CS 347
Parallel and Distributed Data Processing
Spring 2016

Notes 10: Publish/Subscribe Systems

Point-to-Point Communication

To: Alice
From: Bob
Message: "M"

Publish/Subscribe Communication

Description:
- Message: "M"
- Query: "Q"
- Identifier: "I"

Publish/Subscribe Communication

Description:
- Message: "M"
- Query: "Q"
- Identifier: "I"
Publish/Subscribe Applications

- Downstream/end user notifications
- Operational monitoring
- Log aggregation
- Application integration
- Stream processing
- Social networking

Publish/Subscribe Semantics

subscribe(\(Q, I\)):
- add \([Q, I]\) to SDB // subscription database

update(D, M):
- for \([Q, I]\) in SDB do
  - if match(D, Q) then notify(I, M)

PDB Description:
- \(<D>\)
- \(<M>\)

Query:
- \(<Q>\)
- \(<I>\)

Subscription

Can send email, make remote procedure call, write M to DB, etc.

Publish/Subscribe Semantics

publish(D, M):
- add \([D, M]\) to PDB // publication database

query(Q, I):
- for \([D, M]\) in PDB do
  - if match(D, Q) then notify(I, M)
Publish/Subscribe Features

**Space decoupling**
Interacting parties do not need to know each other

**Time decoupling**
Interacting parties do not need to actively participate at the same time

**Synchronization decoupling**
Publishers and subscribers do not block for each other

Other Communication Models

- Message passing (through channels)
- Message queues
- Remote procedure calls (RPCs)
- Shared memory (bulletin boards)

Description/Query Models

**Flat topics**
E.g., Topics = \{business, politics, sports, ...\}

T is set of possible topics
Description \(D\) is a subset of \(T\)
Query \(Q\) is a subset of \(T\)

match\((D, Q)\) = true if \(D \cap Q \neq \emptyset\)

<table>
<thead>
<tr>
<th>Publication</th>
<th>Subscription</th>
</tr>
</thead>
<tbody>
<tr>
<td>Message: 1</td>
<td>Identifier: 1</td>
</tr>
<tr>
<td>Message: 2</td>
<td>Identifier: 2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Publication</th>
<th>Subscription</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description: {b, s}</td>
<td>Query: {p, s}</td>
</tr>
<tr>
<td>Message: ...</td>
<td>Identifier: ...</td>
</tr>
</tbody>
</table>

Description/Query Models

- Flat topics
E.g., Topics = \{business, politics, sports, ...\}
Description/Query Models

Topic hierarchy

all

sports

football

soccer

college

business

tech

politics

hockey

NFL

descrip.

Query

Description path \( d \) matches query path \( q \) if \( q \) is a prefix of \( d \)

\( D \) matches \( Q \) if there exists a path in \( Q \) that matches a path in \( D \)

P1.D = { all/sports/football }  
P2.D = (all/sports)  
P3.D = {all/business/tech, all/politics}

S1.Q = {all/sports/soccer}  
S2.Q = {all/politics/canada, all/sports/hockey}  
P2.D = (all/sports)  
P3.D = {all/business/tech, all/politics}

Description/Query Models

Key-value pairs

P1.D = { [price, 50], [size, L] }  
P2.D = { [price, 50] }  
P3.D = { [size, M], [size, L] }  
S1.Q = { [price, 50] }  
S2.Q = { [price, 50], [size, M] }
Description/Query Models

Key-value pairs

\[ P_1.D = \{ \text{price}, 50 \}, \text{size}, \text{L} \} \]
\[ P_2.D = \{ \text{price}, 80 \} \]
\[ P_3.D = \{ \text{size}, \text{M} \}, \text{size}, \text{L} \} \]
\[ S_1.Q = \{ \text{price}, 50 \} \]
\[ S_2.Q = \{ \text{price}, 50 \}, \text{size}, \text{M} \} \]
\[ S_3.Q = \{ \text{price} > 40 \} \ \text{AND} \ \text{size} \neq \text{L} \]

Matching Description to Queries

\[ \text{Publication} \]
\[ \text{Subscription} \]
\[ \text{DB} \]

Matching Description to Queries

\[ \text{Publication} \]
\[ \text{Subscription} \]

Generic Distributed Matching

\[ P_i \] (To any row)
\[ S_j \] (To any column)
\[ \text{match}(Q_j, D_i) \]
Generic Distributed Matching

\( P_i \) (To any row)

\( S_j \) (To any column)

\( \text{match}(Q_j, D_i) \)

Publish to one of \{a, b\}, \{c, d\}, \{e, f\}
Subscribe to one of \{a, c, e\}, \{b, d, f\}

Can use any quorum

Familiar?
Generic Distributed Matching

**Cost**
- Replicated data (stored subscriptions)
- Balanced load (processed publications)

**Cost example**
At node with \( x \) subscriptions handling \( y \) publications

\[
data(x) = x
\]

Scenario 1:
\[
\text{work}(x, y) = xy
\]
Scenario 2:
\[
\text{work}(x, y) = y
\]

For a 6-node grid with \( s \) subscriptions and \( p \) publications
Each node handles \( s/2 \) subscriptions, \( p/3 \) publications

**Scenario 1**
- total_data = 6 \( \text{data}(s/2) = 3s \)
- total_work = 6 \( \text{work}(s/2, p/3) = 6 (s/2)(p/3) = sp \)

**Scenario 2**
- total_data = 6 \( \text{data}(s/2) = 3s \)
- total_work = 6 \( \text{work}(s/2, p/3) = 6 (p/3) = 2p \)
Topic Matching
E.g., $T = \{t_1, t_2, t_3\}$

Data fragmentation
E.g., on subscriptions
Query localization
E.g., for publications

Topic Hierarchy Matching

Publication dissemination tree

Topic Hierarchy Matching

Publication dissemination tree

Subscriptions to 
(t/3/1 or t/3/2)

Subscriptions to 
(t/3/2 or t/3/3)

Note the replication

Dynamic Dissemination Tree

Publication dissemination tree

Subscriptions to 
(t/1)

Note: Neighborhood
What if all nodes can publish?

Matching at One Node

Set \( \{ [Q_i, I_i] \} \) of stored subscriptions
Match one publication \( p \) from stream

Match semantics
Each publication \( p \) is bag of terms
Each subscription \( s \) has set of terms
There is a match when all \( s \) terms appear in \( p \)
Matching at One Node

Example

<table>
<thead>
<tr>
<th>Subscriptions</th>
<th>Inverted lists</th>
</tr>
</thead>
<tbody>
<tr>
<td>s1 (a, b)</td>
<td>a → S1 S2 S3</td>
</tr>
<tr>
<td>s2 (a, d)</td>
<td>b → S1 S2</td>
</tr>
<tr>
<td>s3 (a, d, e)</td>
<td>c → S1 S2</td>
</tr>
<tr>
<td>s4 (b, f)</td>
<td>d → S1 S2 S3</td>
</tr>
<tr>
<td>s5 (c, d, e, f)</td>
<td>e → S1 S2 S3</td>
</tr>
</tbody>
</table>

Sample publication
a c a f b c
f → S1 S2 S3

s1 = (a, b) matches publication if both terms a and b appear in it
Can generalize, e.g., s1 = a ∨ (b ∧ c) handled as two subscriptions: s1 = (a ∧ b), s2 = (a ∧ c)

Matching at One Node

Example

<table>
<thead>
<tr>
<th>Subscriptions</th>
<th>Inverted lists</th>
</tr>
</thead>
<tbody>
<tr>
<td>s1 (a, b)</td>
<td>a → S1 S2 S3</td>
</tr>
<tr>
<td>s2 (a, d)</td>
<td>b → S1 S2</td>
</tr>
<tr>
<td>s3 (a, d, e)</td>
<td>c → S1 S2</td>
</tr>
<tr>
<td>s4 (b, f)</td>
<td>d → S1 S2 S3</td>
</tr>
<tr>
<td>s5 (c, d, e, f)</td>
<td>e → S1 S2 S3</td>
</tr>
</tbody>
</table>

Sample publication
a c a f b c
f → S1 S2 S3

Intersection of lists for a, b, c, f not useful (∅)
Union of lists = { s1, s2, s3, s4, s5 } gives candidate subscriptions
Need to check each candidate (e.g., s1 matches but s2 does not)

Matching at One Node

Counting method

<table>
<thead>
<tr>
<th>Subscriptions</th>
<th>Inverted lists</th>
<th>Total</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>s1 (a, b)</td>
<td>a → S1 S2 S3</td>
<td>S1</td>
<td>2</td>
</tr>
<tr>
<td>s2 (a, d)</td>
<td>b → S1 S2</td>
<td>S2</td>
<td>2</td>
</tr>
<tr>
<td>s3 (a, d, e)</td>
<td>c → S1 S2</td>
<td>S3</td>
<td>2</td>
</tr>
<tr>
<td>s4 (b, f)</td>
<td>d → S1 S2 S3</td>
<td>S4</td>
<td>2</td>
</tr>
<tr>
<td>s5 (c, d, e, f)</td>
<td>e → S1 S2 S3</td>
<td>S5</td>
<td>3</td>
</tr>
</tbody>
</table>

Sample publication
a c a f b c
f → S1 S2 S3

Distinct term set
a b c f

Matching at One Node

Counting method

<table>
<thead>
<tr>
<th>Subscriptions</th>
<th>Inverted lists</th>
<th>Total</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>s1 (a, b)</td>
<td>a → S1 S2 S3</td>
<td>S1</td>
<td>2</td>
</tr>
<tr>
<td>s2 (a, d)</td>
<td>b → S1 S2</td>
<td>S2</td>
<td>2</td>
</tr>
<tr>
<td>s3 (a, d, e)</td>
<td>c → S1 S2</td>
<td>S3</td>
<td>3</td>
</tr>
<tr>
<td>s4 (b, f)</td>
<td>d → S1 S2 S3</td>
<td>S4</td>
<td>2</td>
</tr>
<tr>
<td>s5 (c, d, e, f)</td>
<td>e → S1 S2 S3</td>
<td>S5</td>
<td>4</td>
</tr>
</tbody>
</table>

Sample publication
a c a f b c
f → S1 S2 S3

Distinct term set
a b c f

When computing the union, count number of times each subscription appears
If count ≥ total then subscription matches
Matching at One Node

Counting method

<table>
<thead>
<tr>
<th>Subscriptions</th>
<th>Inverted lists</th>
<th>Total</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>s1 (a, b)</td>
<td>a → s1 s1 s1</td>
<td>s1</td>
<td>2</td>
</tr>
<tr>
<td>s1 (a, d)</td>
<td>b → s2</td>
<td>s2</td>
<td>2</td>
</tr>
<tr>
<td>s1 (a, d, e)</td>
<td>c → s3</td>
<td>s3</td>
<td>2</td>
</tr>
<tr>
<td>s1 (b, f)</td>
<td>d → s4 s4 s4</td>
<td>s4</td>
<td>4</td>
</tr>
<tr>
<td>s1 (c, d, e, f)</td>
<td>e → s5 s5 s5</td>
<td>s5</td>
<td>0</td>
</tr>
</tbody>
</table>

Sample publication: a c a f b c

Distinct term set: a b c f

When computing the union, count number of times each subscription appears. If count ≥ total, then subscription matches.

Matching at One Node

Counting method

<table>
<thead>
<tr>
<th>Subscriptions</th>
<th>Inverted lists</th>
<th>Total</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>s1 (a, b)</td>
<td>a → s1 s1 s1</td>
<td>s1</td>
<td>2</td>
</tr>
<tr>
<td>s1 (a, d)</td>
<td>b → s2</td>
<td>s2</td>
<td>2</td>
</tr>
<tr>
<td>s1 (a, d, e)</td>
<td>c → s3</td>
<td>s3</td>
<td>2</td>
</tr>
<tr>
<td>s1 (b, f)</td>
<td>d → s4 s4 s4</td>
<td>s4</td>
<td>4</td>
</tr>
<tr>
<td>s1 (c, d, e, f)</td>
<td>e → s5 s5 s5</td>
<td>s5</td>
<td>0</td>
</tr>
</tbody>
</table>

Sample publication: a c a f b c

Distinct term set: a b c f

When computing the union, count number of times each subscription appears. If count ≥ total, then subscription matches.
Matching at One Node

Key method

<table>
<thead>
<tr>
<th>Subscriptions</th>
<th>Inverted lists</th>
<th>Occurrence table</th>
</tr>
</thead>
<tbody>
<tr>
<td>s₁: {a, b}</td>
<td>a → s₁[1, 0, b]</td>
<td>a → s₁[1, 0]</td>
</tr>
<tr>
<td>s₂: {a, d}</td>
<td>a → s₂[1, 0]</td>
<td>b → s₂[1, 0]</td>
</tr>
<tr>
<td>s₃: {a, d, e}</td>
<td>c → s₃[1, 0, d, e]</td>
<td>c → s₃[1, 0]</td>
</tr>
<tr>
<td>s₄: {b, f}</td>
<td>d → s₄[1, 0, d, e, f]</td>
<td>d → s₄[1, 0]</td>
</tr>
<tr>
<td>s₅: {c, d, e, f}</td>
<td>e → s₅[1, 0, d, e, f]</td>
<td>e → s₅[1, 0]</td>
</tr>
</tbody>
</table>

Sample publication: a c a f b c

Distinct term set: a b c f

Matching at One Node

Key method

<table>
<thead>
<tr>
<th>Subscriptions</th>
<th>Inverted lists</th>
<th>Occurrence table</th>
</tr>
</thead>
<tbody>
<tr>
<td>s₁: {a, b}</td>
<td>a → s₁[1, 0, b]</td>
<td>a → s₁[1, 0]</td>
</tr>
<tr>
<td>s₂: {a, d}</td>
<td>a → s₂[1, 0]</td>
<td>b → s₂[1, 0]</td>
</tr>
<tr>
<td>s₃: {a, d, e}</td>
<td>c → s₃[1, 0, d, e]</td>
<td>c → s₃[1, 0]</td>
</tr>
<tr>
<td>s₄: {b, f}</td>
<td>d → s₄[1, 0, d, e, f]</td>
<td>d → s₄[1, 0]</td>
</tr>
<tr>
<td>s₅: {c, d, e, f}</td>
<td>e → s₅[1, 0, d, e, f]</td>
<td>e → s₅[1, 0]</td>
</tr>
</tbody>
</table>

Sample publication: a c a f b c

Distinct term set: a b c f

Each subscription only appears in one inverted list.
Each inverted list entry contains the additional terms in the subscription.
The occurrence table is for fast (hash) lookup for matching additional terms.

Case Study: Twitter

S(e) = {a, d}

Follows inverted lists:
S[a]: b
S[b]: a, d, e
S[c]: d
S[d]: -
S[e]: a, d

Is-followed inverted lists:
S⁻¹[a]: b, e
S⁻¹[b]: a
S⁻¹[c]: b
S⁻¹[d]: b, c, e
S⁻¹[e]: b

Publications by e have description "e".
Messages are 140 characters max.
Users periodically check for updates.
Twitter Architecture

Centralized

Frontend

Users

Notifications by user

Publications by user

Follows lists

Is followed lists

Backend

Log (all publications)

Distributed

Frontend

Users

Notifications by user

Publications by user

Follows lists

Is followed lists

Backend

Log (all publications)

How to split backends?

Frontend

Notifications by user

Publications by user

Follows lists

Is followed lists

Backend

Log (1/2 publications)

Distributed

Notifications by user

Publications by user

Follows lists

Is followed lists

Backend

Log (1/2 publications)

Distributed

Notifications by user

Publications by user

Follows lists

Is followed lists

Backend

Log (1/2 publications)
Case Study: Kafka

Apache Kafka
Distributed publish-subscribe messaging system
Distributed, partitioned, replicated commit log

Kafka Publishers
Set of flat topics
One or more partitions per topic
Total immutable order and persistence of messages in partition
Producers decide which message goes to which partition

Kafka Subscribers
Subscribers organized in consumer groups
Within each group, each partition can be read by ≤1 consumer

Kafka Scenarios
Publish/subscribe (broadcast to all subscribers)
1 consumer per group
**Kafka Scenarios**

Message queue (send to one in a pool of consumers)
All consumer in a single group

![Diagram of Kafka Scenarios]

**Kafka Implementation**

Partitions act like database logs
Are truncated periodically

Partitions can be replicated using RPWP
Each server can be primary for some, backup for others
Good for load balancing across servers

**Summary**

Publish/subscribe semantics
Description/query models
Distributed matching
  Generic
  Topic
  Topic hierarchy
Matching at one node
Case studies