# Complex queries in distributed publish-subscribe systems

Ashwin R. Bharambe,
Justin Weisz and
Srinivasan Seshan

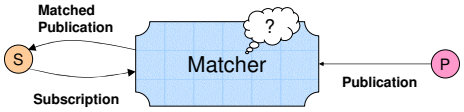


## Outline

- Publish-subscribe systems
- MERCURY architecture
- Preliminary evaluation
  - Scalability
  - Performance
- Future work



## Publish-subscribe systems




- Challenges
  - Subscription language - how to express "interests"?
  - Routing mechanism - how is content routed and where is it matched?



## Subscription language

- How to express interests?
  - Channels or Subjects
  - All content attributes
- Operators?
  - Exact matches
  - Range queries
  - Regular expressions

Subject = MATH

Name = 'A' & Age < 25

Price = 350$

300$ ≤ Price ≤ 350$

Name ~= Ash[win]*B*



## MERCURY subscription language

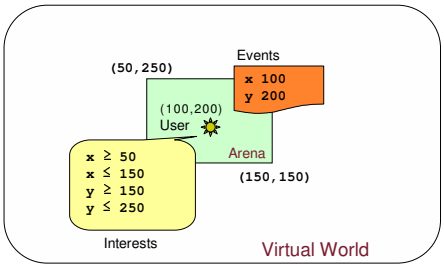
- Subscription =
  `list of` (type, attribute, rel-op, value)
- Can implement boolean expressions (AND/ORs)

( int, price, LESS_THAN, 300 )
( string, name, EQUALS, "Opensig" )

- Publication =
  `list of` (type, attribute, value)



## Example: Virtual reality



## Routing mechanism

- Centralized?
  - Easy to make publications "meet" subscriptions
  - Single point of failure – not robust!
- Distributed?
  - Where are subscriptions stored?
  - How do publications "meet" subscriptions?
  - Broadcast-based solutions not scalable
  - Multicast groups



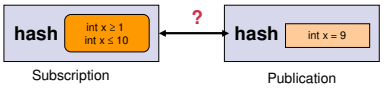
## Distributed routing - goals

- Scalability is a key goal
- Flooding **anything** is bad, bad, bad…
- System should not have hot-spot in terms of:
  - Computational load – matching
    - ⇒ Subscriptions should be evenly distributed
  - Number of packets routed or received
    - ⇒ Publications should be evenly distributed
- Yet – we should have low delivery delays!!



## Hashing

- Systems like Scribe use DHTs for scalability
- Why can't we ?
- Exact matches vs. Range queries!

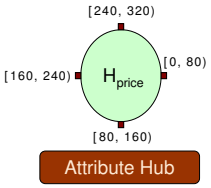



- How about generating 10 subscriptions ?
  - Too many subscriptions
  - Works for discrete-valued attributes only



## Attribute Hubs

- Divide range of an attribute into bins
- Each node responsible for range of attribute values



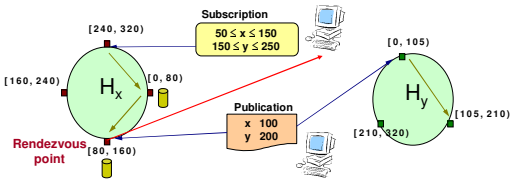

Attribute Hub

- Hub-nodes connected through a circular overlay
- Circle only for connectivity
- One hub per attribute
- Routing algorithm
  - compare value in content to my range



## Routing illustrated

- Send subscription to any one attribute hub
- Send publications to all attribute hubs

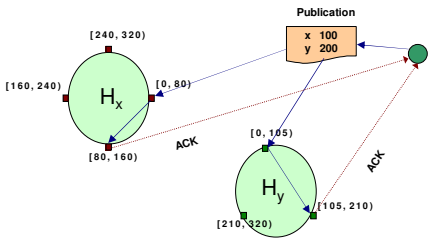





## Efficient routing

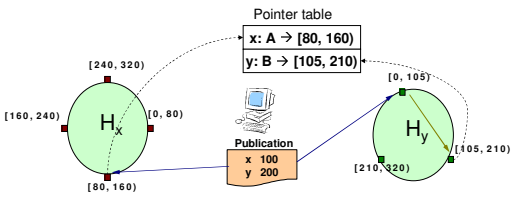
- Reduce number of hops
- Each hub-node maintains "small" number of pointers to distant parts of the hub
- How to maintain these pointers?
  - Send ACKs for publication receipts
  - Various caching policies determine the structure of the pointer table
    - e.g., LRU, Uniform-spacing, Exponential-spacing

## Routing illustrated






## Routing illustrated






## Evaluation

- Workload
- Experimental setup
- Metrics



## Workload

- One of our target apps → multi-player games
- Model
  - Virtual world as square
  - Subscriptions as rectangles around current positions



## Experimental setup

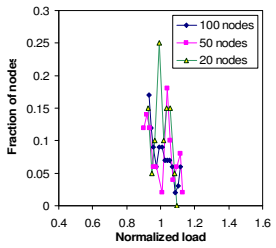
- Player movements simulated using mobility models from **`ns-2`**
- Two hubs – x and y co-ordinates
  - Half the nodes in each hub
  - Uniform partition of range



## Metrics

- Scalability metric → load
  - Number of publications routed by a node
  - Averaged over time
- Performance metric → publication delivery delay
  - Time between sending of a publication and its receipt by all subscribers
  - Averaged over all subscribers of a publication
  - Averaged over all publications

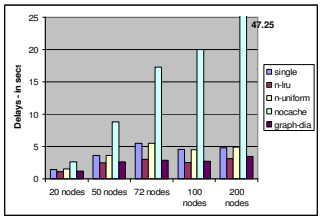
## Results: scalability




- Ideal graph: delta function
- Observed variation: ±12%



## Results: performance




Cache size: log(n)

- Without caching: linear scaling
- Caching reduces delays to near optimal
- Workload effects ?



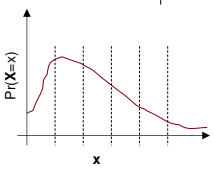
## Conclusions

- Expressive subscription language
- Decentralized architecture
- Scalability
  - Avoids flooding of subscriptions and publications – reduces network traffic
  - Distributes publications and subscriptions throughout the network – prevents swamping



## Future Work

- Load balancing
  - Sensitive to data value distribution
  - Adapt ranges dynamically according to the distribution
- Affects pointer management, caching, etc.






## Future Work

- Perform sensitivity analysis for different kinds of workloads
- Generic API for building applications on top of MERCURY
  - To be released soon
- Build a full-fledged distributed Quake-II