A Decentralized Content-based Aggregation Service for Pervasive Environments

Nanyan Jiang, Cristina Schmidt, Manish Parashar
The Applied Software Systems Laboratory
Rutgers, The State University of New Jersey
Piscataway, NJ USA

ICPS 2006, Lyon, France
Outline

- Motivation, requirements, challenges
- Meteor: Content-based interaction/messaging middleware
- Content-based decentralized aggregation service
- Deployment and evaluation
- Related work
- Conclusion
Motivation

- Ubiquity of devices with embedded computing and communications capabilities
- Emergence of pervasive Grid environments and applications
  - Realize information-driven, context-aware, computationally intensive end-to-end distributed applications
- System, application, information uncertainty!
- Programming abstractions and middleware services
  - Scalable, self-managing, adaptive
  - Based on content
  - Asynchronous and decoupled (opportunistic) interactions
  - Efficient and flexible information access, aggregation, assimilation
Data Aggregation and Assimilation in Pervasive Grid Environments

- Large volumes of heterogeneous and continuous data streams
  - Centralized, batch-processing solutions not feasible

- Dynamic system behavior
  - Availability, quality of data

- Dynamic application requirements
Project Meteor: Middleware Stack

- A self-organizing tiered overlay network
  - Bottom tier is a structured overlay, e.g., Chord, CAN, etc.
- Content-based routing engine (Squid)
  - Flexible content-based routing and querying with guarantees and bounded costs
  - Decentralized information discovery
- Associative Rendezvous Messaging
  - Symmetric post programming primitive
  - Content-based decoupled interaction with programmable reactive behavior
  - Decentralized content-based in-network aggregation service
Associative Rendezvous (AR)

- Content-based decoupled interactions:
  - All interactions are based on content, rather than names or addresses.
  - The participants (e.g. senders and receivers) communicate through an intermediary, the rendezvous point.
  - The communication is asynchronous. The participants can be decoupled both in space and time.

- Programmable reactive behaviors:
  - The reactive behaviors at the rendezvous points are encapsulated within messages => flexibility, expressivity, and multiple interaction semantics.
The Semantics of Associative Rendezvous Interactions

- **Messages:**
  - (header, action, data)
  - Symmetric post primitive: does not differentiating between interest/data

- **Associative selection**
  - match between interest and data profiles

- **Reactive behavior**
  - Execute action field upon matching

Profile = list of (attribute, value) pairs:
Example:
<(sensor_type, temperature), (latitude, 10), (longitude, 20)>

```
post (<p1, p2>, store, data)

(1) C1

match

(2) post(<p1, *>, notify_data(C2))

notify_data(C2)

(3) C2
```
Content-based Aggregation Services

- Extends the AR programming abstraction to enable content-based aggregation services
  - AR post(<L, speed, 3600>, retrieve(AVERAGE, 300))
  - Region L: (35-60, 40-80)
  - find the average speed in the stretch of road specified by region L every 5 minutes for the next 1 hour

Compute the traffic bottleneck along Route 95 from East Brunswick to NYC in the next one hour

1. Anyone exceed speed limit
2. Where is the maximum speed on road
3. Minimum speed on road and where

Estimate the time needed to enter NYC
Self-organizing Overlay (Chord)

- A self-organizing P2P ring overlay
- Nodes and data have unique identifiers (keys), from a circular key space (0 to $2^m$)
- Each node maintains a routing table, called “finger table”
- A key is stored at the first node whose identifier is greater or equal with the key
- The request is routed to the neighbor node closer to the destination
- Routes in $O(\log n)$ hops

**Finger** = the successor of (this node id + $2^{i-1}$) mod $2^m$, $0 \leq i \leq m$

Routing from node 1 to node 6
SquidTON: Reliability and Fault Tolerance

- Pervasive Grid systems are dynamic, with nodes joining, leaving and failing relatively often
  - => data loss and temporarily inconsistent overlay structure
  - => the system cannot offer guarantees
- Build redundancy into the overlay network
- Replicate the data

SquidTON = Squid Two-tier Overlay Network
- Consecutive nodes form unstructured groups, and at the same time are connected by a global structured overlay (e.g. Chord)
- Data is replicated in the group
Content-based Routing (Squid)

- Route based on semantic headers
  - Use a dimension-reducing mappings to map from multidimensional information space => 1-dimensional index space
    - can route based on keywords, partial keywords, wildcards and ranges
  - Builds on an underlying DHT lookup mechanism
- Offers guarantees: all destinations matched by the content will be identified with bounded costs (messages, intermediate nodes, etc.)
Squid – Definitions

- **Keyword tuple (used to specify the destination)**
  - List of \( d \) keywords, wildcards and/or ranges
  - Example:
    - \((\text{temperature, celsius}), (\text{temp}^*, *), (*, 10, 20-25)\)

- **Simple keyword tuple**
  - Contains only complete keywords

- **Complex keyword tuple**
  - Contains wildcards and/or ranges
Content Indexing: Hilbert SFC

- \( f: \mathbb{N}^d \rightarrow \mathbb{N} \), recursive generation

- Properties:
  - Digital causality
  - Locality preserving
  - Clustering

**Cluster:** group of cells connected by a segment of the curve
Content Indexing and Routing

Content profile 1:
(2,1)

Generate the clusters associated with the content profile

Content profile 2:
(4-7,0-3)

Content profile 3:
(*, 4)

Rout to the nodes that store the clusters

Matching messages

Send the results to the requesting node

Content profile 1

Content profile 2

Content profile 3

Matching messages

Send the results to the requesting node
Squid: Routing Optimization

- More than one cluster are typically stored at a node.
- Not all clusters that are generated for a query exist in the network => optimize!
- SFC generation recursive => clusters generation is recursive => the process of cluster generation can be viewed as a prefix tree (trie).
- Optimization: embed the tree into the overlay, and prune nodes during the construction phase.
Decentralized In-network Aggregation

- **Post AR messages** with *aggregation* behaviors in the system
  - Content indexing: from n-dimension to 1-dimension using Hilbert SFC
  - Content-based routing: prefix trie construction
  - Decentralized in-network aggregation
    - Use aggregation tries for back-propagating and aggregating matching data items

Indexing using SFC

Post \(<p_1, p_2, \ldots, p_N>,\) action (aggrBehavior)) \(\rightarrow\) Area in the N-dimensional semantic space \(\rightarrow\) Segments in the 1-dimensional index space

Trie-based In-network Aggregation \(\leftarrow\) Matched data items \(\leftarrow\) Trie-based Content Routing

Back-propagation of matched data \hspace{1cm} Trie construction
Trie Construction

- Recursive resolution of message profile to construct prefix trie
Trie-based Content Routing

- Embed the tree into overlay network
- Node 100001 is responsible for storing the subtree routed at 011*
Trie-based In-Network Aggregation (I)

- AR message was issued at node 111000
- First recursion level results in clusters with prefix 0
- At node 000000, further level recursion, the cluster 011 and 0010 are identified and padded with 0, for example;
- Node receive the sub-queries generate next level recursion, …
Trie-based In-Network aggregation (II)

- Caching to improve latencies
- Managing system dynamics
  - Joins, Leaves, Fails
- Load balancing
- Building geographical awareness into the overlay
Miscellaneous Issues and Optimizations

- Varied Link Latencies
- Caching to reduce routing latencies
- Load balancing
- Addressing churn
  - Join, leave
- Failures
  - Leaf peers, intermediate peers, source peer
- Geographical locality (to an extent)
Meteor: Implementation Overview

- Current implementation builds on JXTA
- Chord, Squid and AR Messaging layer are implemented as event-driven JXTA services
- Deployments
  - Local area network
    - 64 1.6 GHz Pentium IV nodes with 100 Mbps Ethernet interconnection network
  - Orbit sensor network testbed
    - 400 802.11 radio nodes with 802.11 wireless connections
  - PlanetLab wide-area testbed
Experimental Evaluation

- Aggregation services:
  - Complex queries issued from each of the three deployment environments
    - Post(34-444,temp*, retrieve(count))
  - Effectiveness of in-network aggregation
    - Aggregate range queries
    - Number of messages processed in the network
  - Robustness in case of single peer failures

- Note that the performance of Squid and AR have been separately evaluated
Scalability of the Aggregation Services

Aggregate query:
post(<(100-300, 75-125), temperature>, retrieve(avg, 300))
Effectiveness of In-Network Aggregation

- Simulation results
  - 500 nodes
- AR aggregate message:
  - \text{Post(<100-230, 35-120>, retrieve(avg))}
- Fig (a) number of messages with/without in-network aggregation
- Fig (b) actual number of messages per peer
Robustness of Aggregation Service

- Assume single node failure
  - An intermediate node fails at about 205 time ticks
  - On-going aggregation operations are lost at this node
- About 100 ms required to correct the aggregation trie in a LAN environment
Related works

- Aggregation in P2P environments
  - Astrolabe, Cone, PHT
- Aggregation in sensor networks
  - Cougar, TAG
- Meteor
  - Content-based aggregation abstraction
    - Asynchronous and decoupled
  - No persistent state maintained in the network
  - Trie reused across aggregations
  - Guarantees
Summary

- Content-based, decoupled interactions and aggregations are essential for pervasive Grid applications.
- Meteor provides unified programming abstractions for content-based decoupled interactions and decentralized in-network aggregation:
  - Recurrent and spatially constrained queries
  - Scalable and efficient trie-based design and implementation
- Deployments on LAN, Orbit and PlanetLab:
  - Scalability, performance, robustness