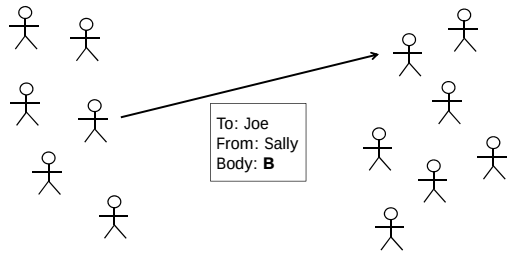
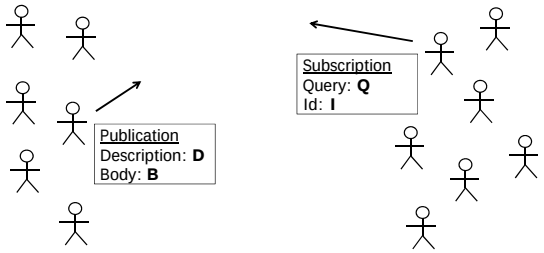
CS 347:
Parallel and Distributed
Data Management
Notes XX: Publish/Subscribe

Hector Garcia-Molina

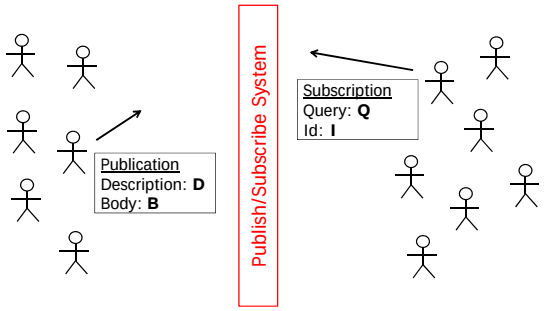
Point to Point Communication



Publish/Subscribe Communication



Publish/Subscribe Communication



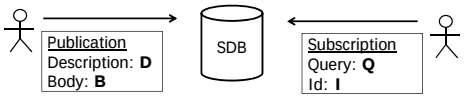
P/S Semantics

- Subscribe(query Q; id I):
  add [Q, I] to SDB
- Publish(description D; body B):
  for [Q,I] in SDB do
  if Q[D] then notify(I, B)

P/S Semantics

- Subscribe(query Q; id I):
  add [Q, I] to SDB
- Publish(description D; body B):
  for [Q,I] in SDB do
  if Q[D] then notify(I, B)

Notify can send email,
or can put B in DB(I)
(which I checks periodically)



P/Q Semantics (Alternative)

- Publish(description D; body B):
  add [D, B] to PDB
- Query(query Q; id I):
  for [D, B] in PDB do
    if Q[D] then notify(I, B)

P/Q Semantics (Alternative)

- Publish(description D; body B):
  add [D, B] to PDB
- Query(query Q; id I):
  for [D, B] in PDB do
    if Q[D] then notify(I, B)

Notify could just send out new material (not seen by previous queries

P/S Features

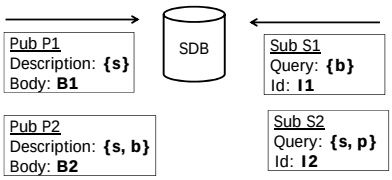
- *Space decoupling:* interacting parties do not need to know each other
- *Time decoupling:* interacting parties no not need to actively participate at the same time
- *Synchronization decoupling:* publishers and subscribers do not block for each other

Other Communication Models

- Message passing
- RPC (remote procedure call)
- Shared Spaces (bulletin board)
- Message Queues

Description/Query Models

- (flat) Topics
  - e.g.: Topics = {sports, business, politics, ...}

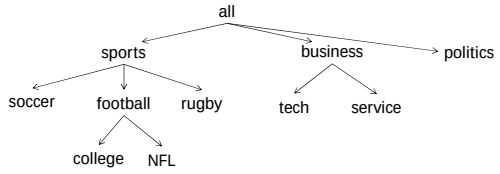


Description/Query Models

- (flat) Topics
  - *T* is set of possible topics
  - description D is a subset of T
  - query Q is a subset of T
  - D matches Q if $D \cap Q \neq \emptyset$ (empty set)

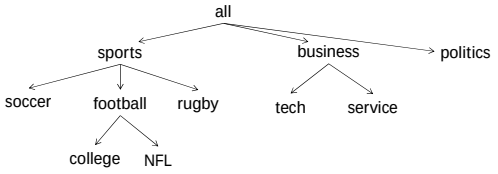
Description/Query Models

- Topic Hierarchy



Description/Query Models

- Topic Hierarchy



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

S1.Q=all.sports.rugby
S2.Q={all.politics.calif,
all.sports.soccer}

Description/Query Models

- (flat) Topics
  - *T* is tree of topics (or DAG?)
  - description *D* is a set of paths in *T*
  - query *Q* is a set of paths in *T*
  - 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

P1.D={[price, 50], [size, L]}
P2.D={[price, 80]}
P3.D={[size, M], [size, L]}

S1.Q={[price, 50]}
S2.Q={[price, 50], [size, M]}

Description/Query Models

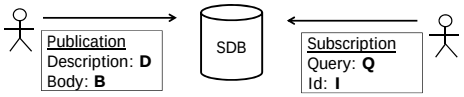
- Key-value pairs

P1.D={[price, 50], [size, L]}
P2.D={[price, 80]}
P3.D={[size, M], [size, L]}

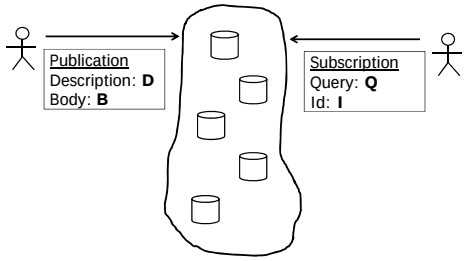
S1.Q={[price, 50]}
S2.Q={[price, 50], [size, M]}

s3.Q= [price > 40] AND
[size = L]

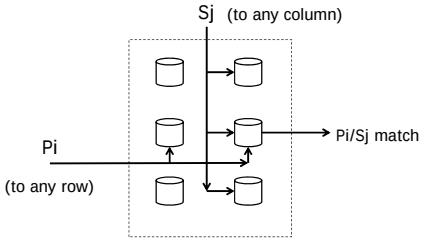
Matching Descriptions to Queries



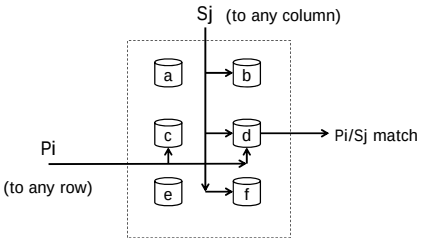
Matching Descriptions to Queries



Generic Distributed Matching



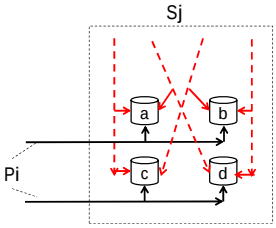
Generic Distributed Matching



- pub to one of {a,b}, {c,d}, {e,f}
- sub to one of {a,c,e}, {b,d,f}

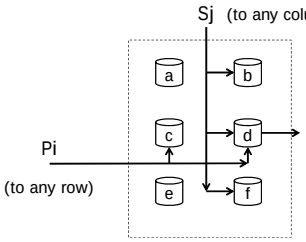
sound familiar??

Can use any abort/commit quorums



pub to one of {{a,b}, {c,d}}
sub to one of {{a,c}, {a,d}, {b,c}, {b,d} }

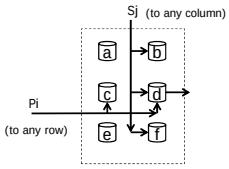
Generic Distributed Matching



Issues:
- Replicated data (subs)
- Load balance
- Total work

- pub to one of {a,b}, {c,d}, {e,f}
- sub to one of {a,c,e}, {b,d,f}

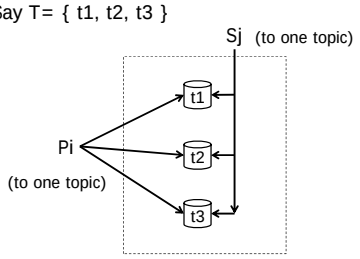
Simple Cost Model



- At node with x subs, y pubs: work(x,y), data(x)
- Ex: work(x,y)=xy (case (i))
  work(x,y)=y (case (ii))
  data(x)=x
- For 6 node grid: s subs, p pubs
- Each node: s/2 subs, p/3 pubs
- Scenario I: total data = 6*data(s/2) = 3s
  total work = 6*work(s/2, p/3) = 6(p/3)= 2p (case (ii))
- Scenario II: total data = 3s (same as before)
  total work = 6*work(s/2, p/3) = 6(s/2)(p/3) = sp (case (i))
- Compare to single node scenario: total data = s
  total work = p (case (ii)) or sp (case (i))

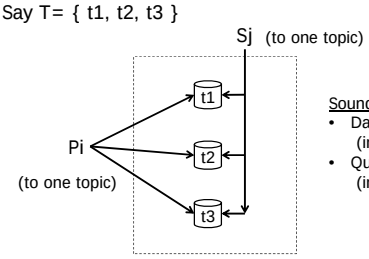
Topic Distributed Matching

Say T= { t1, t2, t3 }



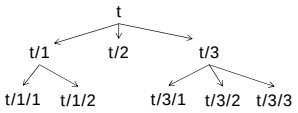
Topic Distributed Matching

Say T= { t1, t2, t3 }

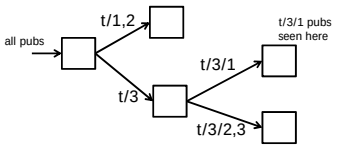


Sound familiar?
- Data fragmentation (in this case subs)
- Query localization (in this case pubs)

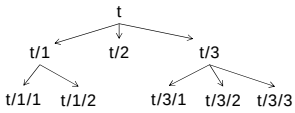
Matching with Topic Hierarchy



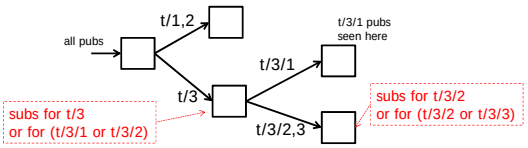
Publication dissemination tree:



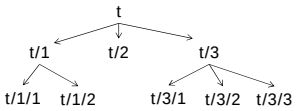
Matching with Topic Hierarchy



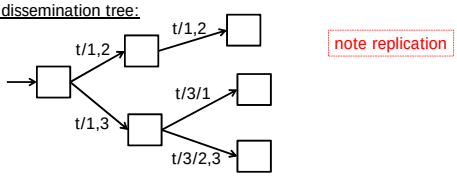
Publication dissemination tree:



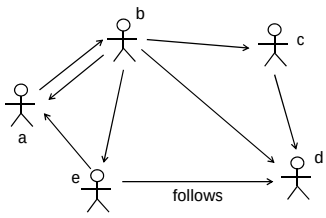
NetNews worked like this



Publication dissemination tree:

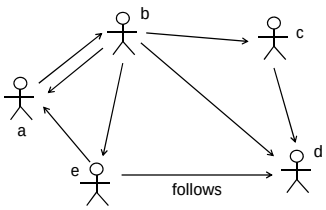


Discussion: Twitter



S(e) = {a,d}
pubs by e have description "e"
body of pub is 140 char max
users periodically check for notifications

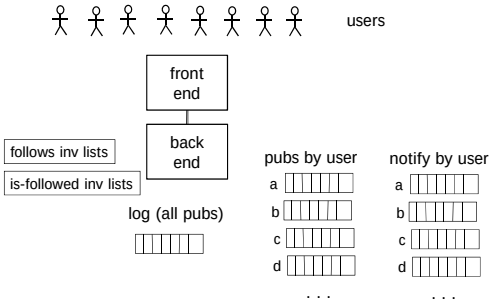
Discussion: Twitter



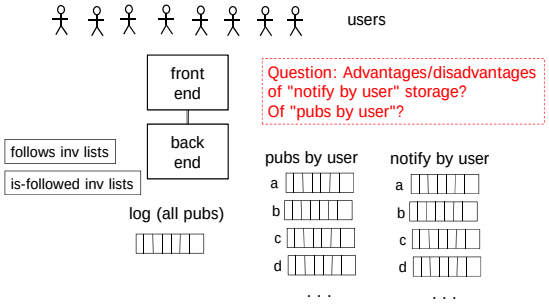
follows inv lists:
S(a): b
S(b): a,c,d,e
S(c): d
S(d): -
S(e): a, d

is-followed inv lists:
S^{-1}(a): b,e
S^{-1}(b): a
S^{-1}(c): b
S^{-1}(d): b,c,e
S^{-1}(e): b

Strawman Architecture- Centralized

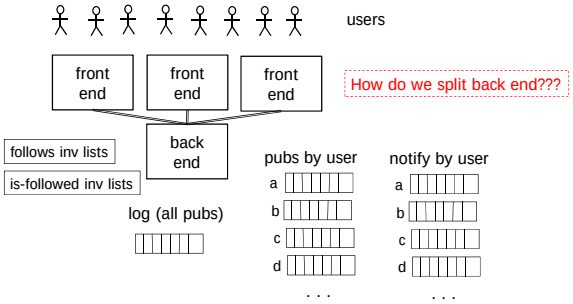


Strawman Architecture- Centralized



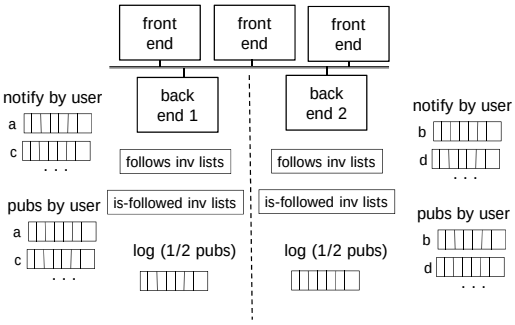
Question: Advantages/disadvantages of "notify by user" storage? Of "pubs by user"?

Strawman Architecture- Distributed

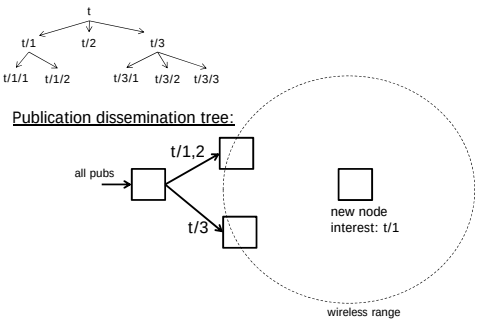


How do we split back end???

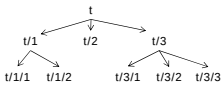
Strawman Architecture- Distributed



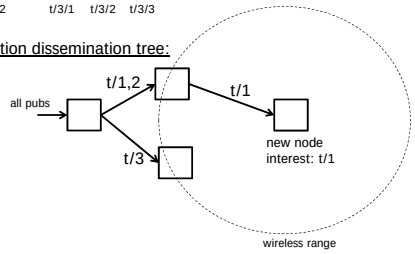
Dynamic Dissemination Tree



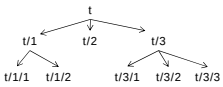
Dynamic Dissemination Tree



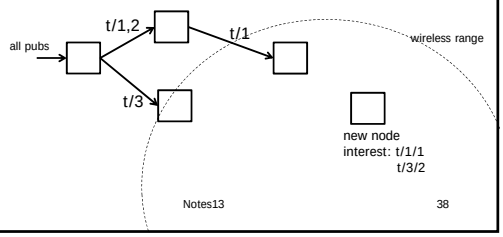
Publication dissemination tree:



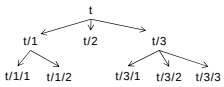
Dynamic Dissemination Tree



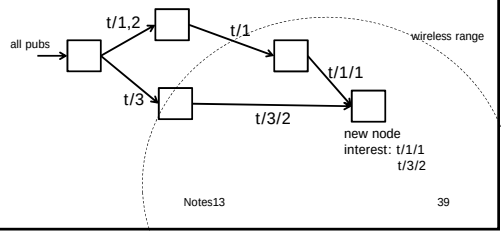
Publication dissemination tree:



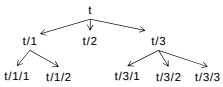
Dynamic Dissemination Tree



Publication dissemination tree:

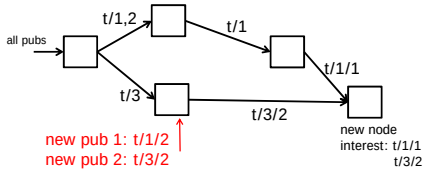


Dynamic Dissemination Tree



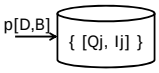
dissemination nodes can also publish:

Publication dissemination tree:



Matching at One Node

- Set { [Qj, Ij] } of stored subscriptions
- Match one publication p from stream



- Index of queries Qj different from index of descriptions (content)

Example

Subscriptions:

s1	(a,b)
s2	(a,d)
s3	(a,d,e)
s4	(b,f)
s5	(c,d,e,f)

Inverted Lists:

a	s1	s2	s3
b	s1	s4	
c	s5		
d	s2	s3	s5
e	s3	s5	
f	s4	s5	

sample publication
a c a f b c

- Match semantics for example: s1=(a,b) matches pub if both a and b appear in pub
- Can generalize, e.g., sj = a ∧ (b ∨ c) handle as two subs, sj1= (a ∧ b), sj2 = (a ∧ c)

Example

Subscriptions:

s1	(a,b)
s2	(a,d)
s3	(a,d,e)
s4	(b,f)
s5	(c,d,e,f)

Inverted Lists:

a	s1	s2	s3
b	s1	s4	
c	s5		
d	s2	s3	s5
e	s3	s5	
f	s4	s5	

sample publication

a c a f b c

- Intersection of lists (=null) not useful
- Union of lists {s1,s2,s3,s4,s5} gives candidate subscriptions
- Need to check each candidate (e.g., s1 matches, s2 does not)

Counting Method

Subscriptions:

s1	(a,b)
s2	(a,d)
s3	(a,d,e)
s4	(b,f)
s5	(c,d,e,f)

Inverted Lists:

a	s1	s2	s3
b	s1	s4	
c	s5		
d	s2	s3	s5
e	s3	s5	
f	s4	s5	

	total	count
s1	2	0
s2	2	0
s3	3	0
s4	2	0
s5	4	0

sample publication

a c a f b c

distinct word set

a b c f

Counting Method

Subscriptions:

s1	(a,b)
s2	(a,d)
s3	(a,d,e)
s4	(b,f)
s5	(c,d,e,f)

Inverted Lists:

a	s1	s2	s3
b	s1	s4	
c	s5		
d	s2	s3	s5
e	s3	s5	
f	s4	s5	

	total	count
s1	2	0
s2	2	0
s3	3	0
s4	2	0
s5	4	0

sample publication

a c a f b c

distinct word set

a b c f

- As we union lists, count number of times each sub appears
- If count ≥ total, then sub matches

Key Method

Subscriptions:

s1	(a,b)
s2	(a,d)
s3	(a,d,e)
s4	(b,f)
s5	(c,d,e,f)

Inverted Lists:

a	s1 [1, (b)]	s2 [1, (d)]	s3 [2, (d,e)]
b	s4 [1, (f)]		
c	s5 [3, (d,e,f)]		
d	null		
e	null		
f	null		

sample publication

a c a f b c

distinct word set

a b c f

occurrence table

a
b
c
f

Key Method

Subscriptions:

s1	(a,b)
s2	(a,d)
s3	(a,d,e)
s4	(b,f)
s5	(c,d,e,f)

Inverted Lists:

a	s1 [1, (b)]	s2 [1, (d)]	s3 [2, (d,e)]
b	s4 [1, (f)]		
c	s5 [3, (d,e,f)]		
d	null		
e	null		
f	null		

sample publication

a c a f b c

distinct word set

a b c f

occurrence table

a
b
c
f

- sub in only one inverted list
- each IL entry contains other terms in sub
- occurrence table is for fast hash lookup

Summary: Publish/Subscribe

- P/S semantics
- Various query/description models
- Distributed matching
- Matching at one node