# Locating Data Sources in Large Distributed Systems

Leonidas Galanis, Yuan Wang, Shawn R. Jeffrey, David J. DeWitt
Proceedings of the 29th VLDB Conference, 2003

Rolando Blanco
CS856 – Winter 2005

# Outline

- Background and problem definition
- Catalogue service and implementation
- Simulation results
- Summary and observations

# Background

- DB Research at Wisconsin
  - Niagara:
    - Centralised XML query engine (with crawler)
    - Finds xml files relevant to a query
  - Niagara distributed
  - Replication of catalogue within an horizon
  - Poor performance

# Problem Definition

- Location of data sources relevant to a given query
- Assumption: thousand of nodes
- Options:
  – Flooding
  – Catalogue service
    - Centralised: expensive/single point of failure
    - Replicated: maintenance issues, scalability
    - Fully distributed
    - Variations (e.g. supernodes)

# Catalogue Service

- Catalogue describes data for all nodes
- Assuming XML data sources
- Entry for node $N_i$: ($k_{j,}$ $S_{ij}$)
  - Associates and element/attribute ($k_j$) in $N_i$ to its summary $S_{ij}$
  - Summary can be structural (unique paths to $k_j$ in $N_i$) or describe value (e.g. Histogram, bloom vector)
  - Structural paths should include ascendant/descendant information
  - Summary updates only by node providing initial summary
- Example:

```
{ (price,
   { '/store/book/price', '/store/dvd/price', '/store/cd/price', ... },
   {(10...200), (5 ... 45), (5 ... 25)}
}
```

# Catalogue Service

- Catalogue implements two main functions:
  1. ***query_parts***: extracts set of elements/attributes for a query
  2. ***map***: decides what nodes are relevant to a query based on results from *query_parts* and summary data
- Goals:
  – Result of executing query in nodes identified by function *map* should be non-empty
  – Data on identified nodes should be required to produce final result
- *map* implementation
  – B+ trees to implement summaries

# Summary Implementation / B+ Trees

- B+ tree:
  - keys: $(k_j, N_i)$
  - values:
    - Structural summary: all paths to $k_j$ in peer $N_i$
    - Value summary: histogram, bloom vector, etc
- Given a query $/a_1/a_2/.../a_n/k$ *op x*
  - Retrieve summaries
  - Use structural summary to decide if $/a_1/a_2/.../a_n/k$
  - Use value summary to decide if *k op x*
- Given query involving several *k*'s:
  - $/a_1/a_2/.../a_n/k_1$ op x
  - $/b_1/b_2/.../b_m/k_2$ op y
  - Note $N_i$ in B+ tree key

# Summary Implementation / B+ Trees

- Issues if *k* in many nodes
- Solutions:
  1. Use *(k, cluster of nodes)* as B+ tree keys; compound summaries for paths of nodes in cluster, or
  2. Use $k/a_n/a_{n-1}/.../a_1$ as B+ tree keys, $N_i$'s as values
     - Allows range scan (useful when query looks like $//.../a_n/k$)
     - Attribute names can be hashed to integers to keep size of index small
     - If a node provides *n* paths to *k*, there will be *n* keys in B+ tree
     - If path is present in *n* nodes there will be *n* nodes in B+ tree value

# Summary Implementation / B+ Trees

"Our study assumes that scalable, efficient and reasonably sized index is available on each participating node"

# Catalogue Implementation

- DHT (Chord)
- DHT Hash keys: $k_j$'s
- DHT Value: Node where summary for $k_j$ is stored
- *Both DHT keys and summaries stored on same node*

# Catalogue Implementation

| | Paths in XML Data |
|---|---|
| $N_1$ | library/catalogs/book/author, library/reservation/book/author |
| $N_2$ | bookstore/book/price, bookstore/book/author |
| $N_3$ | bookstore/book/price, bookstore/book/author |
| $N_4$ | bookstore/book/price, bookstore/book/author |

**Table 1: Nodes with sample data**

| | DHT Index |
|---|---|
| $N_1$ | author: $\{(S_{1,author}), (S_{2,author}), (S_{3,author}), (S_{4,author})\}$ |
| $N_2$ | reservation: $\{(S_{1,res.})\}$ |
| $N_3$ | -- |
| $N_4$ | book: $\{(S_{1,book}), (S_{2,book}), (S_{3,book}), (S_{4,book})\}$, price: $\{(S_{2,price}), (S_{3,price}), (S_{4,price})\}$ |

**Table 2: Part of the DHT index on each node**

Query: $Q_2$: //book[author = “J Smith”]/price on $N_3$

$N_3$: query_parts($Q_2$) = $Q_{21}$: //book/**price**

$Q_{22}$: //book/**author** = “J Smith”

$N_3$: dht::lookup(price) = $\{N_4\}$

$Q_2$ and $Q_{21}$ sent to $N_4$

$N_4$: map(price, //book/price) = $\{ N_2, N_3, N_4 \}$ (B+ tree)

dht::lookup(author) = $\{ N_1\}$

$Q_2$, $Q_{22}$, $\{N_2, N_3, N_4\}$ sent to $N_1$ (why $N_4$ and not $N_3$ ?)

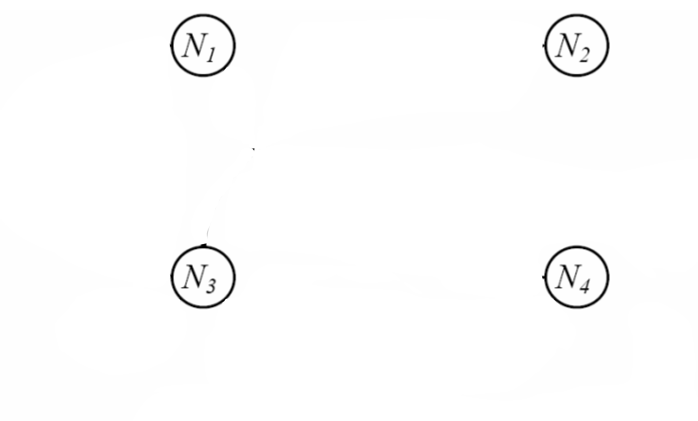
$N_1$: (map(author, /book/author) and *author = “J. Smith”*) = $\{N_2\}$

$\{N_2, N_3, N_4\} \cap \{N_2\} = \{N_2\}$

$\{N_2\}$ sent to $N_3$

$N_3$: sends $Q_2$ to $N_2$

$N_2$: executes $Q_2$ and returns results to $N_3$



# Catalogue Implementation

- General query processing:
  - $Q = /a_1[b_1]/a_2[b_2]/.../a_n[b_n]$ *op value*
  1. Rewrite as multiple simple paths
  2. Result N = { }
  3. For each simple path $/a_{i1}/a_{i2}/.../a_{im_j}$ *op value*
     - Visit node responsible for $a_{im_j}$ summary
     - Retrieve set of $N_i$'s that match path and condition
     - If N is empty then N = $N_i$'s, else N = $N_i$'s $\cap$ N
  - N is the set of nodes where Q should be run

# System Evolution

- Assumption: low volatility (churn rate)
  - Data providers leave system for schedule maintenance
- Node joining: Chord + catalogue entries hosted on same node holding DHT key.
- Node leaving: Chord + inform nodes holding catalogue, *or do nothing ("they will find out overtime"* -- when trying to use data or as part of Chord maintenance?)
- Note high volatility would cause a lot of traffic (catalogue entries must be moved with keys)

# Scalability

- Popular queries increase load in nodes that hold related keys (node holding the data would get loaded as well but data is not moved)
- Solution:
  - Key splitting
  - Key replication

# Key Splitting

- Request for *k* exceeds threshold (20 in simulations)
- Split key into $p_1$/k, $p_2$/k, ..., $p_n$/k
  - book/price, dvd/price, cd/price, ...
  - Node N defines *metakey* map

    *(price ® {book/price, dvd/price, cd/price, ... })*
  - Summaries need to be split as well
  - New keys and summaries inserted in DHT
    - Old key still in DHT
  - Node N handling k can:
    - Keep summaries (split – replicate)
    - Delete summaries (split-toss)

# Key Splitting

- Issues
  - Queries still refer to *k*
  - Node N needs to remember split and inform nodes querying (*what if N dies?)*
  - Some queries still need to be propagated to all nodes or be handled by N (//store[name="..."]//price < 1000)
  - If split-replicate, node with subkeys can discard subkey if # queries below threshold (*so nodes with split keys need to know N's decision*). If split-toss coordination is required to merge.
  - When split no longer possible (*n-1* splits in a *n* path):
    - Replicate

# Key Replication

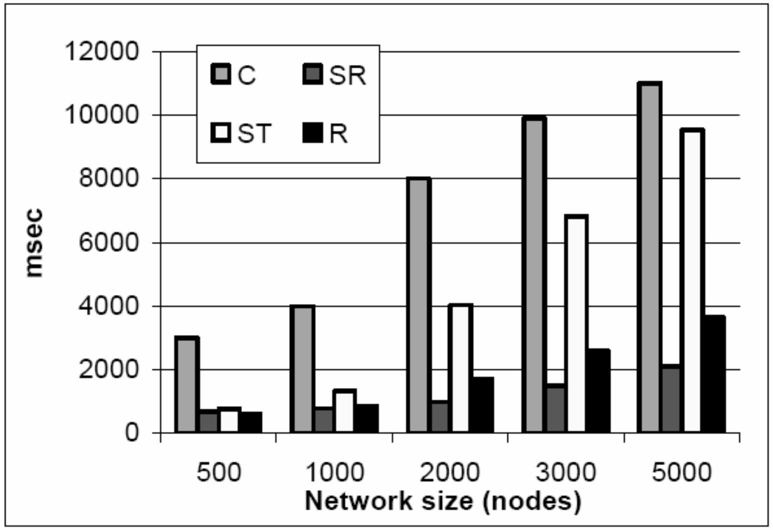
- When request for *k* exceeds threshold (i.e. by itself as load balancing strategy) or splitting not possible
- Replication in one or more sites (configurable)
- Summary goes along with keys
  - Node querying informed of replication
    - Round robin
  - Updates need to be propagated to all replicas
- Updates need to be propagated to all replicas
- *If need to replicate again who makes the decision? original site or copy sites (or both?)*

# Simulations

- Goal: Measure catalogue lookup scalability
- 3,500 keys
- 16,000 paths
- No updates
- Structural summaries 100% accurate
- Value summaries 100% inaccurate
- Some schemas more popular than others. Query credits assigned based on schema popularity
- Queries biased toward leafs
- Query pool: 1’000,000 queries
- Users: 10 x #nodes
- Queries: 800 x #nodes
- User submits query, waits for response, thinks 5 secs, types for 3 secs, submits query from query pool
- Split after 20 requests and queuing (*why not requests/interval*)
- Latency avg 50 ms between nodes
- No volatility, NW stabilises before running queries
- Queue size at each node: 500

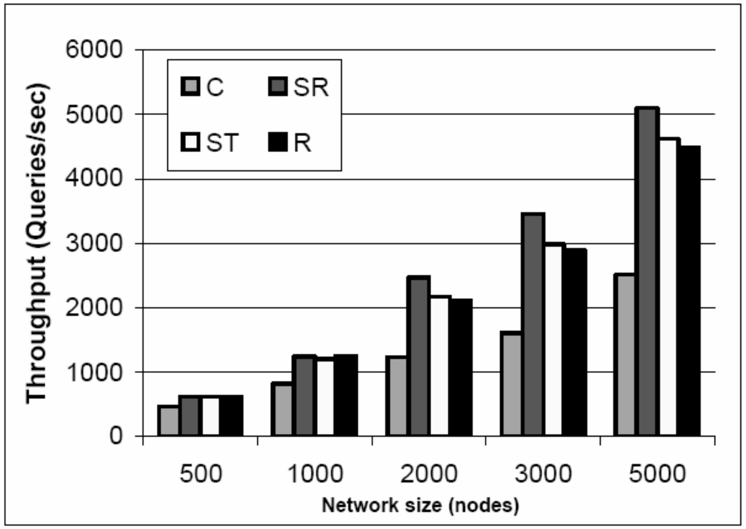
# Simulations / Performance

**Chord (C), Split-Replicate (SR), Split-Toss (ST), Replication one-at-a-time (R)**




**Graph 1: Average response times for catalog queries**

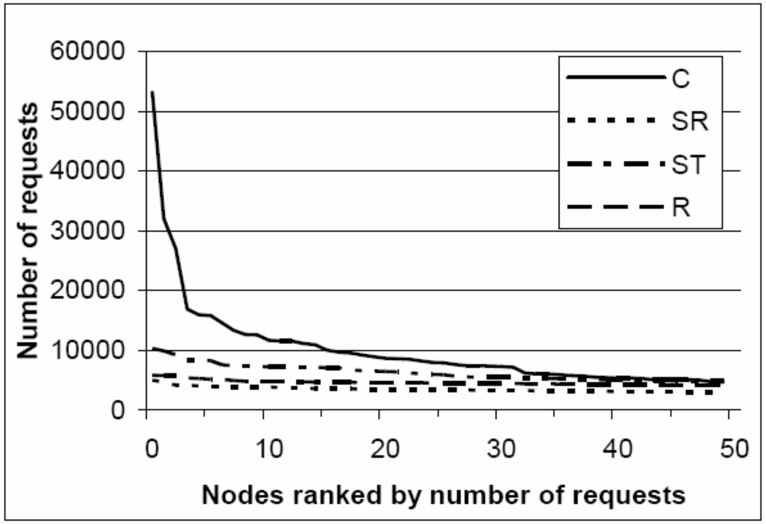
SR: best scalability
ST: effect of toss is substantial
R: better than ST, it does not adapt as fast as SR to load



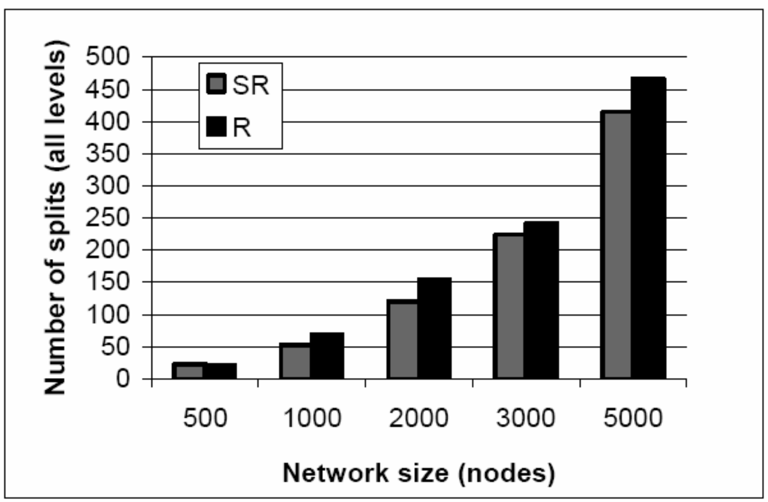

**Graph 2: Combined throughput of queries**

SR: still best
ST: now better than R. More keys are generated per split

# Simulations / Load Distribution




Graph 4: Request distribution (2000 Nodes)




Graph 6: Number of load balancing actions for SR (all levels) and R

-Some keys are more popular. If Chord-only some sites may get overwhelmed

- First (more loaded) 50 nodes:
    - 500 nodes: handling 27% of query load
    - 5000 nodes: handling 7% of query load
    - 34% query load if no load balancing
- ST: 1.5 to 2.5 more catalogue requests than SR

| Network sizes | C | SR | ST | R |
|---|---|---|---|---|
| 500 | 2% | 0% | 0% | 0% |
| 1000 | 4% | 0% | 0% | 0% |
| 2000 | 8% | 0% | 2% | 0% |
| 3000 | 11% | 0% | 4% | 0% |
| 5000 | 18% | 1% | 6% | 1% |

Table 5: Droped requests across all configurations

At 2000 nodes: SR 111 splits, 1793 new keys; R 149 replicas, 149 new keys

Cascading effect noticed: nodes become overwhelmed by accepting popular keys

# Summary

- Catalogue framework over structured P2P to locate XML data sources
- Application (catalogue service) running on Chord
- Distributed design, allows providers to join and make data query-able
- Techniques to adapt to query workload (adaptive key management and summary redistribution)
- Experimental evaluation

# Related Work

- Data location:
  - Unstructured P2P routing indices
    - Bloom filters [koloniari04] / Qiang Wang's work
    - Histograms [Petrakis04]
- Load balancing [Triantafillou03]
  - Fair load distribution
    - Cluster based on semantic similarity
    - Goal: all clusters have similar load
    - Replication within cluster
    - Choose randomly node in cluster (when querying)
    - Not all clusters have same number of nodes

[koloniari04] Koloniari et al. Content-Based Routing of Path Queries in Peer-to-Peer Systems. EDBT, 2004.
[Petrakis04] Petrakis et al. On Using Histograms as Routing Indices in Peer-to-Peer Systmes, DBISP2P 2004.
[Triantafillou03] Triamtafillow et al.Towards High Performance Peer-to-Peer Content and Resource Sharing Systems. CIDR 2003

# Observations / Comments

- Only structural summaries in simulations.
- Issues if voluminous value summaries
- Queries that may require joining data from multiple data sources, located on different nodes.
- Adaptive key management:
  - Routing by DHT and catalogue
  - Reliability implications
  - Multiple DHTs to avoid catalogue routing?
- Identification of redundant data sources (in map?)
- Example / Algorithm

# References


- Leonidas Galanis, et al. Processing Queries in a Large Peer-to-Peer System. CAiSE 2003
- Shawn R. Jeffrey. Peer-to-Peer Research at Wisconsin, Presentation at www.cs.berkeley.edu/~jeffrey/p2patuw.ppt
- G. Koloniari and E. Pitoura. Content-Based Routing of Path Queries in Peer-to-Peer Systems. In Proc. of EDBT (International Conference on Extending Database Technology), 2004.
- Y. Petrakis, G. Koloniari, E. Pitoura. On Using Histograms as Routing Indexes in Peer-to-Peer Systems. In DBISP2P 2004, August 29-30
- P. Triantafillou et al. Towards High Performance Peer-to-Peer Content and Resource Sharing Systems. CIDR 2003
- J. Naughthon et al. The Niagara Internet Query System. VLDB 2000