Querying the Internet with PIER

Article by: Ryan Huebsch, Joseph M. Hellerstein, Nick Lanham, Boon Thau Loo,
Scott Shenker, Ion Stoica, 2003
EECS Computer Science Division, UC Berkeley
International Computer Science Institute

Presentation by: Laura Tolosi

Supervisor: Prof. Gerhard Weikum

Overview

- **Motivation**
- **Querying the Internet**
- **CAN - A particular DHT**
- **PIER**
- **Validation and performance evaluation**

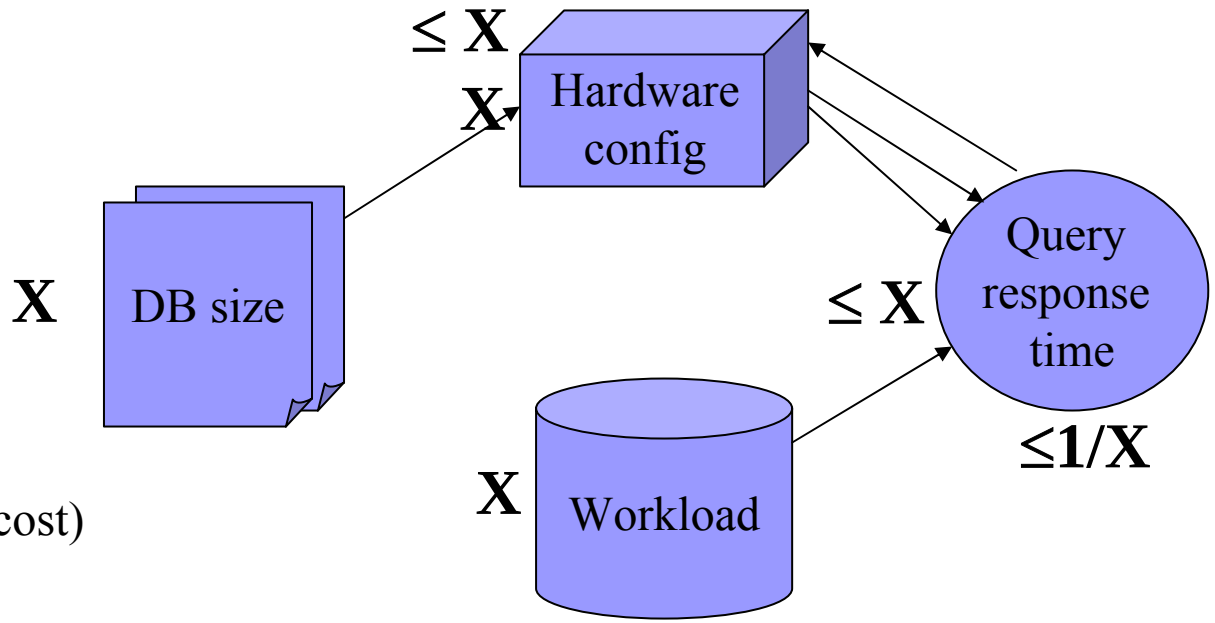
- **Motivation**
  - Scalability of Database Systems
- **Querying the Internet**
- **CAN - A particular DHT**
- **PIER**
- **Validation and performance evaluation**

Scalability of Database Systems

- Internet - hundreds million nodes
- The largest database systems - only few hundred nodes
- Scalability - 'the ability to grow your system smoothly and economically as your requirements increase'
- Parameters: data size, speed, workload, transaction cost
- Goal: huge number of concurrent users, continuous availability, large-stored data volume
- Measuring for scalability success:

- **size-up**
- **speed-up**
- **scale-up**
- **cost**

- workload increase should not increase transaction cost
- **X(**data size) implies ≤ **X (**cost)



- Motivation
- **Querying the Internet**
  - Applications
  - Design principles
- CAN - A particular DHT
- PIER
- Validation and performance evaluation

Querying the Internet

- Application example - **Network Intrusion Detection**
  - attack 'fingerprints': ***sequences of port accesses*** (port scanners), ***port numbers numbers*** and ***packet contents*** (buffer-overrun attacks, web robots), ***application level information on content*** (email spam) …
  - one can detect similar fingerprints, frequently reported

```
SELECT I.fingerprint, count(*)*sum(R.weight) AS wcnt
FROM intrusions I, reputation R
WHERE R.address = I.address
GROUP BY I.fingerprint
HAVING wcnt > 1.5
```

IP address	fingerprint	attribute1
1	F1	...
2	F1	...
3	F2	...
1	F2	...
3	F1	...

Table 1: **intrusions**

IP address	weight	attribute2
1	0.8	...
2	0.5	...
3	0.2	...

Table 2: **reputation**

fingerprint	wcnt
F1	4.5
F2	2.0

Table 3: **join result**

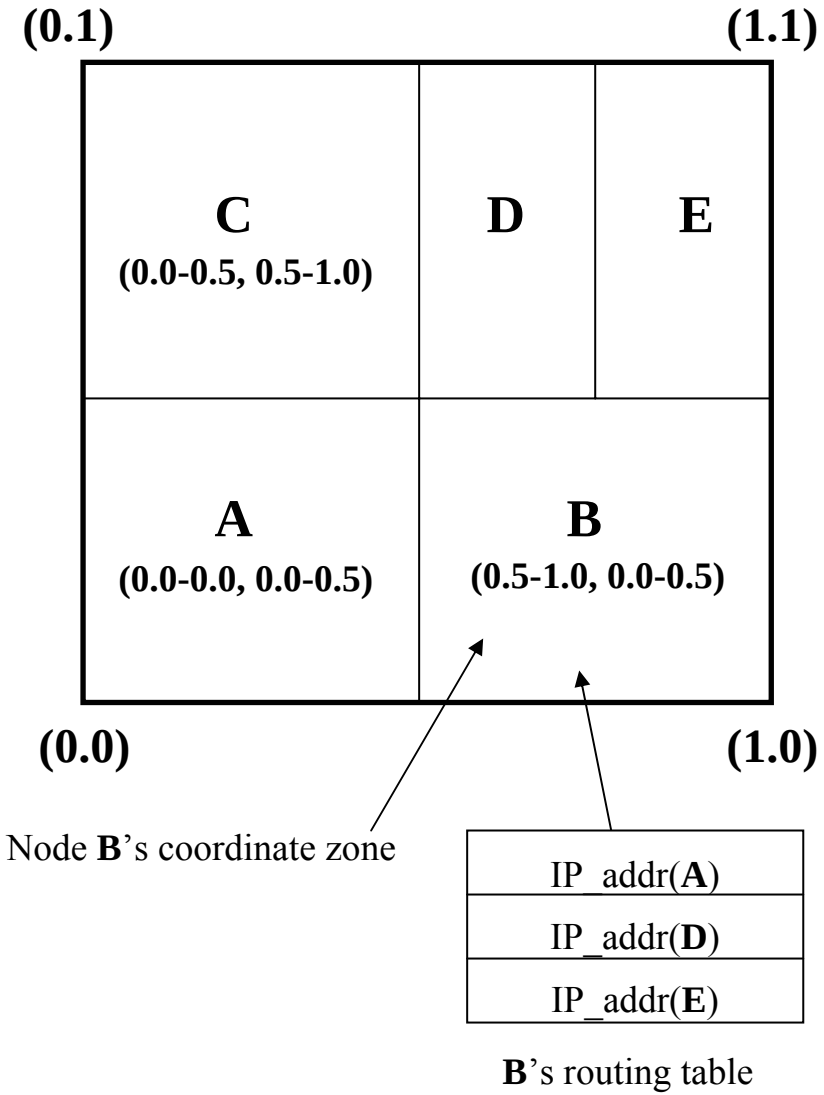
Querying the Internet

- Relaxed principles for scaling
  - Relaxed consistency
  - Organic Scaling
  - Natural Habitats for Data
  - Standard Schemas via Grassroots Software

- Motivation
- Querying the Internet
- **CAN - A particular DHT**
  - Design, construction, joining, routing… (quick overview)
- PIER
- Validation and performance evaluation

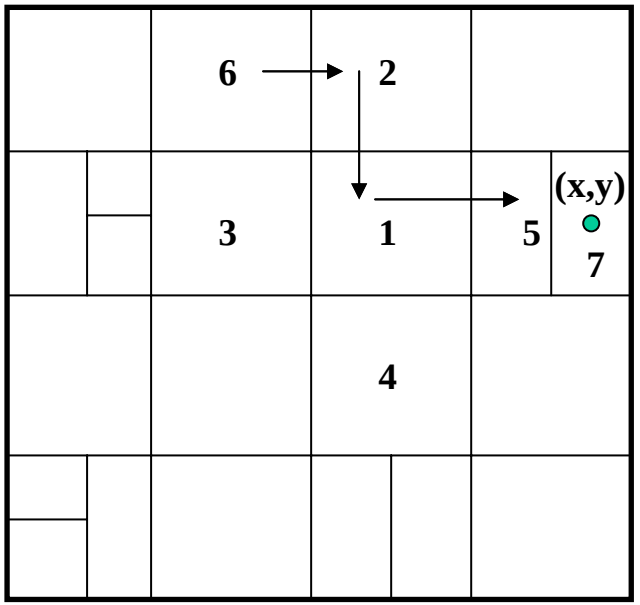
Content-addressable Network (CAN)

- CAN is a particular DHT
- virtual d-dimensional Cartesian coordinate space partitioned among the nodes in the system
- store (**key, value**) pairs using a hash function:
  - **key** → (f_1(**key**), … , f_d(**key**))
- **key** is mapped to a point in the coordinate space
- (**key, value**) will be stored by the node responsible for the zone in which the point lies.
- each node maintains as its set of neighbors the IP addresses of node with adjoining zones



Content-addressable Network (CAN) - continued

- Routing in CAN
  - hash(key) = (x, y)
  - 'simulate' a straight line from source to destination nodes
- Joining
  - a new node randomly chooses a point in the space (say (x,y))
  - sends a join request to an existing node in CAN (say 6)
  - the message is routed until the zone in which the point lies is reached
  - the zone is split
- More on CAN: Node departure, Recovery ...

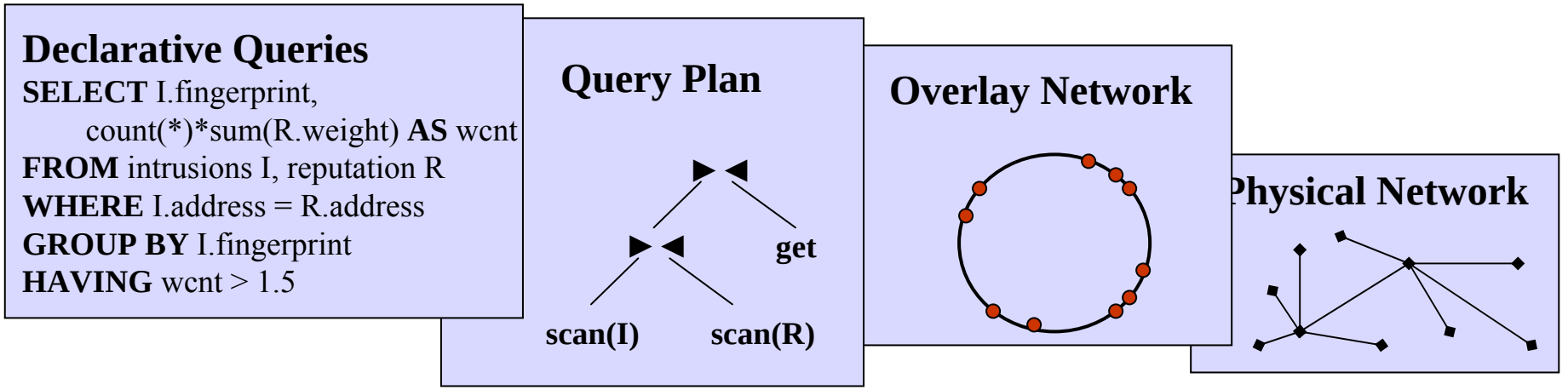


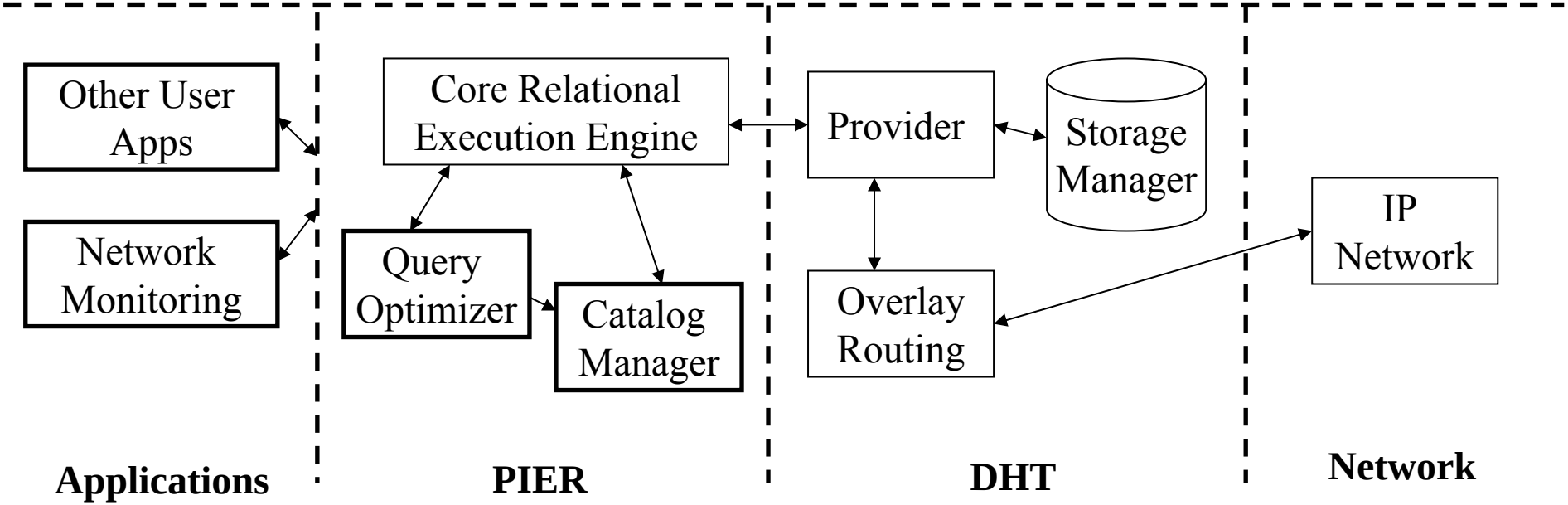
- For a d-dimensional space partitioned in n equal zones:

- the average routing path length : $\frac{d}{4} n^{\frac{1}{d}}$
- an individual node maintains $2d$ neighbors

- Motivation
- Querying the Internet
- CAN - A particular DHT
- **PIER**
  - Architecture
    - Routing layer
    - Storage manager
    - Provider
  - DHT distributed joins
    - Symmetric hash join
  - Validation and performance evaluation
-

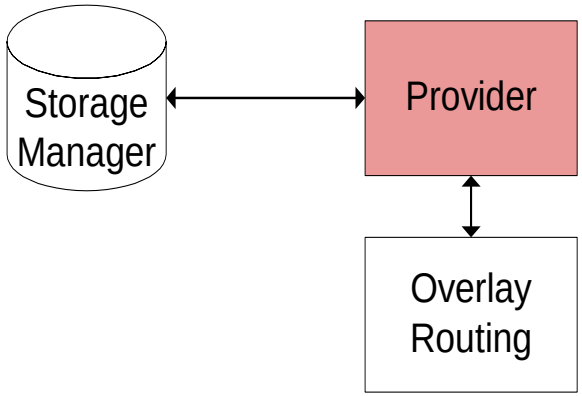
PIER Architecture





DHT Design

- Routing Layer
  - Mapping for keys
  - Dynamic as nodes join and leave
- Storage Manager
  - DHT based data
- Provider
  - Storage access interface for higher levels



- Any particular DHT can be used (CAN, Chord ...)

Routing Layer

- Maps a **key** into the **IP** address of the node currently responsible for that key.
- Provides exact lookups, callbacks higher levels when the set of keys has changed.

- Routing Layer API

  lookup (key) → ipaddr

  join (landmark)

  leave ()

  locationMapChange()

Storage Manager

- **Stores** and **retrieves** records, which consist of (key, value) pairs.
- Keys are used to locate items and can be any data type or structure supported

- Storage Manager API

  store (key, item)

  retrieve (key) → item

  remove (key)

Provider

- Ties routing and storage manager layers and provides an interface to applications
- Each object in the DHT has a ***namespace***, ***resourceID*** and ***instanceID***
- ***namespace***: application or group of objects, table or relation
- ***resourceID***: primary key or any attribute
- ***instanceID***: integer to separate items with the same ***namespace*** and the same ***resourceID***
- DHT key **:** hash (***namespace, resourceID***)
- Provider API

  get (namespace, resourceID) → item
  put (namespace, resourceID, instanceID, item, lifetime)
  renew (namespace, resourceID, instanceID, item, lifetime) → bool
  multicast (namespace, resourceID, item)
  lscan (namespace) → iterator
  newData (namespace) → item

Query Processor

- Performs selection, projection, join, grouping, aggregation
- Items are inserted, updated, deleted via DHT interface
- Operators produce results as quick as possible (***push***)
- Enqueue the data for the next operator (***pull***)

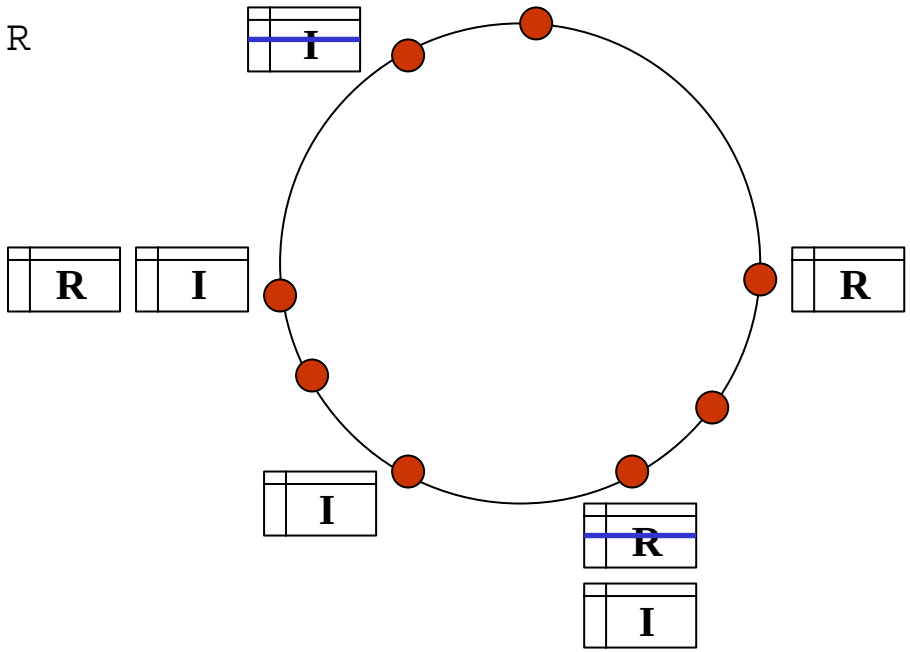
- ***reachable snapshot***: the set of data published by reachable nodes at the time the query is sent from the client node
- ***dilated reachable snapshot***: union of local snapshots of data published by reachable nodes
    - local snapshot: data published at the time the query message arrived at the node

DHT Distributed Joins

• **relational database join** performed to some degree **in parallel** by a umber of processors (**machines**) containing different parts of the relations on which join is performed

• DHT distributed join example:

```
SELECT I.fingerprint,
 count(*)*sum(R.weight) AS wcnt
FROM intrusions I, reputation R
WHERE R.address = S.address
GROUP BY I.fingerprint
HAVING wcnt > 1.5
```



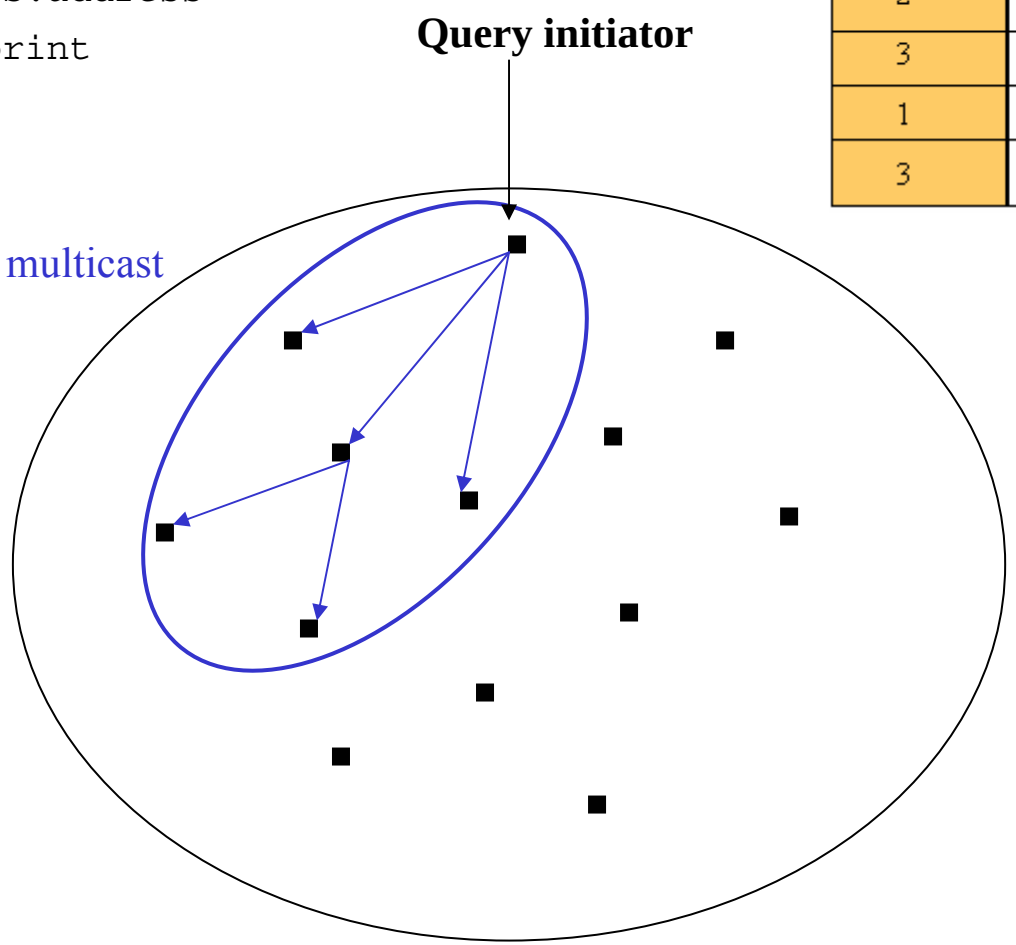
• match records on different machines

DHT Distributed Joins - continued

```
SELECT I.fingerprint,
   count(*)*sum(R.weight) AS wcnt
FROM intrusions I, reputation R
WHERE R.address = S.address
GROUP BY I.fingerprint
HAVING wcnt > 1.5
```

IP address	fingerprint	attributel
1	F1	...
2	F1	...
3	F2	...
1	F2	...
3	F1	...

Table: **intrusions**



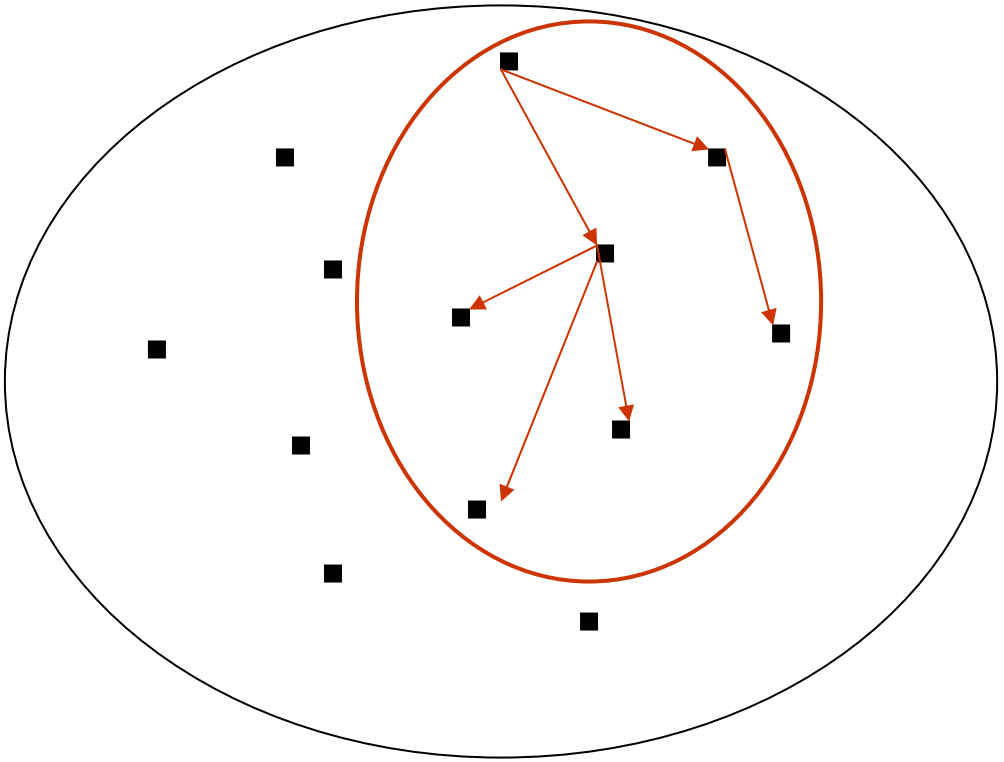
DHT Distributed Joins - continued

```
SELECT I.fingerprint,
   count(*)*sum(R.weight) AS wcnt
FROM intrusions I, reputation R
WHERE R.address = S.address
GROUP BY I.fingerprint
HAVING wcnt > 1.5
```

IP address	weight	attribute2
1	0.8	...
2	0.5	...
3	0.2	...

Table: **reputation**

Namespace N_R = multicast group(***B*** - *group)*



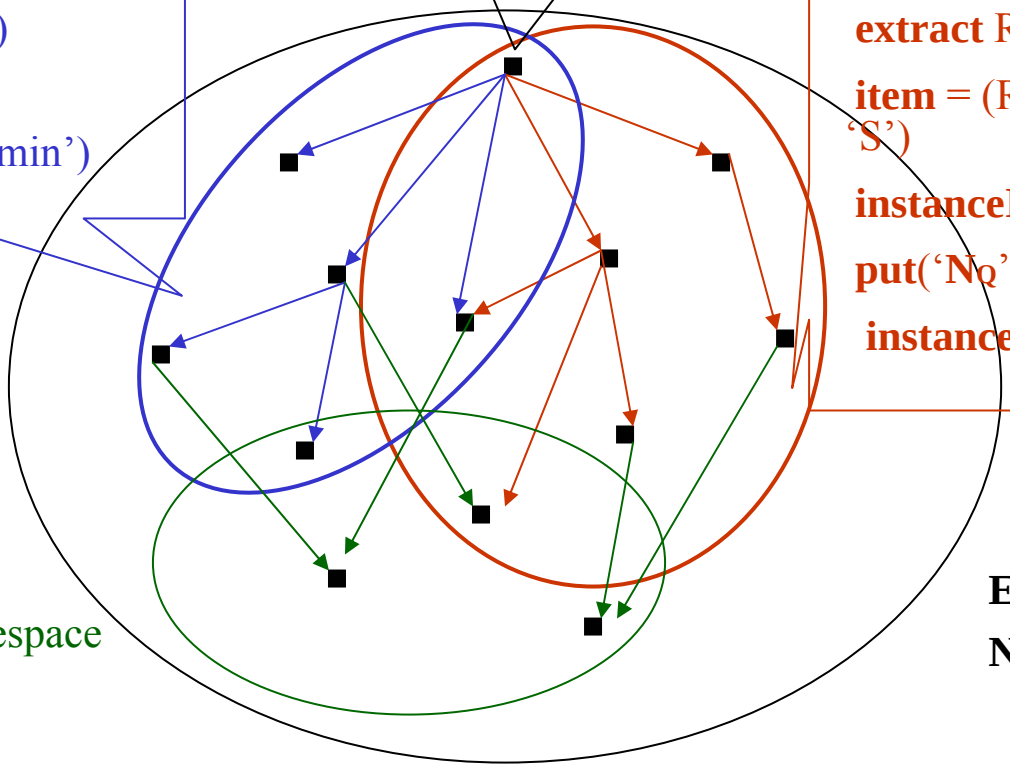
Entire DHT Network

DHT Distributed Joins - continued

[3] ***l*scan**:
For each tuple in I:
extract I.address, I.fingerprint
item = (I.address,
I.fingerprint, 'I')
instanceID = random()
put('N_Q', I.address,
instanceID, **item**, '60 min')

[1] multicast(I-group,'oper-j','SELECT I.fingerprint, ...')
[2] multicast(R-group,'oper-j','SELECT I.fingerprint, ...')

[3] ***l*scan**:
For each tuple in R:
extract R.address, R.weight
item = (R.address, R.weight, 'S')
instanceID = random()
put('N_Q', R.address,
instanceID, **item**, '60 min')



PUSH phase

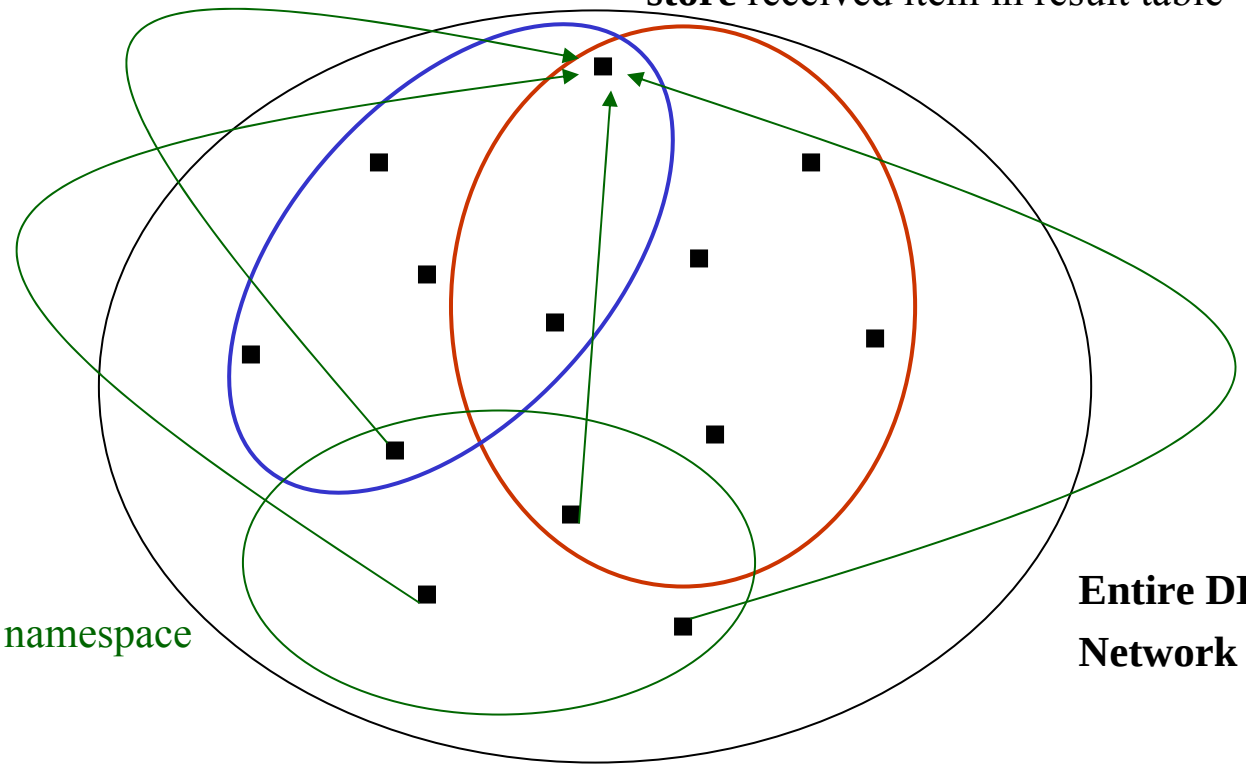
N_Q namespace

Entire DHT Network

DHT Distributed Joins - continued

[4]**newData**(‘**NQ**’, item)
for each call of **newData()**
get(‘NQ’, item.resourceID**)**
if **get** returns both an **I**-derived tuple and an **R**-derived tuple, then
send(‘oper-j’**, hash**(initiator), (I.fingerprint, R.weight)**)**

[5] **receive(**‘oper-x’, (I.fingerprint, R.weight)**)**
store received item in result table



- Motivation
- Querying the Internet
- CAN - A particular DHT
- PIER
- **Validation and performance evaluation**

Simulation setup

- Workload:

```
SELECT R.pkey, S.pkey, R.pad
FROM R, S
WHERE  R.num1 = S.pkey
    AND R.num2 > constat1
    AND S.num2 > constant2
    AND f(R.num3, S.num3) > constant3
```

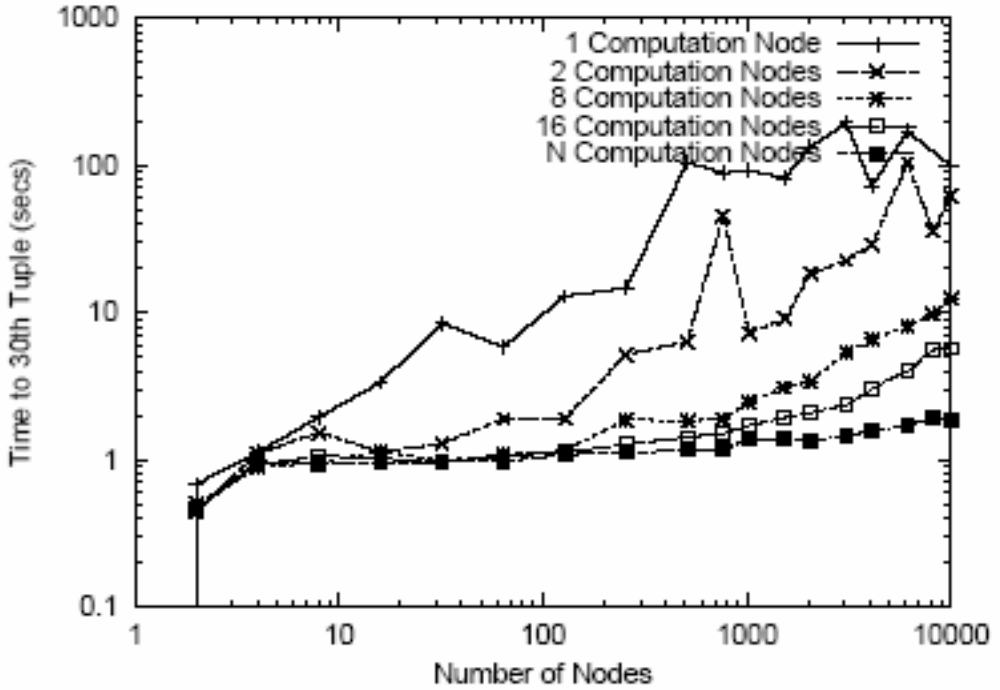
- R has 10 times more tuples than S
- attributes in S, R are uniformly distributed
- constants chosen as to produce 50% selectivity
- R.pad attribute assures that all tuples are 1KB in size

- 10 000 nodes network
- cluster of 64 PCs
- topology : fully connected network, 100ms latency , 10Mbps
- assumption: data changes at a rate higher than the rate of incoming queries
- all measures: performed after CAN routing stabilizes, tables R, S are in the DHT

Scalability evaluation

- evaluate it by proportionally increasing the load with the number of nodes
- each node - 1MB data
- measure the response time for the 30-th tuple
- avoid to measure the first response time

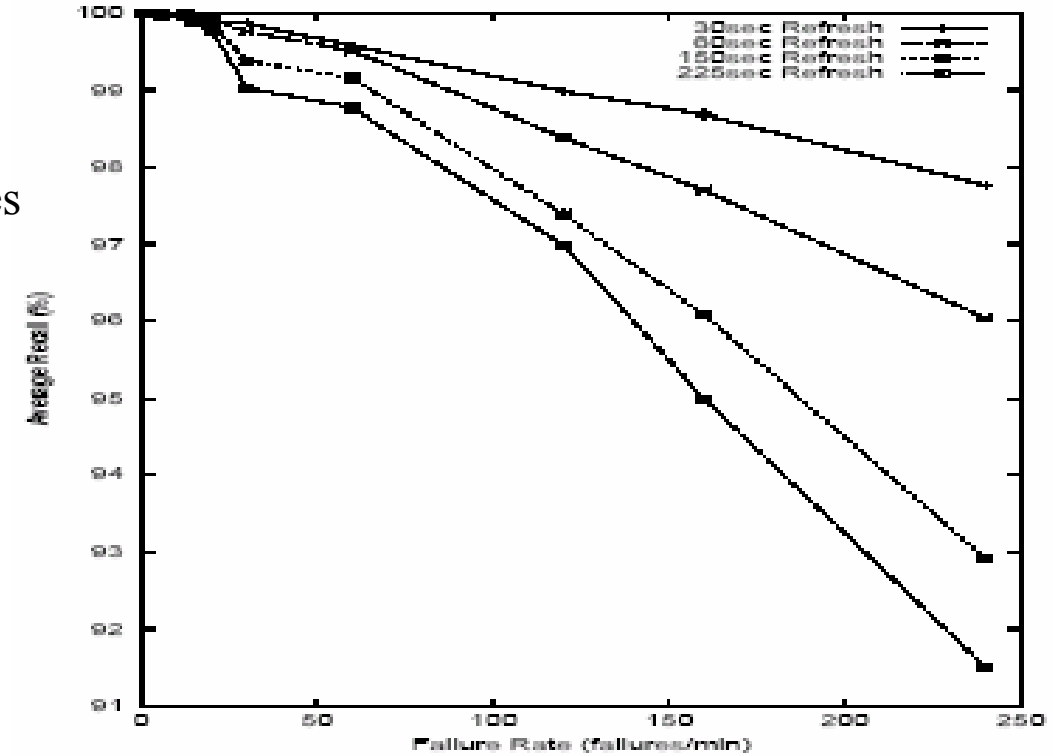
Average time to receive the 30th result tuple when both the size of the network and the load are scaled up.



Effects of soft state

- evaluate the robustness of the system in the face of node failures
- assume it takes 15s to detect a node failure
- all data is lost when a node fails
- periodically renew all tuples

Average recall for different refresh rates



Failure rate (failures/min)

Summary

- PIER - a distributed query engine based on widely distributed environments
- It is internally using a relational database format
- PIER aims to overcome the traditional database lack of scalability by using relaxed principles of consistency and a peer-to-peer like technology
- PIER is a read only system
- PIER is build on top of a DHT
- CAN - particular DHT, based on hashing data onto a d-dimensional space
- Implements DHT- distributed joins algorithms - e.g. symmetric hash join
- Operators are not enqueued: results are produced as quickly as possible (push), and data is transferred for the next operator (pull)
- Queries are optimized in order to minimize the response time

THANK YOU!