Content-based Middleware for Sensor-based Pervasive Environments

Cristina Schmidt, Vincent Matossian, Nanyan Jiang and Manish Parashar
The Applied Software Systems Laboratory
Rutgers University
Overview

- Meteor – a content-based middleware infrastructure that supports decoupled interactions
- Squid – content-based routing infrastructure
- Associative Rendezvous (AR) paradigm for content-based decoupled interactions for pervasive applications
- Cascading Local Behaviors (CLB)
- Implementation and evaluation
Motivation

- Growing ubiquity of sensor/actuator devices with embedded communication and computation capabilities
- The emergence of pervasive applications
  - Sensors, actuators, services, resources – interact and collaborate to satisfy goals
  - Examples:
    - Fire management applications
    - Emergency medical care
Motivation (cont’d)

- Pervasive applications:
  - Manage, adapt, optimize themselves to meet their objectives
  - Large, distributed, complex, heterogeneous, dynamic

=> They need a middleware infrastructure that:
  - Scalable and self-managing
  - Based on content rather than names/addresses
  - Supports asynchronous and decoupled interactions
  - Provides some interaction guarantees
Meteor - Content-based Middleware

- Self-organizing overlay
  - Chord P2P overlay, each peer is an *rendezvous point*
- Content-based routing
  - Squid
- AR Messaging
  - Profile Manager
  - Matching engine
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
Content-based Routing (Squid)

- Chord can route based on unique data identifiers only
- Squid uses a dimension-reducing indexing scheme => can route based on keywords, partial keywords, wildcards and ranges
  - Uses the Space-Filling Curves as the mapping
- Squid offers guarantees: all destinations matched by the keywords will be identified
Hilbert Space-Filling Curve (SFC)

- **f**: \( N^d \rightarrow N \), recursive generation

- **Properties:**
  - Digital causality
  - Locality preserving
  - Clustering
Keyword tuple (used to specify the destination)
- List of $d$ keywords, wildcards and/or ranges
- Example:
  - (temperature, celsius), (temp*, *), (*, 10, 20-25)

Simple keyword tuple
- Contains only complete keywords

Complex keyword tuple
- Contains wildcards and/or ranges
Squid – Content-based Routing (1)

- Using simple keyword tuples
  - Routing to a single destination – equivalent to Chord lookup

Map the point (2, 1) to index 7 using the Hilbert Space Filling Curve (SFC).

Route to node 13 (the successor of the index 7)
Squid – Content-based Routing (2)

- Using complex keyword tuples
  - Routing to multiple destinations – the straightforward solution

Translate the query to relevant clusters on the SFC-based index space
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 tree
- Optimization: embed the tree into the overlay, and prune nodes during the construction phase
Squid: Routing Optimization - Example

Binary complex keyword tuple (011, *)
Squid: Experimental Evaluation (1)

- Simulation
- Up to 5000 nodes, and up to $10^6$ keyword tuples
- Metrics:
  - Number of processing nodes
  - Number of data nodes
- Keyword tuples:
  - Keyword tuples with wildcards
  - Keyword tuples with ranges
- System size increases from 1000 to 5000 nodes, keys from $2*10^5$ to $10^6$
Squid: Experimental Evaluation (2)
# Squid vs. Existing P2P Systems

<table>
<thead>
<tr>
<th>P2P System</th>
<th>Pros &amp; Cons</th>
<th>Examples</th>
</tr>
</thead>
</table>
| Unstructured            | **Pro:** overlay easy to maintain, supports complex queries  
**Con:** no search or cost guarantees                                                                                                             | Gnutella – like systems       |
| Hybrid                  | **Pro:** supports complex queries  
**Con:** not scalable                                                                                                                           | Napster, Morpheus             |
| Data-lookup             | **Pro:** efficient lookup with guarantees  
**Con:** - complex queries not supported  
- high structure overlay maintenance cost                                                                                                     | Chord, CAN, Tapestry, Pastry  |
| Structured Keyword Search | **Pro:** supports more complex queries (e.g. keywords, SFC-based systems support even partial keywords, wildcards and ranges)  
**Con:** high structure overlay maintenance cost                                                                                           | Inverted Indices, PeerSearch, SFC-based systems (HP Labs, Squid) |
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
Associative Rendezvous: Interaction Model

- **Messages:**
  - (header, action, data)
  - Symmetric post primitive

- **Associative selection**

- **Reactive behavior**

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

![Diagram of AR model showing Profile Manager, Matching Engine, and Action Dispatcher with the flow of AR message, profile, action, and data.](image-url)
**Associative Rendezvous: Example**

1. post(⟨p₁, p₂⟩, notify_interest(C₁))
2. notify_interest(C₁)
3. notify_interest(C₁)
4. post(⟨p₁, p₂⟩, store, data)
5. notify(C₂)
6. post(⟨p₁, *⟩, retrieve(C₂))
7. retrieve(C₂, data)
8. post(⟨p₁, *⟩, delete_interest(C₂))
9. post(⟨p₁, p₂⟩, delete_data(C₁))

Meteor middleware: P2P system, each peer is a *rendezvous point*, and runs the Meteor stack.

---

*The Applied Systems Laboratory*
Cascading Local Behaviors (CLB)

- An abstraction for content-based decoupled interaction with programmable reactive behaviors

- A programming model, where the behaviors of individual application elements are locally defined in terms of local state, and context and content events, and result in data and interest messages being produced.

- Application flows emerge as a consequence of the cascading effect of local behaviors, without having to be explicitly programmed.
Cascading Local Behaviors – Example

Temperature Sensor

if (temp > 80) then publish temp

Thermostat

if (temp > 85)
then temp_control = on;
publish temp_control

Window Actuator

if (temp_control == on) then close windows

if (temp > 85)
then publish temp

if (temp > 85)
then temp_control = on;
publish temp_control

if (temp_control = on)
then Close window
Current implementation builds on JXTA
Chord, Squid and AR Messaging layer are implemented as event-driven JXTA services
Each layer exposes an API for the upper layer
Chord interacts directly with JXTA, it uses:
  - JXTA discovery mechanism to find other nodes already in the group
  - JXTA resolver service to send messages between peers
Implementation Overview (cont’d)

- Squid uses Chord’s *lookup* primitive, to route clusters to their destination
- Squid’s API consists of one primitive:
  - post (AR_Message) – routes based on keywords extracted from the message’s profile
- AR Messaging Layer
  - API consists of a single function: post (header, action, data)
  - It currently uses a MySQL database as storage for profiles, and as a matching engine
Meteor: Experimental Evaluation (1)

- **Current Deployment**
  - 64 node Linux cluster, 1.6 GHz Pentium IV, each node running a peer (rendezvous node)
  - 100 Mbps Ethernet interconnection network

- **Experimental evaluation:**
  - Chord lookup overhead as a function of system size
  - Content-based routing overhead
  - Associative Messaging overhead at each RP node:
    - querying the database – associative selection
    - constructing the notification messages – reactive behavior
Meteor: Experimental Evaluation (2)

Overlay network lookup overhead (Chord)

Content-based routing overhead (Squid)
Meteor: Experimental Evaluation (3)

Matching overhead at a single RP

Number of profiles stored locally
Load Balancing (LB) – Why Necessary?

- The node identifiers (Chord) are uniformly distributed in the identifier space.
- The data to be stored in the system is not uniformly distributed in the $d$-dimensional space.
- The SFC preserves locality.

$\Rightarrow$ The SFC index is not uniformly distributed, and we need load balancing.
Load Balancing - Discussion

- Assume that all nodes have the same storage and computational capabilities
- The LB algorithms used balance the storage load, not the computational one
- Future directions:
  - Better LB algorithms, that take into consideration
    - The nodes’ heterogeneity
    - The “hot-spots” – some nodes will store more popular information, and they will have lots of requests
Load balancing – Algorithms

- Load balancing at node join:
  - generate more than one ID for the new node, send join requests in the network and join with the ID that places the node in the most crowded part of the network

- Load balancing at runtime:
  - run a local load balancing algorithm between neighbors (from time to time), and redistribute the load
  - use virtual nodes that can migrate to less loaded physical nodes
The distribution of the keys in the index space. The index space was partitioned into 5000 intervals. The Y-axis represents the number of keys per interval.

The distribution of the keys when using only the load balancing at node join technique.

The distribution of the keys when using both the load balancing at node join technique, and the local load balancing.
Conclusions

- Content-based, decoupled programming model is suited to address the challenges of pervasive applications.
- A P2P infrastructure is a natural solution to implement the Associative Rendezvous abstraction.
- JXTA is a convenient platform to develop decoupled, content-based middleware.
More Information

- Meteor project page:
  - [http://www.caip.rutgers.edu/TASSL/Projects/Meteor/](http://www.caip.rutgers.edu/TASSL/Projects/Meteor/)

- Squid project page:
  - [http://www.caip.rutgers.edu/~cristins/research.html/](http://www.caip.rutgers.edu/~cristins/research.html/)

- Recent report:
  - [http://www.jxta.org/universities/rutgers.html](http://www.jxta.org/universities/rutgers.html)

- Meteor JXTA implementation:
  - Vincent Matossian: [vincentm@caip.rutgers.edu](mailto:vincentm@caip.rutgers.edu)