2000] $O \left( \log^2 (n) \right)$
- Results
  - Asymptotic bounds: Magnitude order of routing performance
Shortcut selection and routing performance

- **Random selection**
  - Shortcuts picked uniformly at random
  - Greedy routing performance
    \[ O(n^{1/3}) \]

- **Kleinberg selection**
  - Selection with probability proportional to distance
    \[ \delta(B) = \frac{1}{d(A,B)^2} \]
  - B chosen by A with
    \[ P = \frac{\delta(B)}{\sum B \in S \delta(B)} \]
  - S = Set of peers not neighbour of A

- **Greedy routing performance**
  \[ O(\log^2(n)) \]
Small-world gossip-based networks

Assume each node has some coordinates in a d-dimensional space

**Clustering service**
- Peer selection: “closest”
- Data exchange: c entries
- Data Processing: “closest” kept

**Peer sampling service**
- Peer selection: random
- Data exchange: c/2 entries
- Data Processing: random

[Watts & Strogatz 1998]
Topologies

Grid, Manhattan distance
Close neighbours: neighbours on the Grid

Grid, Euclidean distance
Close neighbours: one in each wedge
Routing performance

- Example of the Grid, greedy routing
- \( r \) (distance to which local neighbours are chosen) = \( q \) (number of shortcuts) = 1
- \( f(d) \): average number of routing hops between 2 peers at distance \( d \).
- \( d(i) = \) probability that a shortcut is at distance \( d \).

\[
f(d) = 1 + \left( \sum_{i=0}^{d-2} d(i) f(i) \right) + \left( 1 - \sum_{i=0}^{d-2} d(i) \right)
\]

- \( d=1 \)
- Use of shortcut
- Use of local contact
Gossip-based small-world networks

- Leverage theory
- Decentralized selection of neighbours
  - Clustering protocol: local neighbours
  - Peer sampling: shortcuts
- Shortcut selection: peer sampling service
  - Random selection: random peer sampling
  - Kleinberg selection: tunes the view so that it matches the Kleinberg’s distribution
- What are we interested in?
  - Impact on the routing efficiency
  - Impact on the graph properties
Kleinberg’s peer sampling

- Use standard clustering protocol for local neighbors
- Shortcuts: bias peer sampling protocol to approximate Kleinberg’s distribution (probability of being kept is $1/d^2$)

**Peer sampling service**
- Peer selection: random
- Data exchange: k entries, (c-k) kept biased by Kleinberg’s distribution
- Data Processing: (c-k) entries exchanged

K=2
Implementation

Peer A

Peer B

Prob to keep as a Kleinberg shortcut

Data exchange: [E,F]

Data exchange: [I,J]
Kleinberg’s peer sampling

- Example

Peer A: B C D E F
Peer B: G H I J A

Peer selection

Kleinberg’s shortcuts

Data exchanged

Peer A: B C D I J
Peer B: G H E F A
Routing performance

Average number of hops

Number of cycles

Cyclon
Cyclonberg 1
Cyclonberg 2
Cyclonberg 3
Cyclonberg 4
Cyclonberg 5
Cyclonberg 6
Cyclonberg 7
Cyclonberg 8
Cyclonberg 9
Perfect Kleinberg
Shortcuts distribution

The graph shows the distribution of shortcuts percentage for different models over the distance (over the maximal distance). The models include Perfect Random, Cyclonberg2, Cyclonberg3, Cyclonberg4, Cyclonberg5, Cyclonberg6, Cyclonberg7, Cyclonberg8, and Perfect Kleinberg.
Outcomes

- Possible to tune the peer sampling to achieve a routing similar to the one obtained with a Kleinberg’s shortcut selection
  - Driven by the shuffle length

- Resulting graph properties
  - Degree distribution and average path length similar to a random peer sampling
  - Clustering coefficient: slightly higher
  - Harmless to most distributed applications

- Improves the clustering algorithm
Web personalization with gossip

[Bertier et al. Middleware 2010]
[Bai et al, ACM TODS 2011]
A real-world example

1- AMERICAN GIRL, NATIVE ENGLISH SPEAKING BABYSITTER IN LILLE.
2- Native English-speaking babysitter wanted!:Babysitting:Rennes, France
3- Brittany - looking for English-speaking babysitter rennes 35: The
4- Garde d'enfant à domicile

What if someone had made the right query?

Alice

Google

« English-Speaking baby-sitter Rennes »
Personalized query

**Daycare**
[babysitter: 500]

**Teaching Assistants**
[babysitter: 1]

**International schools**
[school, kids]

**Jonathan Coe novels**
[British authors, novels]

Alice  
Bob

November 2012
Example: personalized query expansion

English speaking baby sitter

Query expansion

English speaking baby sitter
Teaching assistant

www.assistantsinFrance.com
Example: Personalized top-k processing

English speaking baby sitter

Top-k processing

[ABLS, VLDB’08]

assistantsinFrance.com

Sport
Football: 14
Volley-bal: 2
Badminton: 1
...

Baby-sitter
AssistantinFrance: 5
Daycare: 1
...

assistantsinFrance.com
Which nodes should be considered for the social network?

- Tagging similarity
- Cosine similarity
- Multi-interest similarity
Interest-based Web 2.0 applications

- **Model**
  - $U(\text{sers}) \times I(\text{tems}) \times T(\text{ags})$
  - $Tagged_u(i, t)$: User $u$ annotates item $i$ with tag $t$
  - $Profile(u) = \{Tagged_u(i, t)\}$
1: Tagging similarity

- **Efficient network-aware search in collaborative tagging sites** [ABLS, VLDB’08]

- **User score: common tagging actions**

\[
\text{Score}_{ui}(uj) = \left| \text{Profile}(u_i) \bigcap \text{Profile}(u_j) \right|
\]

\[
\text{Score}_{ui}(uj) = \left| \{(i,t), \text{Tagged}_{ui}(i,t) \land \text{Tagged}_{uj}(i,t)\} \right|
\]
2: Item cosine similarity

Normalized overlap
- bigger overlap increases the score
- no shared interests decreases it
- directly takes into account the weight of items

$$ItemCos(\tilde{u}_1, \tilde{u}_2) = \frac{|Items(\{\tilde{u}_1\}) \cap Items(\{\tilde{u}_2\})|}{\sqrt{|Items(\{\tilde{u}_1\})| \cdot |Items(\{\tilde{u}_2\})|}}$$
3: Coping with multi-interests

**Item cosine similarity**: favours specific and dominant interests

![Diagram showing the relationship between individual and multi-interest ratings with Alice's interests highlighted.](image)
3: Multi-Interest cosine similarity

- Rate the set of users **as a whole** instead of individuals
- Choose a set of users that covers the user’s interests

\[
SetItemVect(set) = \sum_{p \in set} \frac{(ItemVect(p) \otimes ItemVect(n))}{\|ItemVect(p)\|}
\]

\[
SetScore(n, set) = SetItemVect(set).ItemVect(n)^* \cos(SetItemVect(set), ItemVect(n))^b
\]

*Items of interest for nodes in set(n)*
*Normalized not to take into account non shared interests*
*Distribution*
Heuristic

**Input:** set of nodes `candidateSet`

**Output:** a view of size `viewSize`

`bestView = {}`

**for** `setSize` **from** 1 to `viewSize` **do**

**foreach** `candidate` in `candidateSet` **do**

`candidateView = bestView U {candidate}`
`viewScore = SetScore(candidateView)`

`bestCandidate = candidate` that got the highest `viewScore`

`bestView = bestView U {bestCandidate}`

`candidateSet` -= `{bestCandidate}`

**Result:** `bestView`
Multi-interest acquaintances?

![Graph showing normalized recall over b]

- delicious
- edonkey
- citeulike
- lastfm
Piling up gossip protocols

Gossip (Bloom-filter based) similarity protocol.

Gossip-based peer sampling service
An implicit social network

### Friends

<table>
<thead>
<tr>
<th>@IP:port</th>
<th>132.154.8.5:2020</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bloom Filter</td>
<td>01011011001</td>
</tr>
<tr>
<td>Profile</td>
<td><a href="http://www.inria.fr">www.inria.fr</a>: inria, computer</td>
</tr>
<tr>
<td></td>
<td><a href="http://www.assistants.fr">www.assistants.fr</a>: baby-sitter, english</td>
</tr>
<tr>
<td>Update time</td>
<td>5</td>
</tr>
</tbody>
</table>

### Uniform sample

<table>
<thead>
<tr>
<th>@IP: port</th>
<th>102.14.18.1:2110</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bloom Filter</td>
<td>100100000110</td>
</tr>
<tr>
<td>Update time</td>
<td>30</td>
</tr>
</tbody>
</table>
Bloom filter

h hash functions are used on the item to obtain h positions in the array:

Possible false positive
Building the social network

- Two gossip protocols
  - Similarity-based Peer Sampling
  - Random Peer Sampling

- When $p$ encounters $q$
  - Evaluate potential new view, based on set similarity metric
  - Use of Bloom filters to limit the communication overhead
Convergence du SPS

Ratio between $c$ current neighbours and the $c$ ideal ones

Random sampling

Biased sampling

Cycles
Set item cosine similarity

![Graph showing normalized recall over cycles for different simulation types and bootstrap settings.

- Bootstrap simulation b=0
- Bootstrap simulation b=4
- Bootstrap planetlab b=4
- Nodes joining simulation b=4]
Bandwidth usage at cold start

![Graph showing bandwidth usage and downloaded profiles over cycles.](image)

- **Downloaded profiles per user in simulation**
- **Bandwidth usage in simulation**
- **Bandwidth usage in Planetlab**

November 2012
Applications

Query expansion

- English speaking baby sitter
- Teaching assistant

Top-k

- [English speaking, baby sitter]
- Personalized Top-K
Illustration

Query expansion

[Bertier et al, Middeware 2010]
Expanding queries

**GNet(n)**
Personalized Network

Profiles (items, tags)

**TagMap (n)**
Personalized view of the tag relations

Set cosine similarity metric

Item cosine similarity metric between tags

Data structures

**Query Expansion**

Expanded query (initial and additional weighted tags)

Search engine

Results
Evaluation metrics: recall & precision

November 2012
Personalized view of the world

The TagMap
Relating tags: Item tag cosine

- GNet(u): set of nodes in the network
- Item Cosine similarity
- Distance between tags $t_i$ and $t_j$ as seen by node $n$
  
  For each tag: number of occurrences of the tag in GNet (u)

\[
V_t [item_i] = v
\]

\[
TagMap[t_i,t_j] = \cos(\vec{V}_{t_i},\vec{V}_{t_j})
\]
The TagMap: tag/tag score

<table>
<thead>
<tr>
<th></th>
<th>Music</th>
<th>BritPop</th>
<th>Vivaldi</th>
<th>ColdPlay</th>
</tr>
</thead>
<tbody>
<tr>
<td>Music</td>
<td>1</td>
<td>0.7</td>
<td>0.1</td>
<td>0</td>
</tr>
<tr>
<td>BritPop</td>
<td>1</td>
<td>0</td>
<td>0.7</td>
<td></td>
</tr>
<tr>
<td>Vivaldi</td>
<td></td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>ColdPlay</td>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
</tbody>
</table>
Expanding Query
Direct reading (DR)

- Query expansion of size $q$
- Select the $q$ tags scoring the highest

$$score_{DR}(t_i) = \sum_{t \in \text{InitialQuery}} \text{TagMap}[t, t_i]$$

- DR will never associate Music and Oasis
- Worse DR could expand Music with Bach
Expanding queries with GRank

Item sparsity: hidden relationships between tags

GRank: computation of tag centrality
- Adaptation of the PageRank algorithm
- Relative importance of a tag for a given tag and user
Expanding queries

Example of TagMap’s Graph

Initial Query = Music

Direct Reading (Social Ranking)

GOSSPLE Query Expansion

Weighted expanded queries

Search Engine

Music: 1
BritPop: 0.7
Bach: 0.1

Music: 0.4425
BritPop: 0.315
Oasis: 0.147
Recall

50,000 user Delicious trace

37% of requests not satisfied w/o QE based on the information of 20 neighbors

Recall for the items originally not found

query expansion size

Social Ranking

Gossple 10 neighbors
Gossple 20 neighbors
Gossple 100 neighbors
Gossple 2000 neighbors

Nov 2012
References


