

Distributed Scalable Content Discovery Based on Rendezvous Points

Jun Gao

Ph.D. Thesis Proposal

Computer Science Department
Carnegie Mellon University

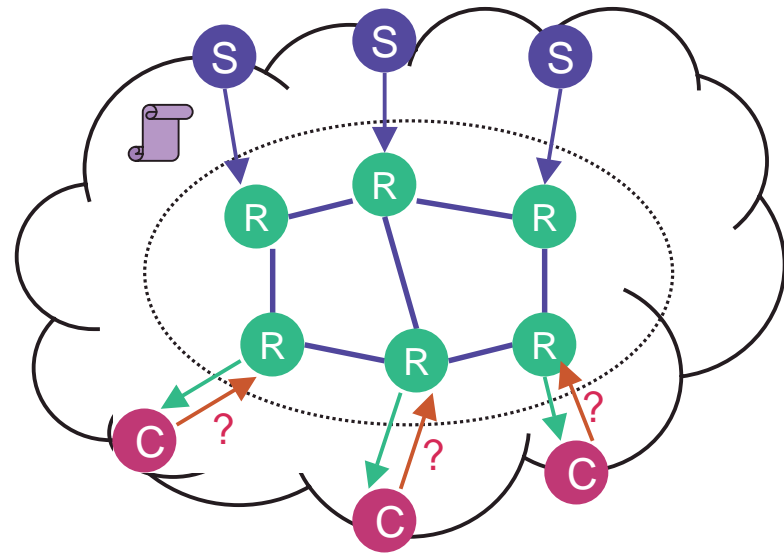
May 20th, 2002

Outline

- Content Discovery System (CDS)
- Thesis statement
- Related work
- Proposed CDS system
- Research plan
- Time line
- Expected contributions

Content Discovery System (CDS)

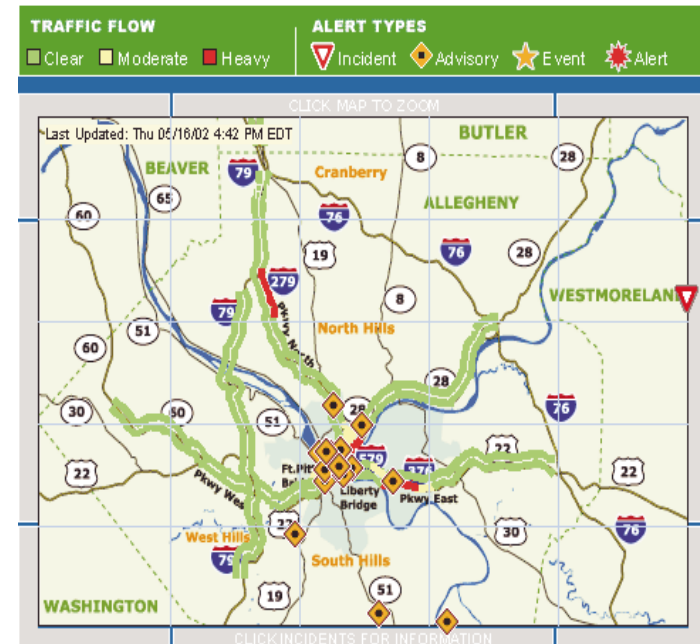
- Distributed system that allows the discovery of contents
 - Three logical entities
 - “content name” discovery
 - Broad definition of “content”
- Example CDS systems
 - Service discovery
 - Peer-to-peer object sharing
 - Pub/sub systems
- Separation of **content discovery** and **content delivery**



S: content providers (servers)
C: content consumers (clients)
R: content resolvers

Example: A Highway Monitoring Service

- Allows users to discover traffic status observed by cameras and sensors
 - What is the speed around Fort Pitt tunnel?
 - Are there any accidents on I-279?
 - What sections around Pittsburgh are congested?
- Characteristics of this service
 - Support large number of devices
 - Devices must update frequently
 - Support high query rate



Snapshot from: Traffic.com

Thesis Statement

In this thesis, I propose a distributed and scalable approach to content discovery that supports flexible and efficient search of dynamic contents.

CDS Properties

- Contents must be searchable
 - Find contents without knowing the exact names
 - Contents can be dynamic
 - Content names are not hierarchical
- Scalability
 - System performance remains as load increases
- Distributed and robust infrastructure
 - No centralized administration
- Generic software layer
 - Building block for high level applications

Related Work

- Existing systems have difficulties in achieving both scalability and rich functionality
- Centralized solution
 - Central resolver(s) stores all the contents
 - Supports flexible search
 - Load concentration at the central site
 - Single point-of-failure.
- Distributed solution
 - Graph-based schemes
 - Tree-based schemes
 - Hash-based schemes

Distributed Solutions

➤ Graph-based systems

- Resolvers organized into a general graph
 - ▶ Registration flooding scheme
 - ▶ Query broadcasting scheme
- Not scalable
- Robust infrastructure

➤ Tree-based systems

- Resolvers organized into a tree
- Scale well for hierarchical names
 - ▶ E.g., DNS
 - ▶ Hard to apply to non-hierarchical names
- Robustness concern
- Load concentration close to the root

Hash-based Lookup Systems

- Resolvers form an overlay network based on hashing
 - E.g., Chord, CAN, Pastry, Tapestry
- Provide a simple name lookup mechanism
 - Associating content names with resolver nodes
 - ▶ No flooding or broadcasting
- Do not support search
 - Clients must know the exact name of the content
- Our system utilizes the hash-based lookup algorithms

Proposed CDS system

- Basic system design
 - Naming scheme
 - Resolver network
 - Rendezvous Point (RP) based scheme
- System with load balancing
 - Load concentration problem
 - Load Balancing Matrices (LBM)

Attribute-Value Based Naming Scheme

- Content names and queries are represented with AV-pairs
 - Attributes may be dynamic
 - One attribute may depend on another attribute
- Searchable
 - Query is a subset of the matched name
 - $2^n - 1$ matched queries for a name that has n AV-pairs
- Example queries
 - find out the speed at I-279, exit 4, in Pittsburgh
 - find the highway sections in Pittsburgh that speed is 45mph

Service description (SD)

```
Camera number = 5562
Camera type = q-cam
Highway = I-279
    Exit = 4
City = pittsburgh
Speed = 45mph
Road condition = dry
```

Query 1:

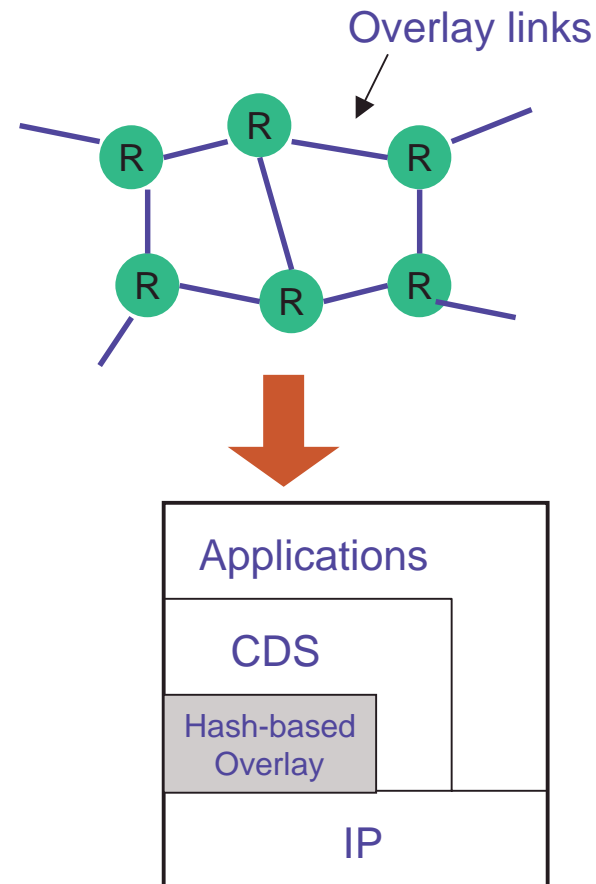
```
Highway = I-279
    Exit = 4
City = pittsburgh
```

Query 2:

```
City = pittsburgh
Speed = 45mph
```

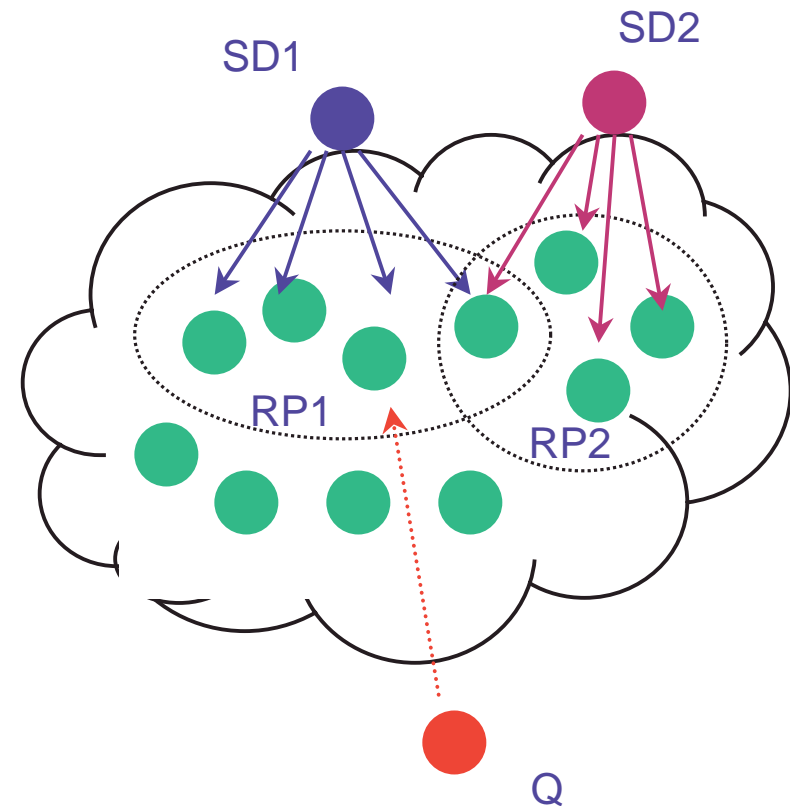
Hash-based Resolver Network

- Resolvers form a hash-based overlay network
 - Use Chord-like mechanisms
 - Node ID computed based on a hash function H
 - Node ID based forwarding within the overlay
 - ▶ Path length is $O(\log Nc)$
- CDS is decoupled from underlying overlay mechanism
 - We use this layer for content distribution and discovery



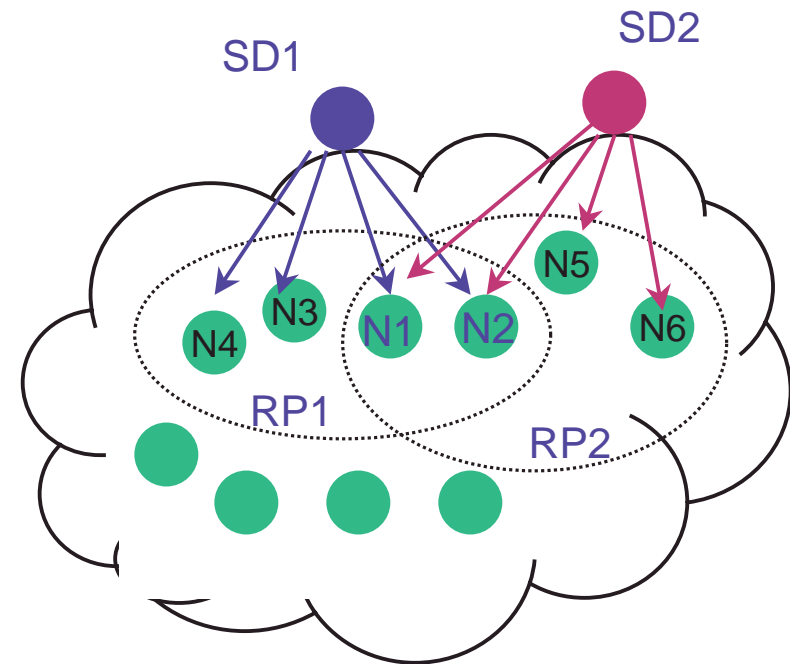
Rendezvous Point (RP) -based Approach

- Distribute each content name to a set of resolver nodes, known as RPs
 - Queries are sent to proper RPs for resolution
- Guidelines
 - The set should be small
 - Use different set for different names
 - Ensure that a name can be found by all possible matched queries



Registration with RP nodes

- Hash each AV-pair individually to get a RP node ID
 - Ensures correctness for queries
 - RP set size is n for a name with n AV-pairs
- Full name is sent to each node in the RP set
 - Replicated at n places
- Registration cost
 - $O(n)$ messages to n nodes



SD1: {a1=v1, a2=v2, a3=v3, a4=v4}

SD2: {a1=v1, a2=v2, a5=v5, a6=v6}

$H(a1=v1) = N1$, $H(a2=v2) = N2$

Resolver Node Database

- A node becomes the specialized resolver for the AV-pairs mapped onto it
 - Each node receives equal number of AV-pairs
 - ▶ $k = N_d / N_c$

- Size of the name database is determined by the number of names contain each of the k AV-pair

$$t = \sum_{i=1}^k N_{avi}$$

- Contain the complete AV-pair list for each name
 - Can resolve received query completely

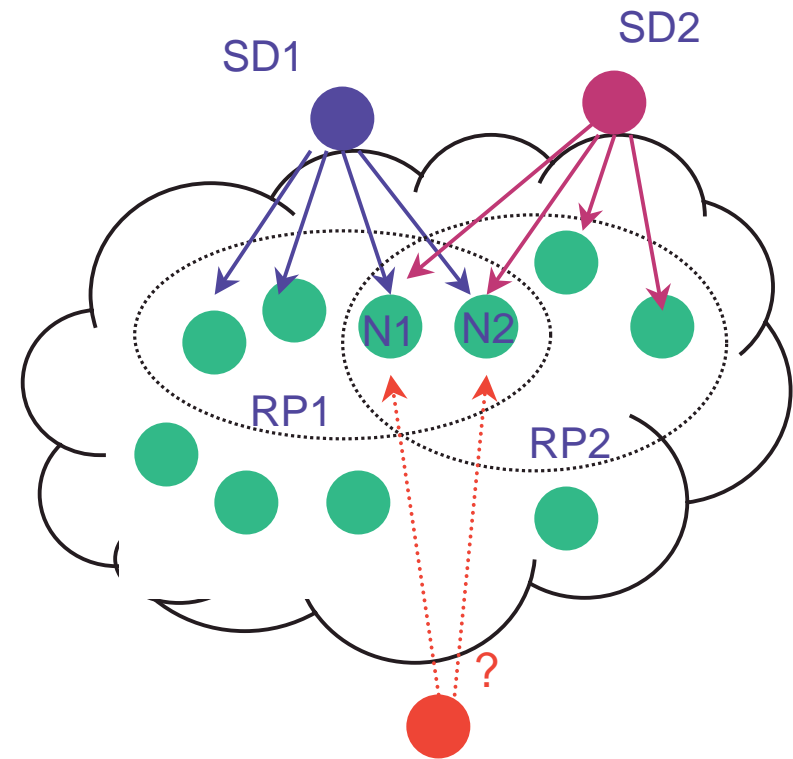
N_d : Number of different AV-pairs
 N_c : Number of Resolver nodes
 N_{avi} : Number of names that contain av_i

N1:
 (a1=v1)
 SD1: a1=v1, a2=v2, a3=v3, a4=v4
 SD2: a1=v1, a2=v2, a5=v5, a6=v6
 SD3: a1=v1, ...
 ...
 (a7=v7)
 ...

N2:
 (a2=v2)
 SD1: a2=v2, a1=v1, a3=v3, a4=v4
 SD2: a2=v2, a1=v1, a5=v5, a6=v6
 SD4: a2=v2, ...
 ...

Query Resolution

- Client applies the same hash function to m AV-pairs in the query to get the IDs of resolver nodes
 - Query can be resolved by any of these nodes
- Query optimization algorithm
 - Client selects a node that has the best performance
 - ▶ E.g., probe the database size on each node
- Query cost
 - $O(1)$ query message
 - $O(m)$ probe messages



$Q:\{a1=v1, a2=v2\}$

$H(a1=v1) = N1, H(a2=v2) = N2$

Load Concentration Problem

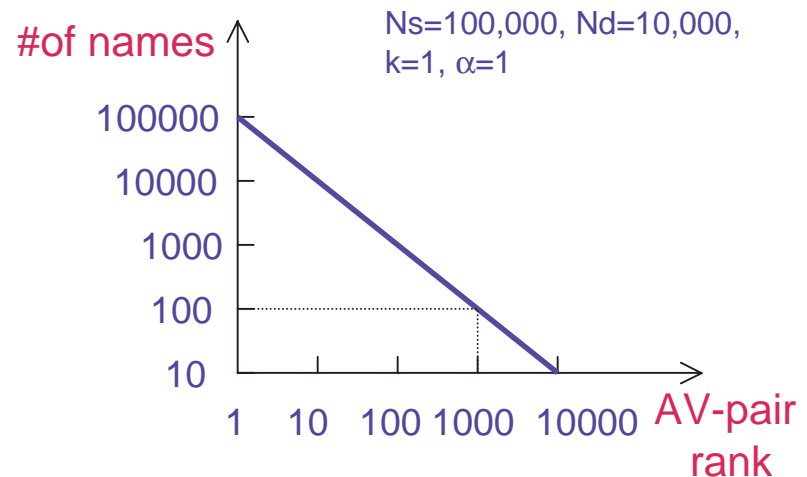
- Basic system performs well under balanced load
 - Registrations and queries processed efficiently
- However, one node may be overloaded before others
 - May receive more names than others
 - ▶ Corresponds to common AV-pairs in names
 - May be overloaded by registration messages
 - May be overloaded by query messages
 - ▶ Corresponds to popular AV-pairs in queries

Example: Zipf distribution of AV-pairs

- Observation: some AV-pairs are very popular, and many are uncommon
 - E.g. speed=45mph vs. speed=90mph
- Suppose the popularity distribution of AV-pairs in names follow a Zipf distribution
- Example:
 - 100,000 names have the most popular AV-pair
 - ▶ Will be mapped onto one node!
 - Each AV-pair ranked from 1000 to 10000 is contained in less than 100 names

$$N_{av_i} = N_s \cdot k \cdot \frac{1}{i^\alpha}$$

N_s : total number of names
 N_d : number of different AV-pairs
 i : AV-pair rank(from 1 to N_d)
 k : constant
 α : constant near 1



CDS with Load Balancing

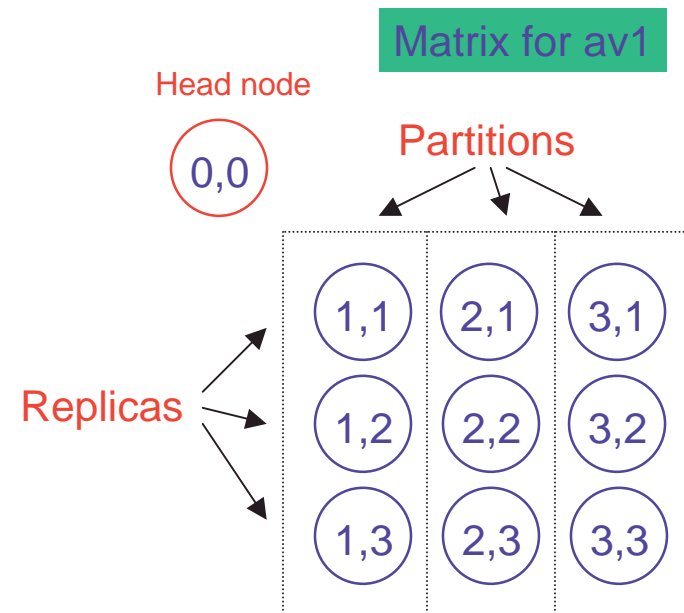
- Intuition
 - Use a set of nodes for a popular AV-pair
- Mechanisms
 - Partition when registration load reaches threshold
 - Replicate when query load reaches threshold
- Guideline
 - Must ensure registrations and queries can still find RP nodes efficiently

Thresholds maintained on each node

T_{SD} : Maximum number of content names can host
 T_{reg} : Maximum sustainable registration rate
 T_q : Maximum sustainable query rate

Load Balancing Matrix (LBM)

- Use a matrix of nodes to store all names that contain one AV-pair
 - RP Node → RP Matrix
- Columns are used to share registration load
- Rows are used to share query load
- Matrix expands and contracts automatically based on the current load
 - Self-adaptive
 - No centralized control



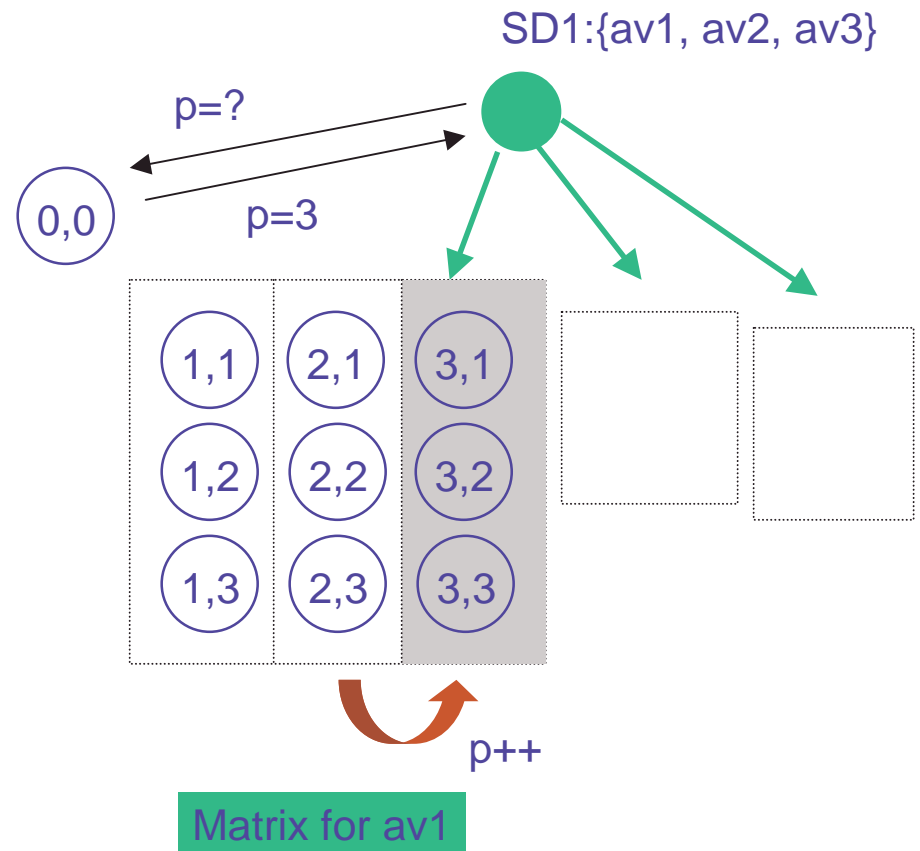
Nodes are indexed

$$N_1^{(p,r)} = H(\text{av1}, p, r)$$

Head node: $N_1^{(0,0)} = H(\text{av1}, 0, 0)$, stores the size of the matrix (p, r)

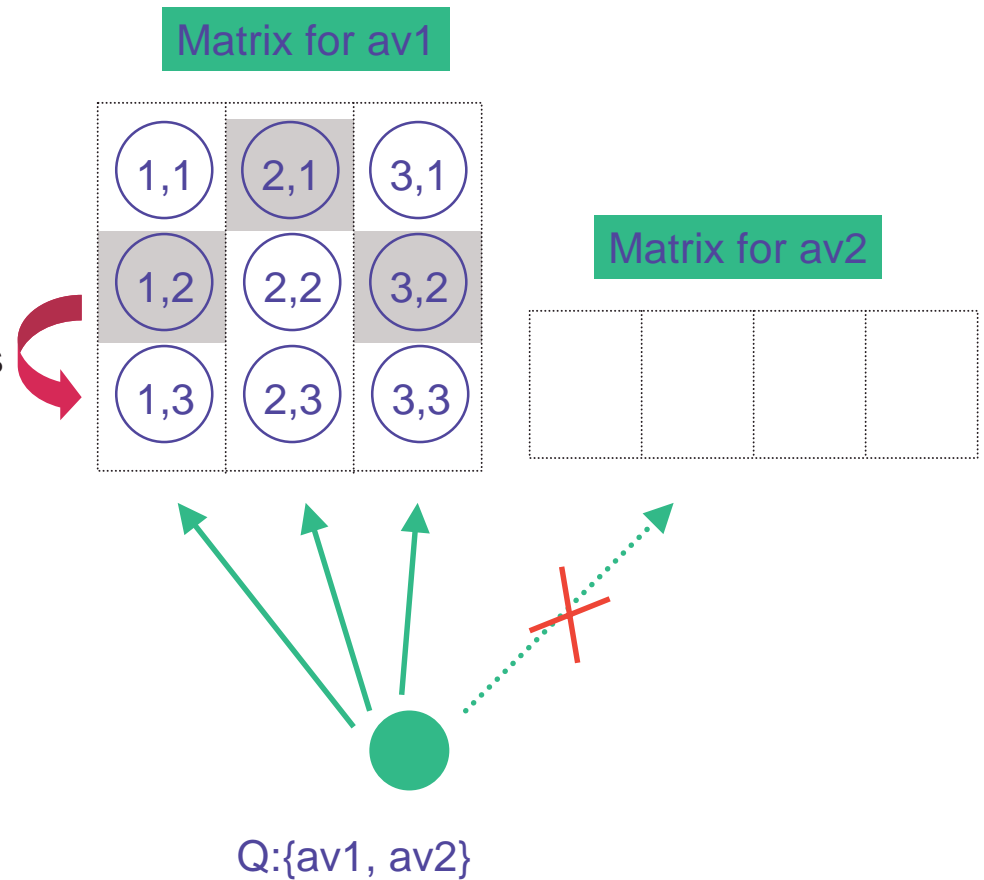
Registration

- New partitions are introduced when the last column reaches threshold
 - Increase the p value by 1
 - Accept new registrations
- Discover the matrix size (p, r) for each AV-pair
 - Retrieve from head node $N_1^{(0,0)}$
 - Binary search to discover
 - Use previously cached value
- Send registration to nodes in the last column
 - Replicas
- Each column is a subset of the names that contain $av1$



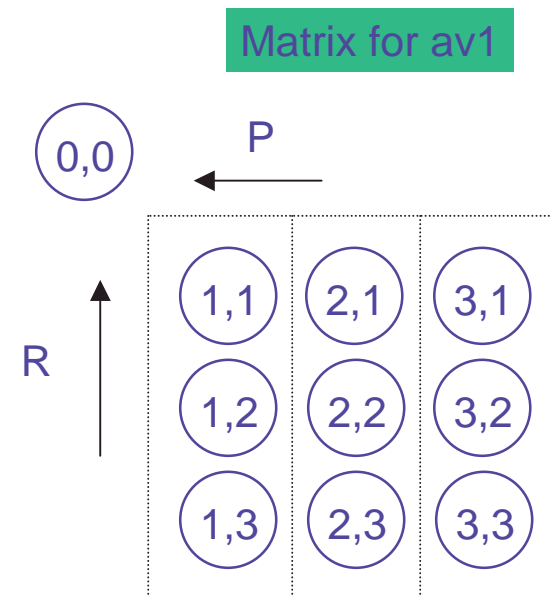
Query

- Select a matrix with the fewest columns
 - Small $p \rightarrow$ few partitions
- Sent to one node in each column
 - To get all the matched contents
- Within each column, sent to a random node
 - Distribute query load evenly
- New replicas are created when the query load on a node reaches threshold
 - Increase r value by 1
 - Duplicate its content at node $N_1(p, r+1)$
 - Future queries will be shared by $r+1$ nodes in the column



Matrix Compaction

- Smaller matrix is more efficient for registrations and queries
- Matrix compaction along P dimension
 - When earlier nodes in each row have available space
 - ▶ Push
 - ▶ Pull
 - Decrease p value by 1
- Matrix compaction along R dimension
 - When observed query rate goes below threshold
 - Decrease r value by 1
- Must maintain consistency



System Properties

- From a resolver node point of view
 - Load observed is upper bounded by thresholds
- From whole system point of view
 - Load is spread across all resolvers
 - System does not reject registrations or queries until all resolvers reach thresholds

- Registration cost for one AV-pair
 - $O(r_i)$ registration messages, where r_i is the number of rows in the LBM

$$r_i = \frac{Q_{avi}}{T_q}$$

- Query cost for one AV-pair
 - $O(p_i)$ query messages, where p_i is the number of columns in the LBM

$$p_i = \max\left(\frac{N_{avi}}{T_{SD}}, \frac{R_{avi}}{T_{reg}}\right)$$

Matrix Effects on Registration and Query

- Matrix grows as registration and query load increase
 - Number of resolver nodes in one matrix
 - ▶ $m_i = r_i p_i$
- Matrices tend not to be big along both dimensions
 - Matrix with many partitions gets less queries
 - ▶ Query optimization algorithm
 - ▶ Large $p \rightarrow$ small r
 - Matrix with fewer partitions gets more queries
 - ▶ Small $p \rightarrow$ large r
 - ▶ Replication cost small
- Will study the effects in comprehensive system evaluation

Roadmap

- Content Discovery System (CDS)
- Thesis statement
- Related work
- Proposed CDS system
- Research plan
- Time line
- Expected contributions

Implementation Plan

- Simulator implementation
 - For evaluation under controlled environment
 - Plan to use Chord simulator as a starting point
- Actual implementation
 - Implement CDS as a generic software module
 - Deploy on the Internet for evaluation
 - Implement real applications on top of CDS

Evaluation Plan

- Work load generation
 - Synthetic load
 - ▶ Use known distributions to model AV-pair distribution in names and queries
 - Benchmarks
 - ▶ Take benchmarks used in other applications, e.g., databases
 - Collect traces
 - ▶ Modify open source applications to obtain real traces
- Performance metrics
 - Registration and query response time
 - Success/blocking rate
 - System utilization

System Improvements

➤ Performance

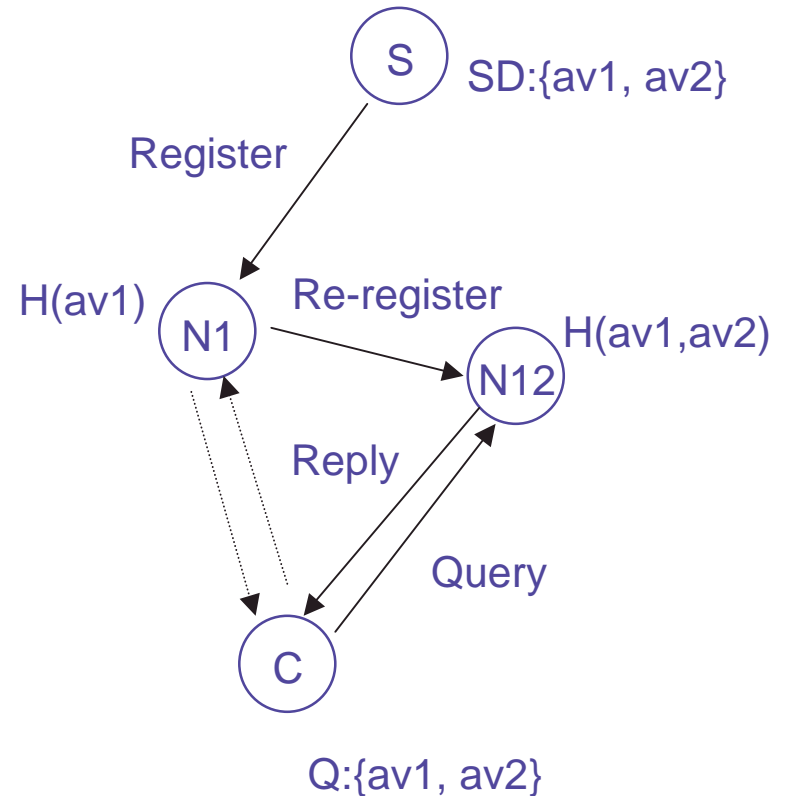
- Specialized resolvers
 - ▶ Combine AV-pairs
- Search within a matrix

➤ Functionality

- Range search
 - ▶ Auxiliary data structure to index the RP nodes
- Database operations
 - ▶ E.g., “project”, “select”, etc.

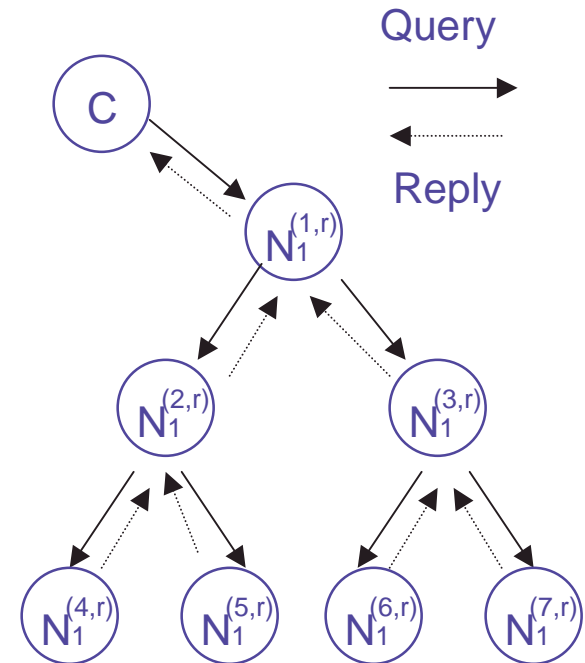
Specialized Resolvers

- Problem
 - All the RP matrices corresponding to a query are large, but the number of matched contents is small
 - ▶ $Q:\{\text{device}=\text{camera}, \text{location}=\text{weh7110}\}$
- Idea
 - Deploy resolvers that correspond to the AV-pair combination
- Mechanism
 - First level resolver monitors query rate on subsequent AV-pair
 - Spawn new node when reaches threshold
 - Forward registration to it



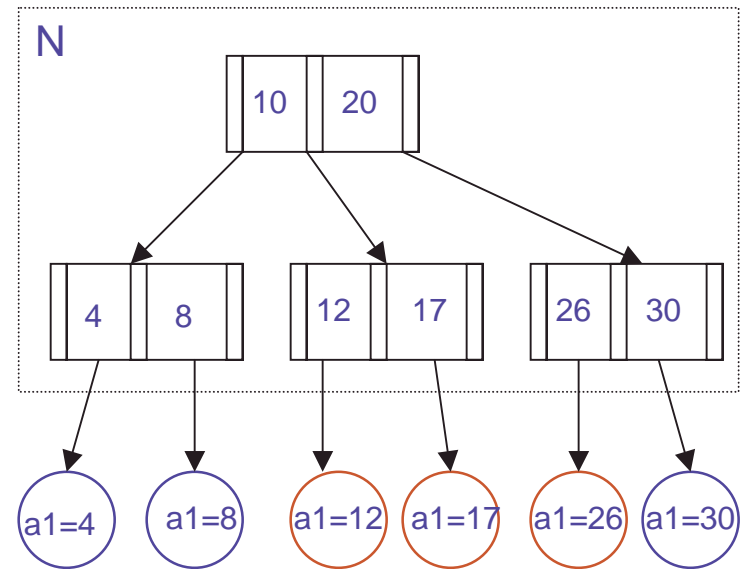
Improve Search Performance within LBM

- For a query, the selected matrix may have many partitions
 - Reply implosion
- Organize the columns into logical trees
 - Propagate query from root to leaves
 - Collect results at each level
 - ▶ Can exercise “early termination”



Support for Range Search

- Hash makes range search difficult
 - No node corresponds to $a_1 > 26$
 - Nodes do not know each other even if share attribute
- Mechanism
 - Use an auxiliary data structure to store the related nodes
 - ▶ E.g., B-tree stored on $N = H(a_1)$
 - Registration and query go through this data structure to collect the list of nodes to be visited



Q: { $8 < a_1 < 30$ }

Time Line

Tasks	Summer'02	Fall'02	Spring'03	Summer'03	Fall'03
Basic CDS simulator implementation	←→				
Incorporate load balancing mechanisms	←→				
Synthetic load and Benchmark evaluation	←→				
Actual implementation		←→			
Collect traces and comprehensive evaluation		←→			
System improvement		←→			
Internet evaluation		←→			
Writing	←→				

Expected Contributions

- System
 - Demonstrate the proposed CDS provides a scalable solution to the content discovery problem
- Architecture
 - Show content discovery is a critical layer in building a wide range of distributed applications
- Software
 - Contribute the CDS software to the research community and general public