Point-to-Point Communication

To: Alice
From: Bob
Message: <M>

Publish/Subscribe Communication

Publication
Description: <D>
Message: <M>

Subscription
Query: <Q>
Identifier: <I>

Publish/Subscribe System

Publication
Description: <D>
Message: <M>

Subscription
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)

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, \ldots\}

\[ \text{match}(D, Q) = \begin{cases} \text{true} & \text{if } D \cap Q \neq \emptyset \\ \text{false} & \text{otherwise} \end{cases} \]
Description/Query Models

Topic hierarchy

all

sports

business

politics

soccer

football

hockey

tech

service

college

NFL

Description/Query Models

Topic hierarchy

all

sports

business

politics

soccer

football

hockey

tech

service

college

NFL

P_1.D = \{ all/sports/football \}

S_1.Q = \{ all/sports/soccer \}

P_2.D = \{ all/sports \}

S_2.Q = \{ all/politics/canada, all/sports/hockey \}

P_3.D = \{ all/business/tech, all/politics \}

Description/Query Models

Topic hierarchy

all

sports

business

politics

soccer

football

hockey

tech

service

college

NFL

Description/Query Models

Topic hierarchy

all

sports

business

politics

soccer

football

hockey

tech

service

college

NFL

P_1.D = \{ price, 50 \}, \{ size, L \}

S_1.Q = \{ price, 50 \}

P_2.D = \{ price, 80 \}

S_2.Q = \{ price, 50, [size, M] \}

P_3.D = \{ [size, M], [size, L] \}

S_3.Q = \{ [price, 50], [size, M] \}

Description/Query Models

Topic hierarchy

all

sports

business

politics

soccer

football

hockey

tech

service

college

NFL

Description/Query Models

Topic hierarchy

all

sports

business

politics

soccer

football

hockey

tech

service

college

NFL

P_1.D = \{ price, 50 \}, \{ size, L \}

S_1.Q = \{ price, 50 \}

P_2.D = \{ price, 80 \}

S_2.Q = \{ price, 50, [size, M] \}

P_3.D = \{ [size, M], [size, L] \}

S_3.Q = \{ [price, 50], [size, M] \}

Key-value pairs

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.
Description/Query Models

Key-value pairs

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

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

Familiar?

Can use any quorum
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: \( \text{work}(x, y) = y \)

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

Scenario 1
\[
\begin{align*}
\text{total\_data} &= 6 \times \text{data}(s/2) = 3s \\
\text{total\_work} &= 6 \times \text{work}(s/2, p/3) = 6 \times (s/2) \times (p/3) = sp
\end{align*}
\]

Scenario 2
\[
\begin{align*}
\text{total\_data} &= 6 \times \text{data}(s/2) = 3s \\
\text{total\_work} &= 6 \times \text{work}(s/2, p/3) = 6 \times (p/3) = 2p
\end{align*}
\]

Cost example
For a single node

Scenario 1
\[
\begin{align*}
\text{total\_data} &= s \\
\text{total\_work} &= sp
\end{align*}
\]

Scenario 2
\[
\begin{align*}
\text{total\_data} &= s \\
\text{total\_work} &= p
\end{align*}
\]
**Topic Matching**

E.g., \( T = \{ t_1, t_2, t_3 \} \)

**Topic Hierarchy Matching**

E.g., \( T = \{ t_1, t_2, t_3 \} \)

---

**Data fragmentation**

E.g., on subscriptions

Query localization

E.g., for publications

**Publication dissemination tree**

All publications

\( t/1 \)

\( t/2 \)

\( t/3 \)

\( t/3/1 \) publications

processed here

\( t/3/1 \)
Topic Hierarchy Matching

Publication dissemination tree

All publications

Note the replication

Dynamic Dissemination Tree

Publication dissemination tree

Close node

Neither node

Subscriptions: \{t/1\}

Neighborhood
What if all nodes can publish?

**Matching at One Node**

Set \( \{ [Q_j, I_j] \} \) 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

Subscriptions
s_1 \{ a, b \}
s_2 \{ a, d \}
s_3 \{ a, d, e \}
s_4 \{ b, f \}
s_5 \{ c, d, e, f \}

Inverted lists
a \rightarrow s_1, s_2, s_3
b \rightarrow s_1, s_5
\vdots
d \rightarrow s_2, s_3, s_5
e \rightarrow s_3
f \rightarrow s_4, s_5

Sample publication
a, c, a, f, b, c

s_1 = \{ a, b \} matches publication if both terms a and b appear in it
Can generalize e.g., s_4 = a \land (b \lor c) handled as two subscriptions: s_{j1} = (a \land b), s_{j2} = (a \land c)

Subscriptions
s_1 \{ a, b \}
s_2 \{ a, d \}
s_3 \{ a, d, e \}
s_4 \{ b, f \}
s_5 \{ c, d, e, f \}

Inverted lists
a \rightarrow s_1, s_2, s_3
b \rightarrow s_1, s_5
\vdots
d \rightarrow s_2, s_3, s_5
e \rightarrow s_3
f \rightarrow s_4, s_5

Sample publication
a, c, a, f, b, c

Intersection of lists for a, b, c, f not useful (= ∅)
Union of lists = \{ s_1, s_2, s_3, s_4, s_5 \} gives candidate subscriptions
Need to check each candidate (e.g., s_1 matches but s_2 does not)

Counting method

Subscriptions
s_1 \{ a, b \}
s_2 \{ a, d \}
s_3 \{ a, d, e \}
s_4 \{ b, f \}
s_5 \{ c, d, e, f \}

Inverted lists
a \rightarrow s_1, s_2, s_3
b \rightarrow s_1, s_5
\vdots
d \rightarrow s_2, s_3, s_5
e \rightarrow s_3
f \rightarrow s_4, s_5

Sample publication
a, c, a, f, b, c

Total Count

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

Subscriptions
s1 \{ a, b \}
s2 \{ a, d \}
s3 \{ a, d, e \}
s4 \{ b, f \}
s5 \{ c, d, e, f \}

Inverted lists

Sample publication
a c a f b c

Distinct term set
a b c f

Total Count
s1 2 2
s2 2 1
s3 3 1
s4 2 2
s5 4 0

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

Matching at One Node

Counting method

Subscriptions
s1 \{ a, b \}
s2 \{ a, d \}
s3 \{ a, d, e \}
s4 \{ b, f \}
s5 \{ c, d, e, f \}

Inverted lists

Sample publication
a c a f b c

Distinct term set
a b c f

Total Count
s1 2 2
s2 2 1
s3 3 1
s4 2 2
s5 4 2

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 \rightarrow s1(b)</td>
<td>a</td>
</tr>
<tr>
<td>s2: {a, d}</td>
<td>b \rightarrow s2(d)</td>
<td>b</td>
</tr>
<tr>
<td>s3: {a, d, e}</td>
<td>c \rightarrow s3(d, e)</td>
<td>c</td>
</tr>
<tr>
<td>s4: {b, f}</td>
<td>d \rightarrow null</td>
<td></td>
</tr>
<tr>
<td>s5: {c, d, e, f}</td>
<td>e \rightarrow null</td>
<td></td>
</tr>
</tbody>
</table>

**Distinct term set**

\[\{a, b, c, f\}\]

Each subscription only appears in one inverted list

Case Study: Twitter

**S(e)** = \{a, d\}

Follows inverted lists
S(a): b, e
S(b): a, c, d, e
S(c): d
S(d): e, a, d
S(e): a, d

Is-followed inverted lists
S'(a): b, e
S'(b): a
S'(c): b
S'(d): b, e
S'(e): b

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

Centralized

Users

Frontend

Backend

Log (all publications)

Notifications by user

Publications by user

a
b
c
d

Notifications by user

Publications by user

a
b
c
d

Twitter Architecture

Distributed

Users

Frontend

Frontend

Frontend

Backend

Notifications by user

Publications by user

a
b
c
d

Notifications by user

Publications by user

a
b
c
d

Twitter Architecture

Distributed

How to split backends?

Notifications by user

Publications by user

a
b
c
d

Notifications by user

Publications by user

a
b
c
d

Twitter Architecture

Distributed

Notifications by user

Publications by user

a
b
c
d

Notifications by user

Publications by user

a
b
c
d

How to split backends?
**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

---

Partition 1
\[ m_{11} \quad m_{12} \quad m_{13} \quad m_{14} \]

Partition 2
\[ m_{21} \quad m_{22} \]

Partition 3
\[ m_{31} \quad m_{32} \quad m_{33} \]

Time

Kafka Publishers

Partitions

Kafka Subscribers

Kafka Scenarios
Kafka Scenarios

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

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