Content-Based Overlays

Ken Birman

Cornell University. CS5410 Fall 2008.

Content filtering

- Two kinds of publish-subscribe
- **Topic-based:** A *topic* defines the group of receivers.
  - Some systems allow you to subscribe to a pattern that matches sets of topics, by having a special "topics" meta-topic, but this is still topic-oriented
  - For scaling, typically must *map* topics to a smaller set of multicast groups or overlays
- **Content-based:** A *query* determines the messages that each receiver will accept
  - Can implement in a database or in an overlay

Challenges…

- Each approach has substantial challenges
  - For topic-based systems, the "channelization" problem (mapping many topics to a small number of multicast channels or overlays) is very hard
    - In the most general cases, channelization is NP-complete!
    - Yet some form of channelization may be critical because few multicast mechanisms scale well if huge numbers of groups are needed
  - Today we won't look closely at the channelization problem, but may revisit it later if time permits
    - Under some conditions, may be solvable

Challenges…

- What about content-based solutions?
  - We need to ask how to express queries "on content"
    - Could use Xquery, the new XML query language
    - Or could define a special-purpose packet inspection solution, a so-called "deep packet inspector"
  - Then would ideally want to build a smart overlay
    - Any given packet routes towards its destinations…
    - … and any given router optimizes so that it doesn't have an amount of work proportional to the number of pending content queries

Scenarios

- When would content routing be helpful?
  - In cloud systems, often want to route a request to some system that processed prior work of a related nature
  - For example, if I interact with Premier Cru to purchase 2007 Rhone red wines, as I query their data center it could build up a cache of data. If my queries revisit the same nodes, they perform far better
- In (unpublished) work at Amazon.com, the company found that almost *every* service has “opinions” about how to route messages within service clusters!

Scenarios

- What about out in the wild?
  - Here, imagine using content filtering as a way to query huge sets of RSS feeds
  - User expresses "interests" and these map to content queries… which route exactly the right stuff to him/her
- IBM Gryphon project: used this model, assumed that clients would be corporate users (often stock traders)
- Siena: similar model but assumes more of a P2P community in the Internet WAN

Things known about settings?

- All of these settings are very different
  - Amazon’s world is dominated by machine-controlled layout algorithms that selectively place services on clusters. Produces all sorts of “regularities”
    - E.g. clones of aservice often subscribe to the same data
    - And if A_0 and B_0 are collocated on node X, probably representatives of A and B will always be collocated
  - IBM’s world is dominated by heavy-tailed interest behaviors: Traders specialize in various ways
  - Siena world is more like a web search stream

Examples of issues raised

- Early work on IBM's Gryphon platform focused on in-network aggregation of the queries
  - They assumed that each message has an associated set of tags (attached by sender for efficiency)
  - Subscription was a predicate over these tags
  - Their focus was on combining the predicates, in the network, to avoid redundant work
- They got good results and even sold Gryphon as a product. But…

Thought question

- How often would you "expect" to have an opportunity to do in-network query combinations?
- Would you prefer to do an in-network solution, like Gryphon, or build a database solution like Cornell's Cayuga, where events can also be stored?

… and the answer is

- For IBM’s corporate clients, there turned out to most often be just a single Gryphon router per data center, with WAN links between them
  - In effect: Broadcast every event to all data centers
  - Then filter at the last hop before delivery to client nodes
  - Turns out that the router was fast enough for this model
- So all that in-network query combination work was unneeded in most client settings!

… and the rest of the answer?

- The majority of users had some form of archival storage unit in each data center
  - It subscribes to everything and keeps copies
  - So in effect, the average user "turned Gryphon into something much like Cayuga"
- Given this insight, Cayuga assumes full broadcast for event streams, focuses on a database model with rapid update rates. A more natural solution…

What about Amazon?

- Amazon has *lots* of packet-inspection routers that peek inside data quickly and forward as appropriate
  - Customized on a per-service basis
  - Many packet formats… hence little commonality between these inspection “applets”

- Motivates Cornell’s current work on “featherweight processes” to inspect packets at line speeds and exploit properties of multicore machines for scalability

Taking us to… Siena

- Relatively popular
  - Claimed user community of a few hundred thousand downloads
  - Perhaps a few thousand of whom actually use the system
- Little known about the actual users
- Today we'll look at a slide set generously provided by the development team

Remainder of today’s talk

- We’ll dive down to look closely at Siena
- Covering all three scenarios is just more than we have time to do

Siena