Cache-and-Query for Wide Area Sensor Databases

Amol Deshpande, UC Berkeley
Suman Nath, CMU
Phillip Gibbons, Intel Research Pittsburgh
Srinivasan Seshan, CMU
Outline

• Overview of IrisNet

• Example application: Parking Space Finder

• Query processing in IrisNet
  • Data partitioning
  • Distributed query execution

• Conclusions
Internet-scale Resource-intensive Sensor Network Services (IrisNet)

- **Motivation**
  - Proliferation of resource-intensive sensors attached to powerful devices
    - Webcams, pressure gauges, microphones
    - Rich data sources with high data volumes
  - Typically distributed over wide geographical areas
  - Useful services utilizing such sensors missing

- **IrisNet:** An infrastructure to support deployment of sensor services over such sensors
IrisNet: Design Goals

• Ease of deployment of sensor services
  • Minimal requirements from the service provider

• Distributed data storage and querying for high throughputs

• Ease of querying
  • XML as the data format, XPATH as the query language
  • Natural geographical hierarchy on data as well as queries
  • Continuously evolving data

• Location transparency
  • Logical view of the entire distributed database as a single centralized XML document
Sensing Agents (SA)
- PDA/PC-class processor, MBs–GBs storage
- Collect & process data from sensors, as dictated by “senselet” code uploaded by OAs
- Processed data sent to the OAs for update in-place

Organizing Agents (OA)
- PC/Server-class processor, GBs storage
- Provide data storage, discovery, querying facilities
- Use an off-the-shelf database to store data locally
  - Interface with the local database using XPATH/XSLT
Outline

• Overview of IrisNet
• Example application: Parking Space Finder
• Query processing in IrisNet
  • Data partitioning
  • Distributed query execution
• Conclusions
Example Application: Parking Space Finder (PSF)

- Webcams monitor parking spaces and provide real-time information about their availability
  - Image processing to extract availability information
- Natural geographical hierarchy on the data
Example XML Fragment for PSF

```xml
<State id="Pennysylvania">
  <County id="Allegheny">
    <City id="Pittsburgh">
      <Neighborhood id="Oakland">
        <total-spaces>200</total-spaces>
        <Block id="1">
          <GPS>...</GPS>
          <pSpace id="1">
            <in-use>no</in-use>
            <metered>yes</metered>
          </pSpace>
          <pSpace id="2">
            ...
          </pSpace>
        </Block>
      </Neighborhood>
      <Neighborhood id="Shadyside">
        ...
      </Neighborhood>
    </City>
  </County>
</State>
```
Example XML Fragment for PSF
**Example Queries**

- **Users issue queries against the document as a whole**
  - Find all available parking spots in Oakland
    - `/State[@id="Pennsylvania"]`
    - `/County[@id="Allegheny”]/City[@id="Pittsburgh”]`
    - `/Neighborhood[@id="Oakland”]/Block/pSpace[in-use = “no”]`
  
  - Find all blocks in in Allegheny have more than 20 metered parking spots
    - `/State[@id="Pennsylvania”]/County[@id="Allegheny”]`
      - `//Block[count(./pSpace[metered = “yes”]) > 20]`
  
  - Find the cheapest parking spot in Oakland Block 1
    - `/State[@id="Pennsylvania”]/County[@id="Allegheny”]/City[@id="Pittsburgh”]`
      - `/Neighborhood[@id="Oakland”]/Block[@id=’1’]`
      - `/pSpace[not(../pSpace/price > ./price)]`

- **Challenge**: Evaluate arbitrary XPath queries against the document even though the document may be partitioned across multiple OAs
Data Partitioning and Query Processing: Overview

- Maintain data partitioning invariants
  - Used to guarantee that an OA always has sufficient information to participate correctly in a query

- Use DNS to maintain the data distribution information and to route queries to data

- Convert the XPATH query to an XSLT query that:
  - Walks the document recursively
  - Evaluates part of the query that can be done locally
  - Gathers missing information by asking subqueries
Outline

• Overview of IrisNet
• Example application: Parking Space Finder
• Query processing in IrisNet
  • Data partitioning
  • Distributed query execution
• Conclusions
**Partitioning Granularity**

- **Definition:** An **IDable node** in the document
  - Has an “id” attribute with value unique among its siblings
  - All its ancestors in the document are IDable
Definition: **Local Information** of an IDable node

- All its attributes and all its non-IDable descendants
- IDs of all its IDable children
Partitioning Granularity

• Definition: **Local Information** of an IDable node
  • All its attributes and all its non-IDable descendants
  • IDs of all its IDable children
Data Partitioning

- Data storage, ownership always in units of local informations corresponding to the IDable nodes in the document
  - These form a nearly-disjoint partitioning of the overall document
  - Granularity can be controlled using the “id” attributes
  - A partitioning unit can be uniquely identified using the “id”’s on the path to the root of the document

- Data ownership:
  - Each partitioning unit owned by exactly one OA
Data Partitioning

• Data stored locally at each OA:
  • A document fragment consisting of union of partitioning units
  • Constraints:
    • Must store the document fragment it owns
    • If stored the “id” of an IDable node, must also store the local information of all its ancestors
  • We minimize the amount of information required to store (details in paper)
    • Only need to store ID’s of all ancestors, and of their children

• Invariant:
  • If an OA has the “id” of an IDable node, it either
    • Has the local information for the node, or
    • Has the “id”’s on the path to the root allowing it to locate the local information for that node
Data Partitioning: Example

OA 1 Owns

County
id = 'Allegheny'

City
id = 'Pittsburgh'

Neighborhood
id = 'Shadyside'

OA 2 Owns

Neighborhood
id = 'Oakland'

Block
id = '1'

Block
id = '2'

pSpace
id = '1'

pSpace
id = '2'

pSpace
id = '3'
Data Partitioning: Example

Data storage configuration at OA 1
Data Partitioning: Example

Data storage configuration at OA 2

- County id = 'Allegheny'
- City id = 'Pittsburgh'
- Neighborhood id = 'Oakland'
- Neighborhood id = 'Shadyside'
- Block id = '1'
- pSpace id = '1'
- Block id = '2'
- pSpace id = '2'
- Block id = '3'
- pSpace id = '3'

Local information required

Local information optional
Mapping Data to OAs

- Mapping of nodes to physical OAs maintained using DNS
  - For each IDable node, create a unique DNS-style name by concatenating the IDs on the path to the root

OA 1 Owns

- Allegheny-County.iris.net
- Pittsburgh-City.Allegheny-County.iris.net

OA 2 Owns

- Oakland-Neighborhood.Pittsburgh-City.Allegheny-County.iris.net
- 1-Block.Oakland-Neighborhood.Pittsburgh-City.Allegheny-County.iris.net

• Mapped to OA 1:
  - Allegheny-County.iris.net
  - Pittsburgh-City.Allegheny-County.iris.net

• Mapped to OA 2:
  - Oakland-Neighborhood.Pittsburgh-City.Allegheny-County.iris.net
  - 1-Block.Oakland-Neighborhood.Pittsburgh-City.Allegheny-County.iris.net
  - …
Outline

• Overview of IrisNet
• Example application: Parking Space Finder
• Query processing in IrisNet
  • Data partitioning
  • Distributed query execution
• Conclusions
Self-Starting Distributed Queries

- Each query has a hierarchical prefix
  
  /State[@id='Pennsylvania']/County[@id='Allegheny']
  
  /City[@id='Pittsburgh']/ /Neighborhood[@id='Oakland']/Block/pSpace

- Simple parsing of the query to extract the least common ancestor (LCA) of the possible query result
  
  - Send the query to
    
    Oakland-Neighborhood.Pittsburgh-City.
    Allegheny-County.Pennsylvania-State.parking.intel-iris.net

- Name extracted from query **without any global or per-service state**
QEG Details

• **Nesting depth of an XPATH query**
  - Maximum depth at which a location path that traverses over IDable nodes occurs in the query

  **Examples:**
  - `/a[@id='x']/b[@id='y']/c` → 0
  - `/a[@id='x']///c` → 0
  - `/a[./b/c]/b` → 1 (if b is IDable)
  - `/a[count(./b/[./c[@id='1']])` → 2

• **Complexity of evaluating a query increases with nesting depth**
Queries with Nesting Depth = 0

• Any predicate in the query can be evaluated using just the local information for an IDable node
  • Example: .../Block[@id='1'][./available-spaces > 10]

• Sketch of the XSLT program:
  • Walk the document recursively
  • If local information for the node under consideration available, evaluate the part of the query that refers to that node, otherwise tag the returned answer with the tag “asksubquery”

• Postprocessor finds the missing information by asking subqueries
Caching

• A site can add to its document any fragment as long as the data partitioning constraints are satisfied.

• We generalize subqueries to fetch the smallest superset of the answer that satisfies the constraints and cache it.

• Data time-stamped at the time of caching.

• Queries can specify freshness requirements.
Further Details in Paper

- Queries with Nesting Depth > 0
- Schema changes
- Data partitioning changes
- Implementation details and experimental study
Conclusions

- Identified the challenges in query processing over a distributed XML document
- Developed formal framework and techniques that
  - Allow for flexible document partitioning
  - Integrate caching seamlessly
  - Correctly and efficiently answer XPATH queries
- Experimental results demonstrate the advantages of flexible data partitioning and caching
Further Information

• IrisNet project website
  • http://www.intel-iris.net
Outline

• Overview of IrisNet
• Example application: Parking Space Finder
• Query processing in IrisNet
  • Data partitioning
  • Distributed query execution
• Performance Study
Performance Study Setup

- Current prototype written in Java
- A cluster of 9 2GHz Pentium IV machines
- Apache Xindice used as the backend XML database
- Artificially generated database
  - 2400 parking spaces with 2 cities, 6 neighborhoods and 120 blocks
- Five query workloads
  - QW-1: Asking for a single block
  - QW-2: Asking for two blocks from a single neighborhood
  - QW-3: Asking for two blocks from two neighborhoods
  - QW-4: Asking for two blocks from two cities
  - QW-Mix: 40% of QW-1 and QW-2, 15% QW-3, 5% QW-4
Architectures Compared

Centralized

Centralized querying, distributed update
Caching

• Architecture already allows for caching data
  • An OA is allowed to store more data than that it owns
  • Data time-stamped at the time of caching
  • Queries can specify freshness tolerance
Architectures Compared

Distributed querying/update, fixed two-level organization

Distributed querying/updates, hierarchical organization
Query Throughputs

![Bar chart showing query throughputs for different architectures and workloads. The chart compares architectures 1, 2, 3, and 4 across different query workloads: QW-1, QW-2, QW-3, QW-4, and QW-Mix. Each bar represents the average throughput (queries/sec) for a given workload and architecture.]
Data Partitioning: Example 2

- OA 1 OWNS
- OA 2 OWNS

* e.g. OA 2 must store local information of the County(Allegheny) node
Conclusions

- Location transparency
  - distributed DB hidden from user

- Flexible data partitioning

- Low latency queries & Query scalability
  - Direct query routing to LCA of the answer
  - Query-driven caching, supporting partial matches
  - Load shedding; No per-service state needed at web servers

- Support query-based consistency

- Use off-the-shelf DB components
Example XML Fragment for PSF

```xml
...<County id="Allegheny">
  <City id="Pittsburgh">
    <Neighborhood id="Oakland">
      <available-spaces>8</available-spaces>
      <Block id="1">
        <pSpace id="1">
          <in-use>no</in-use>
          <metered>yes</metered>
        </pSpace>
        ...
      </Block>
    </Neighborhood>
  </City>
</County>
...```

```
County
  id = 'Allegheny'
City
  id = 'Pittsburgh'
Neighborhood
  id = 'Oakland'
  Block
    id = '1'
      Block
        id = '1'
          pSpace
            id = '1'
              in-use
                no
              metered
                yes
        id = '2'
          pSpace
            id = '2'
              GPS
                no
              metered
                yes
```

June 2003
IrisNet
Outline

• Overview of IrisNet

• Example application: Parking Space Finder

• Query processing in IrisNet
  • Data partitioning
  • Distributed query execution

• Performance Study