

The Data Cyclotron Query Processing Scheme^[1]

Huijing Deng

09.06.2010

[1] R. Goncalves, M. KERSTEN
CWI, Amsterdam, The Netherlands

Outline

- ***Motivation***
- Background Knowledge
- Data Cyclotron Architecture
 - System Structure
 - Interaction between layers
 - Algorithms
- Evaluation
- Summary
- Outlook

Distributed Query Processing

- What is distributed query processing?

The retrieval of data from different sites in a distributed database system is referred to as distributed query processing

- Example:

```
SELECT regionType
FROM Employee p , Postcode.PostcodeRegionType@postcode.com r
ON p.postcode = r.postcode
WHERE name = 'Anja'
```

regionType

rural

Database: Survey_People			
Table: Employee			
name	gender	age	postcode
Sam	m	18	37083
Frank	m	30	60111
Anja	f	30	42987

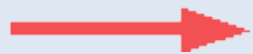
Database: Postcode	
Table: PostcodeRegionType	
postcode	regionType
60111	city
42987	rural
56100	rual

Distributed Query Processing

- Why we need distributed system ?
 - Changing bussiness requirement
 - Improve query throughput and reduce latency via parallel processing
- Key concerns
 - Cost of data transmission over network
 - Data transfer to and from disk
 - Data allocation
 - Workload behavior

Challenges

- Optimization was based on predictable workload, which is too ideal
- Presence of skewed and volatile workload
- Adaptation to changes in workload is complex and expensive



Self-organizing architecture ?

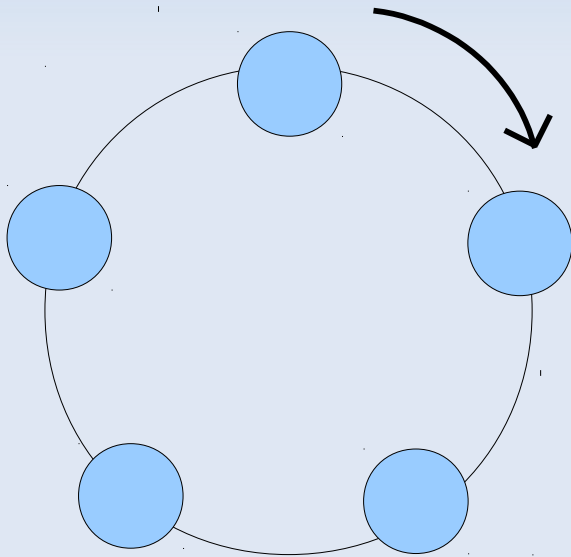
Opportunity on its way ...

- Hardware advances
 - bandwidth (10Gb/s Ethernet or even more)
- Remote Direct Memory Access technology
 - free CPU
 - reduce local I/O cost
 - **solve conflict between increase network speed and limited processing power of compute nodes**

!!! Data Cyclotron Scheme

What is Cyclotron ?

- Cyclotron: in physics, charged particle accelerator



- ▶ Virtual Storage Ring Topology
- ▶ Turbulent Data Movement
- ▶ RDMA

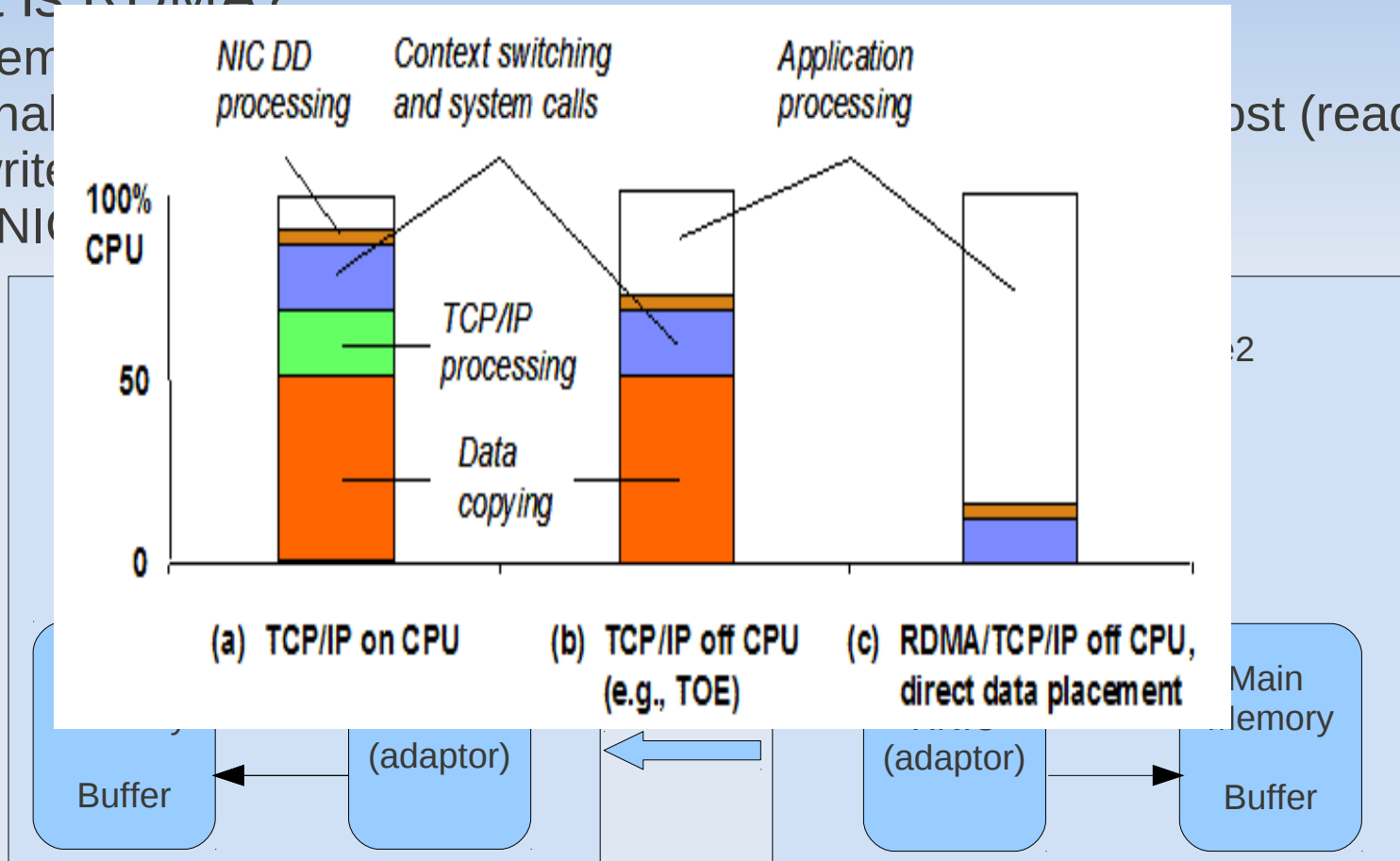
Outline

- Motivation
- ***Background Knowledge***
- Data Cyclotron Architecture
 - System Structure
 - Interaction between layers
 - Algorithms
- Evaluation
- Summary
- Outlook

RDMA Technology

- What is RDMA?

- Remote
- Enable
- write
- RNIC



ost (read and

2

Main memory

Buffer

Fit in Readily !

- Perfect match to benefit from RDMA
 - Buffer elements have big sizes (a dozen of hundred megabytes)
 - Connection between nodes should be Point-to-Point
- Target of Data Cyclotron
 - Workloads require huge blocks of data
 - Data transferred from node to node at high speed

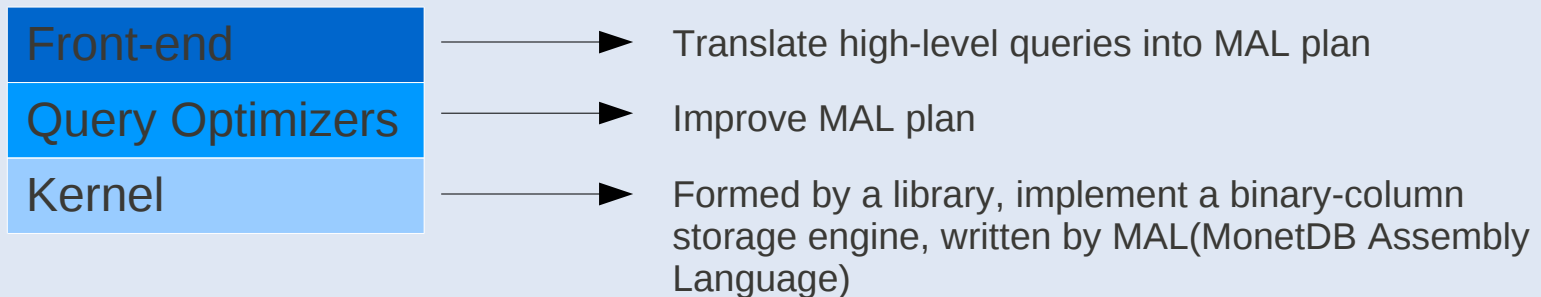


DBMS Selection

- Data Cyclotron is designed from the outset to extend an existing DBMS.
- MonetDB is a DBMS, whose inner working are well known and expertise is at hand

MonetDB

- Stores data column-wise in binary structures
- Relation in MonetDB is called BATs(Binary Associable Tables)
- 3-layer software stack

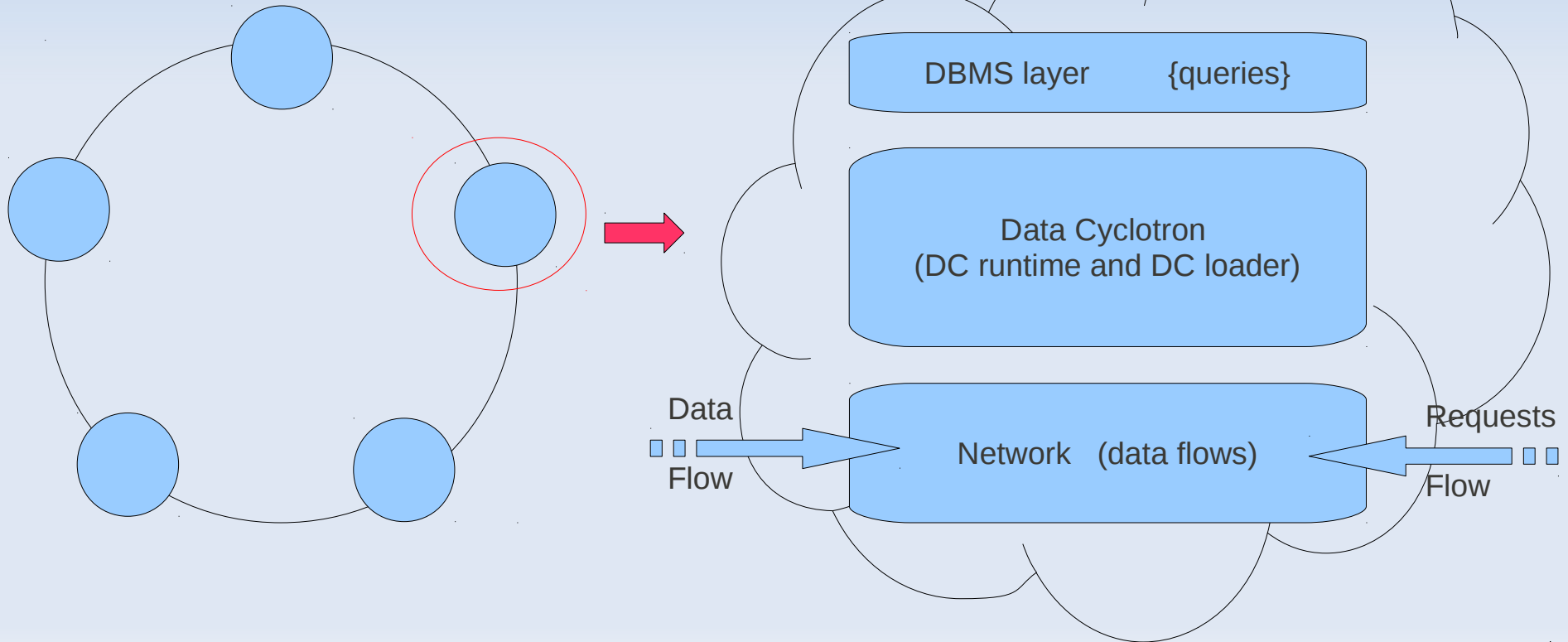


Outline

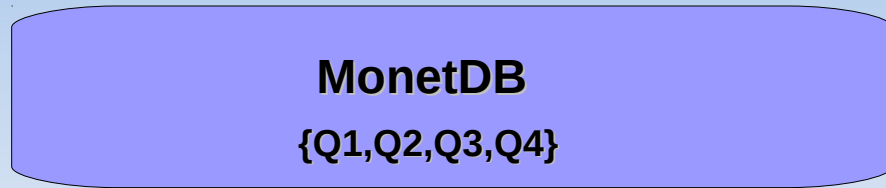
- Motivation
- Background Knowledge
- ***Data Cyclotron Architecture***
 - System Structure
 - Interaction between layers
 - Algorithms
- Evaluation
- Summary
- Outlook

System Structure

- Built around a ring with homogeneous nodes
- 3-layers in one single node: DBMS layer, Data Cyclotron layer, and Network layer.
- RDMA infrastructure in network layer



DBMS Layer

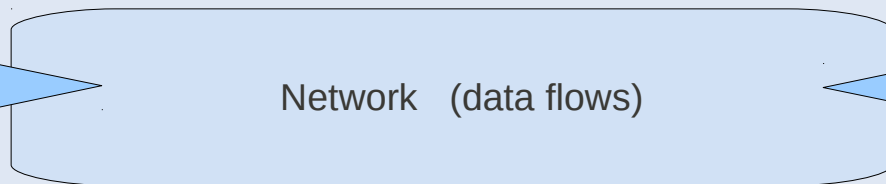
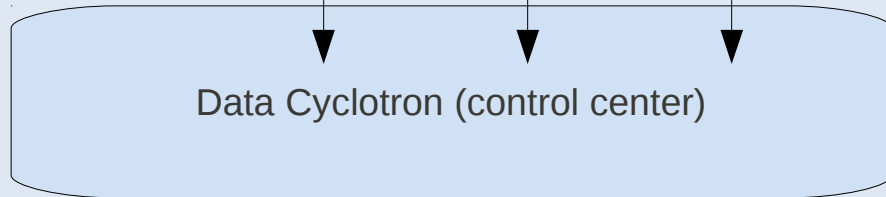


Queries-->MAL plan
-->Optimized plan

request

pin

unpin



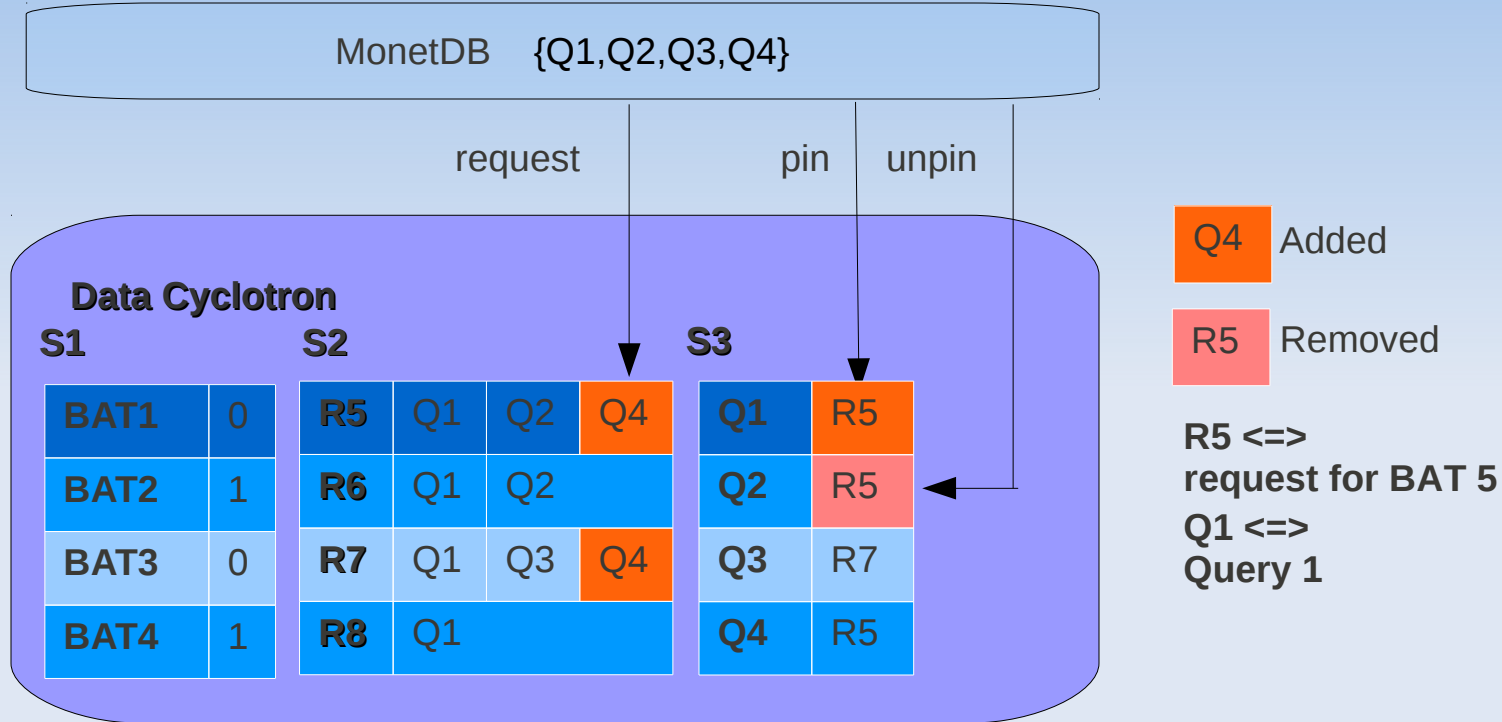
request(): identifies the required BATs

pin(),unpin(): lock and release the required BATs

Data Cyclotron Layer

- Control center, containing the Data Cyclotron runtime and a Data Cyclotron data loader
- Local data loader will become the owner of the BATs which are randomly assigned to the node.
- It has 3 catalogs
- It serves 3 message streams
- It carries out 3 main algorithms to handle requests(***Request Propagation algorithm***), BATs(***BAT Propagation algorithm***), and to manage the hot set in the ring(***Hot Set Management algorithm***).

Catalogs in Data Cyclotron Layer



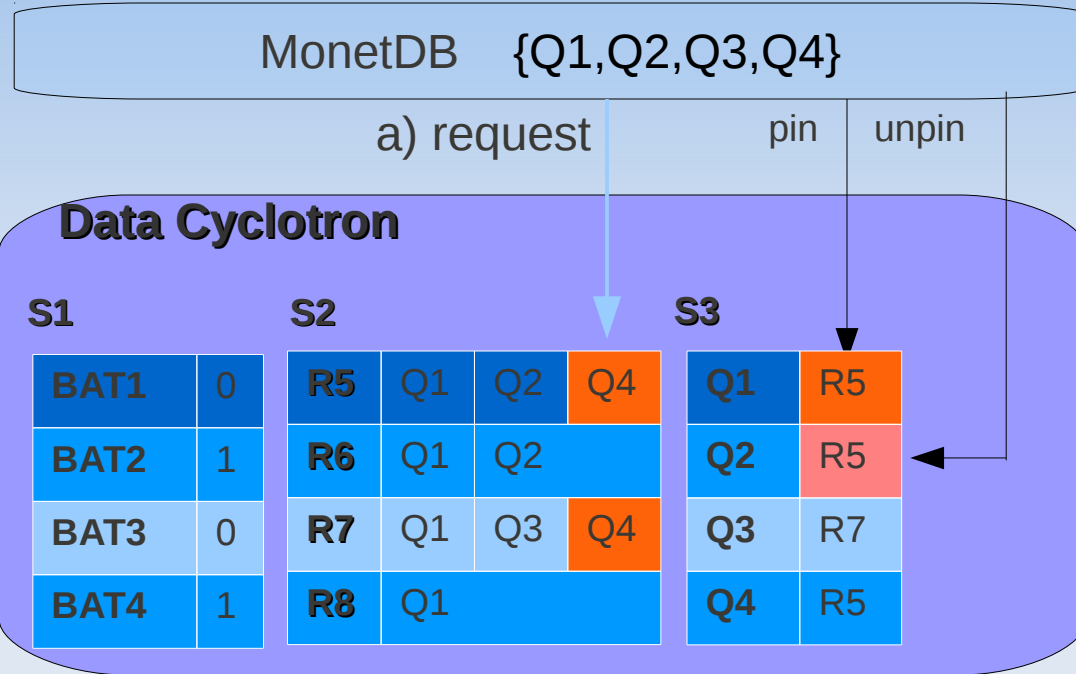
Three catalogs S1, S2, and S3:

S1--- contains information about all BATs owned by the local node

S2--- administer request for active queries

S3--- contains the identity of required BATs as indicated by the pin call

Message Streams in Data Cyclotron layer

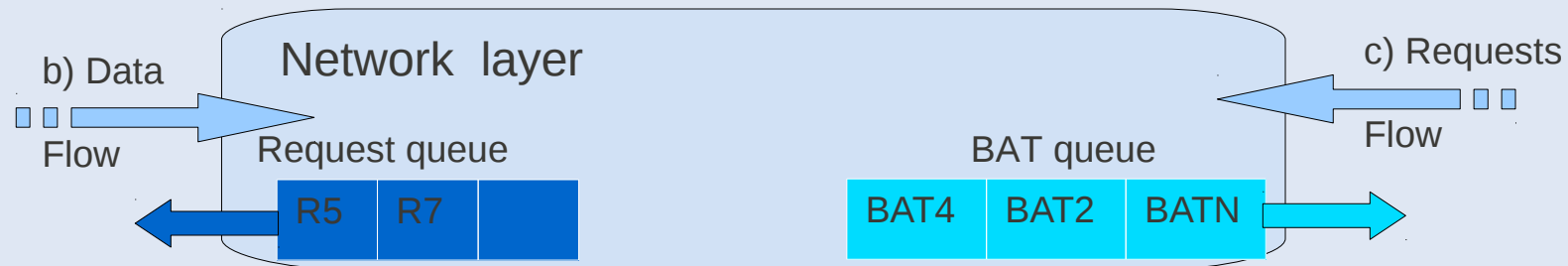


DC layer is the control center, it serves three message streams:

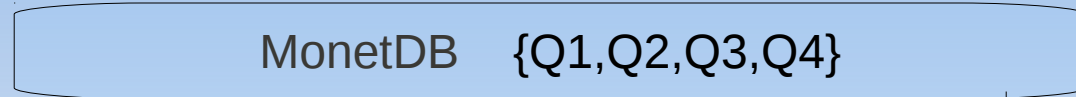
a) the request from the local MonetDB instance

b) the predecessor's BATs

c) the successor's requests from the network layer



Interaction between layers



BAT request pin unpin

Data Cyclotron

S1		S2			S3		
BAT1	0	R5	Q1	Q2	Q4	Q1	R5
BAT2	1	R6	Q1	Q2		Q2	R5
BAT3	0	R7	Q1	Q3	Q4	Q3	R7
BAT4	1	R8	Q1			Q4	R5

If BAT is owned by the local data loader--> retrieve from disk or local memory--> put into DBMS.

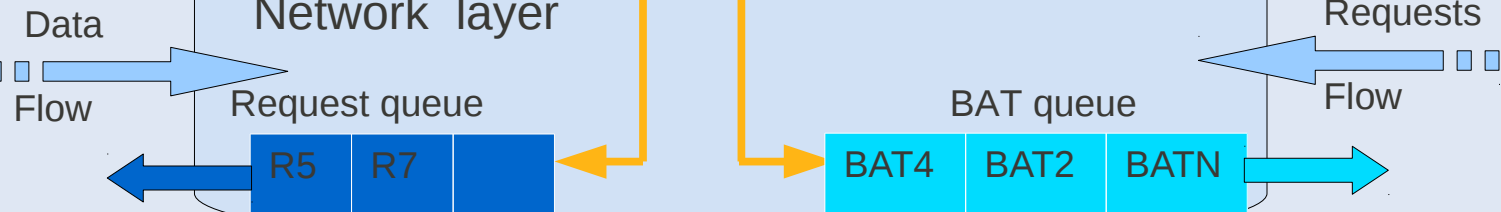
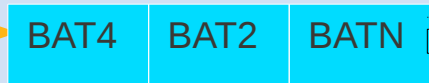
Otherwise:
 1) update S2.
 2) Sent BAT request to the ring towards its owner.
 3) The requested BAT will be sent towards the requesting node.

Send request Send BAT

Network layer

Request queue

BAT queue



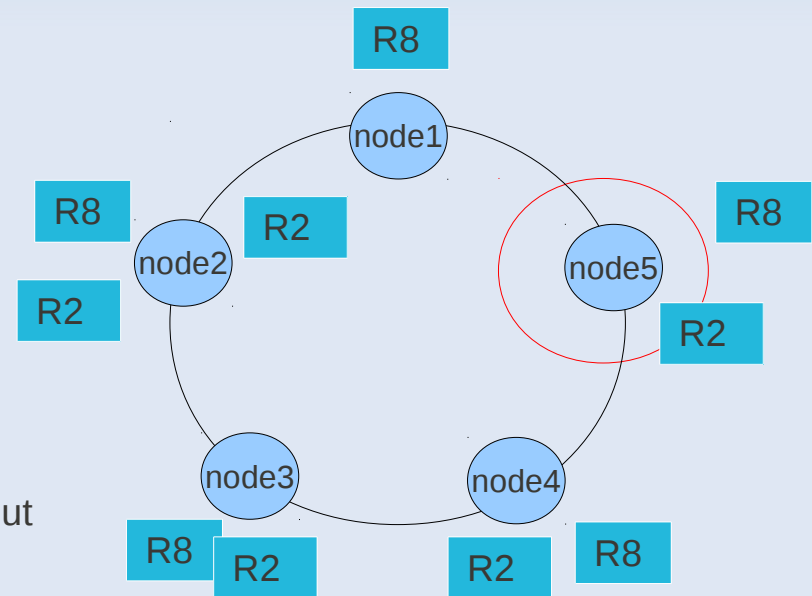
Request Propagation Algorithm

- The requests are handled by **Request Propagation algorithm**, there are 6 possible outcomes in this algorithm:

S1		S2				S3	
BAT1	0	R5	Q1	Q2	Q4	Q1	R5
BAT2	1	R6	Q1	Q2		Q2	R5
BAT3	0	R7	Q1	Q3	Q4	Q3	R7
BAT4	1	R8	Q1			Q4	R5

node5

- BAT request back to its originator node-->unregister request.
- current node is the BAT owner
 - BAT already in the ring--> ignore the request
 - BAT was not yet loaded, the ring is full-->mark the request as **pending**
 - BAT was not yet loaded, and ring is not full-->load the BAT
- current node is not the BAT owner nor the request originator, but has same request-->absorb request. Otherwise, forward request



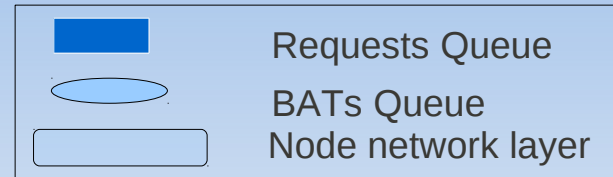
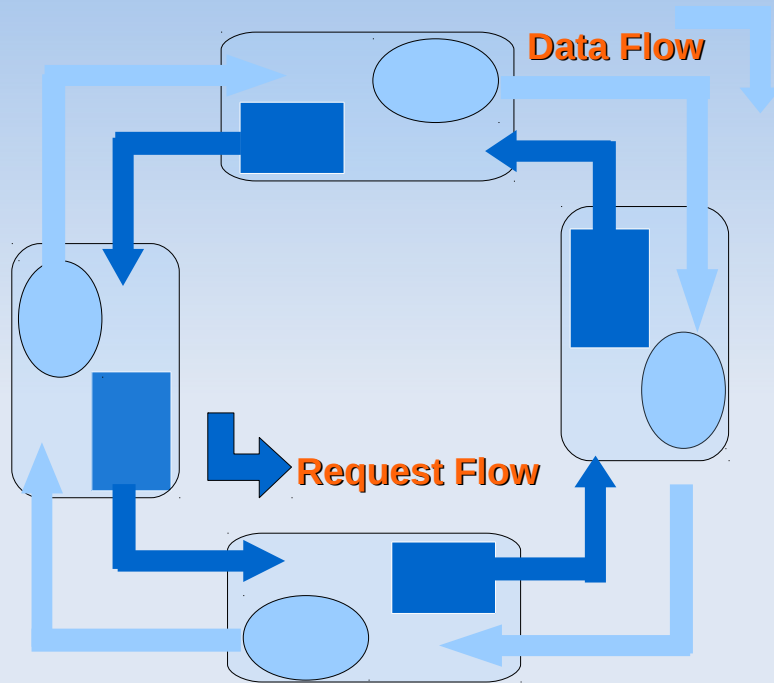
BAT Propagation Algorithm

- BAT Propagation algorithm handles the BATs received from the predecessor node
- For each received BAT, the algorithm searches for an corresponding request in S2. Once found, it checks the S3 and unblocks related queries blocked in a pin() call
- The memory region is then released by the unpin() call

Note: variables **hops** and **copies** generated by this algorithm are will be used later for hot set management.

S2				S3	
R5	Q1	Q2	Q4	Q1	R5
R6	Q1	Q2		Q2	R5
R7	Q1	Q3	Q4	Q3	R7
R8	Q1			Q4	R5

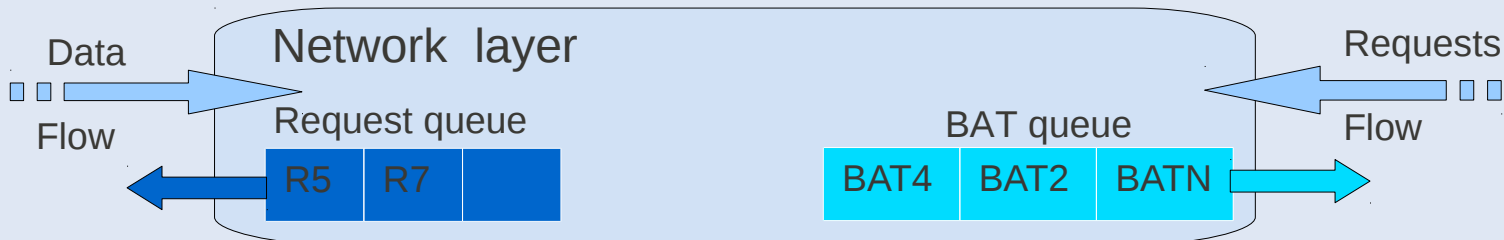
Storage Ring and Network Layer



Runtime functions:
 The network layer manages the stream of BAT message and the request message in the storage ring.
~~resend()~~ -- indicates package loss
~~loadAll()~~ -- executes BAT loads which were made as *pending* in Request Propagation algorithm

★ BAT message (*owner, bat_id, bat_size, loi, copies, hops, cycles*), calls for *hot data management* or *BAT Propagation algorithm*

★ BAT request message (*owner, req_id*), calls for *Request Propagation algorithm*

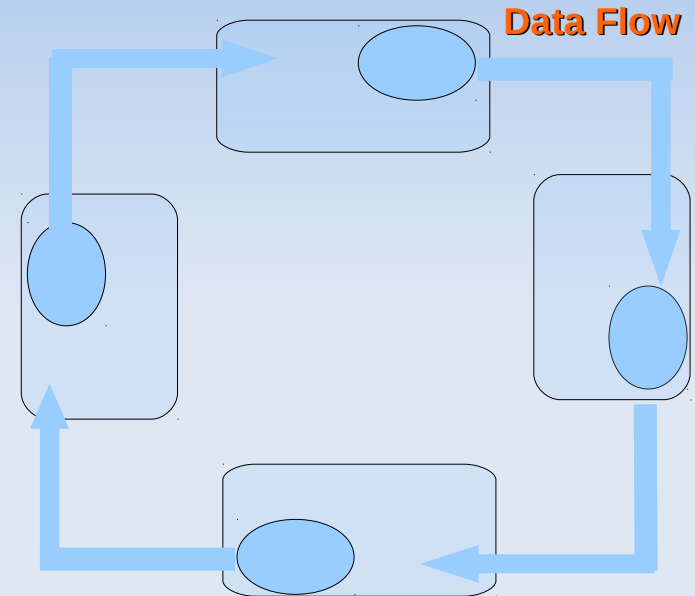


Hot Set Management

- Why we need hot set ?

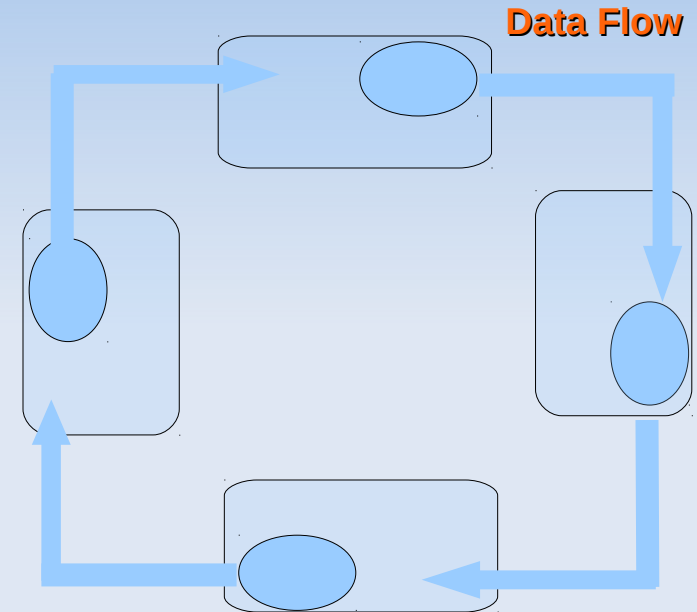
!! *Hot Set Management is a highlight in the Data Cyclotron Architecture, since:*

- It proposes a model to make data move in a ring nonstop, to facilitate the access to them with the assistance of RDMA technology.
- It provides metric to judge the "degree of fever" of the data (here data can be interpreted as BATs), and dynamically keeps the most frequently required data in the "Best Seller" shelf.



Hot Set Management Algorithm

- BATs in circulation are considered hot data. They can keep moving around the storage ring as long as it is hot.
- How hot the BATs would be is depending on its level of interest, LOI for short, which changes dynamically over time
- The LOI for a BAT is updated each time as it returns to its owner node, 3 factors: *the previous LOI*, the number of ***copies***, the number of ***hops***, and the number of ***cycles***
- Variables ***copies***, ***hops*** are updated in each node, the variable ***cycles*** is only updated by the BAT owner when it completes a cycle.



Hot Set Management Algorithm

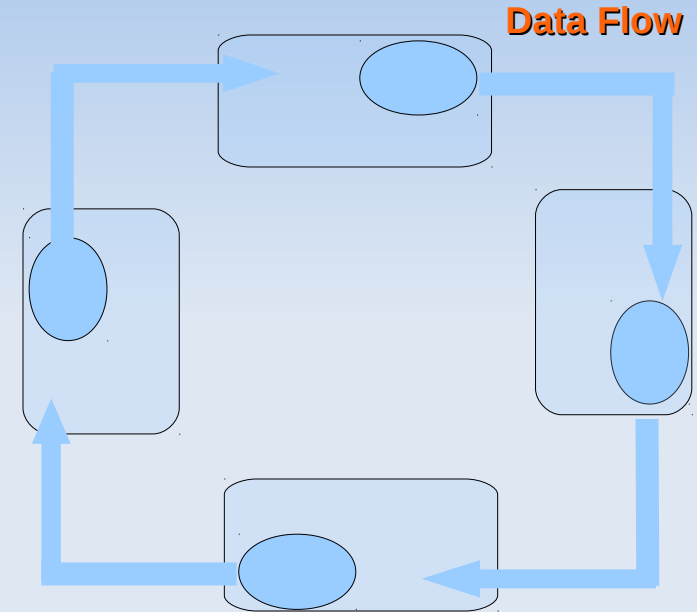
The new loi is calculated as follows:

$$CAVG = \text{copies} / \text{hops}$$

$$\text{newLOI} = \text{LOI} / \text{cycles} + CAVG$$

Then compare newLOI with the level of interest threshold $LOIT_n$.

If $\text{newLOI} < LOIT_n$ --> remove BAT,
otherwise keep in the ring.



!! threshold $LOIT_n$ -- minimum level of interest. Each node has its own $LOIT_n$, and its value is derived from the local BAT queue load.

Outline

- Motivation
- Background Knowledge
- Data Cyclotron Architecture
 - System Structure
 - Interaction between layers
 - Algorithms
- ***Evaluation***
- Summary
- Outlook

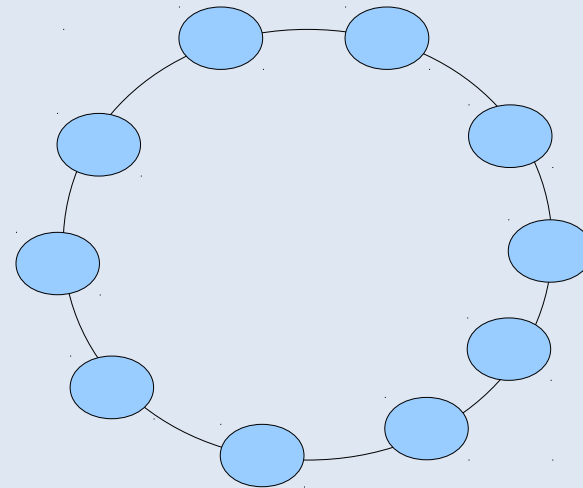
Experimental Setup

■ Configuration

- 10-nodes
- each node(linux system, Intel Core2 Quad CPU at 2.40 GHz, 8 GB RAM and 1 TB disk, 200MB BAT queue)
- interconnection(duplex-link 10 Gb/s bandwidth, 350 us delay, DropTail policy)
- RDMA-equipped clusters

■ Data Set

- 8GB(1000 BATs)
- BAT size(1 MB-10 MB)
- 0.8GB per node



Experimental Outline

- Micro-benchmark
- Three scenarios
 - Impact of LOITn on the query latency and throughput in a limited capacity ring
 - How Data Cyclotron run against skewed workloads with turbulent hot set
 - Data Cyclotron behavior for non-uniform access patterns
- Test with real benchmark TPC-H

Limited Ring Capacity

Experiment:

- 80queries/s on each node for 60s
- Queries request 1-5 random BATs
- 11 times, with LOIT_n=0.1 to 1.1 in step of 0.1

Assumption:

- All required data is available

-- NetQueryTime=
 $\Sigma(100\text{msec}--200\text{msec}) * \text{requiredBATs}$

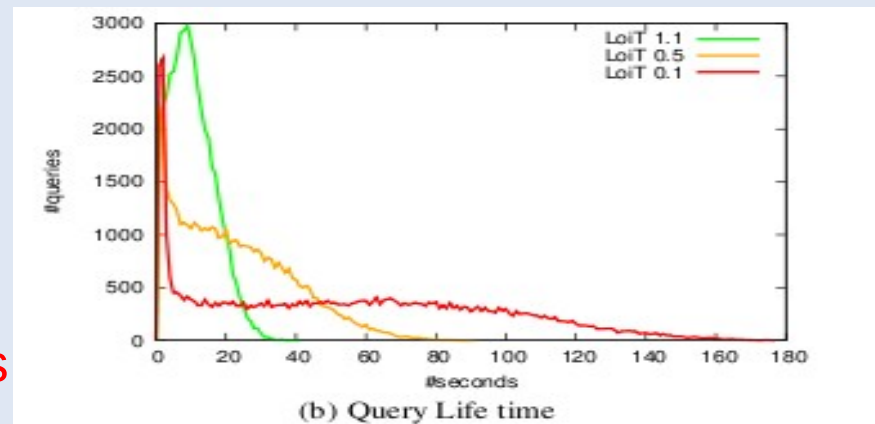
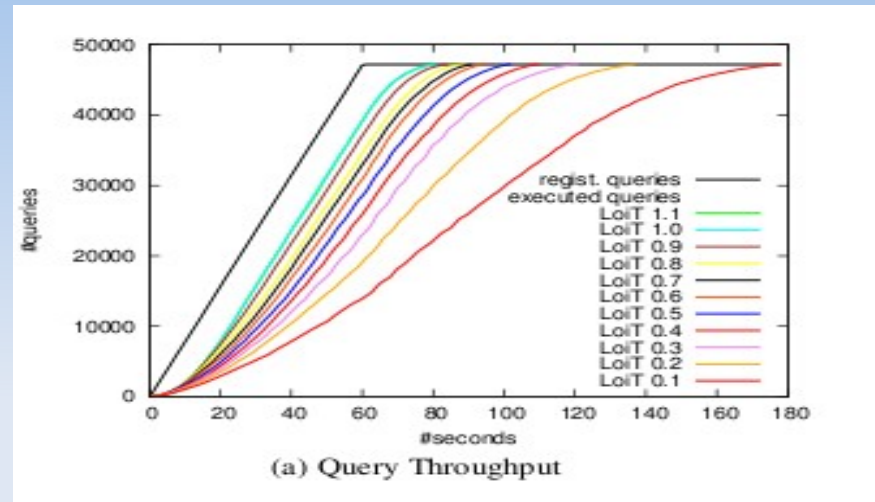


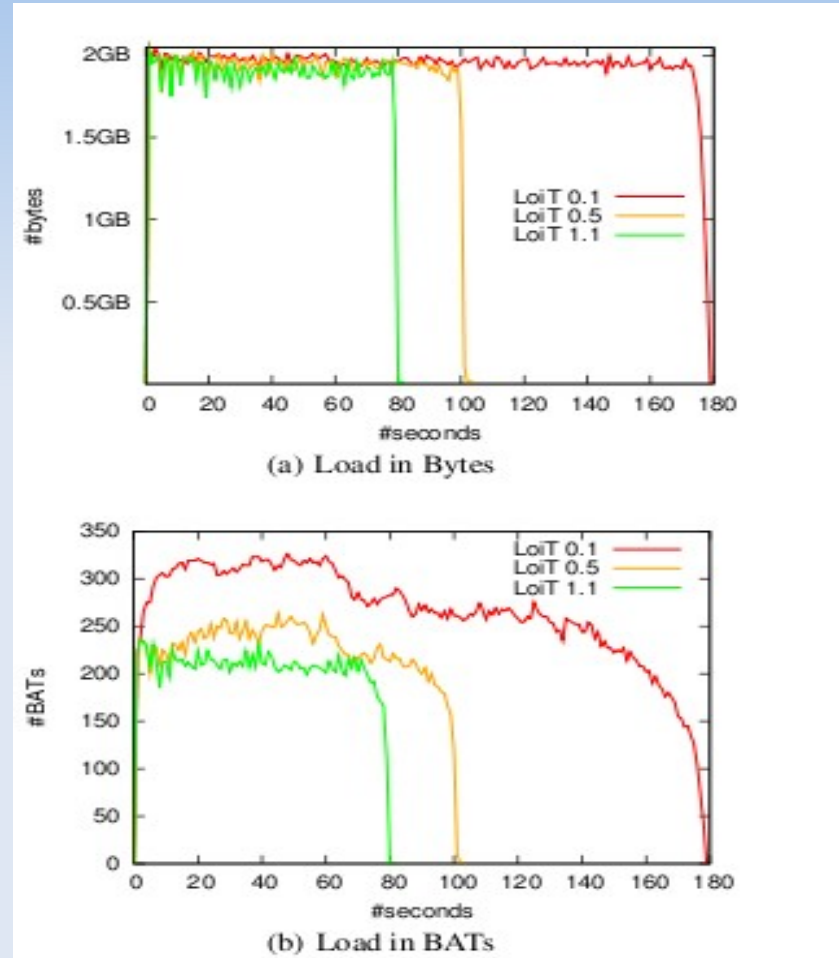
Figure 6: Multiple $LOIT_n$ levels

Limited Ring Capacity

- BAT length in the hot set over time
- Load of big BATs is postponed, queries waiting for these BATs stay pending until end

Conclusions:

- 1) $LOIT_n$ is inversely proportional to the local BAT queue load
- 2) $LOIT_n$ should dynamically adapt



Skewed Workloads

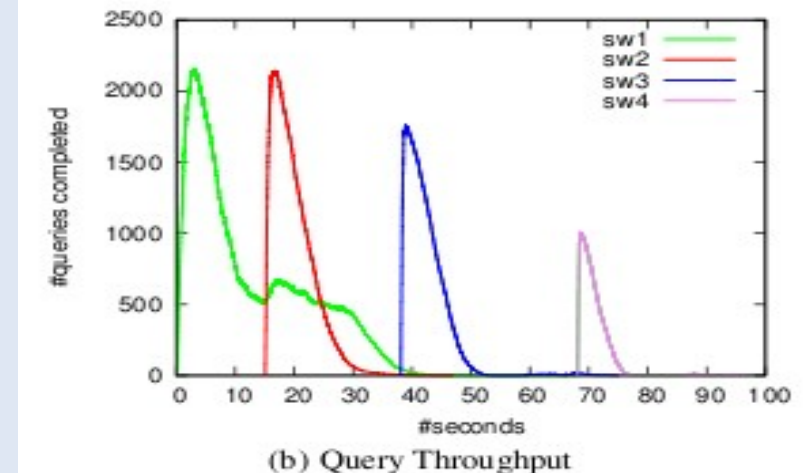
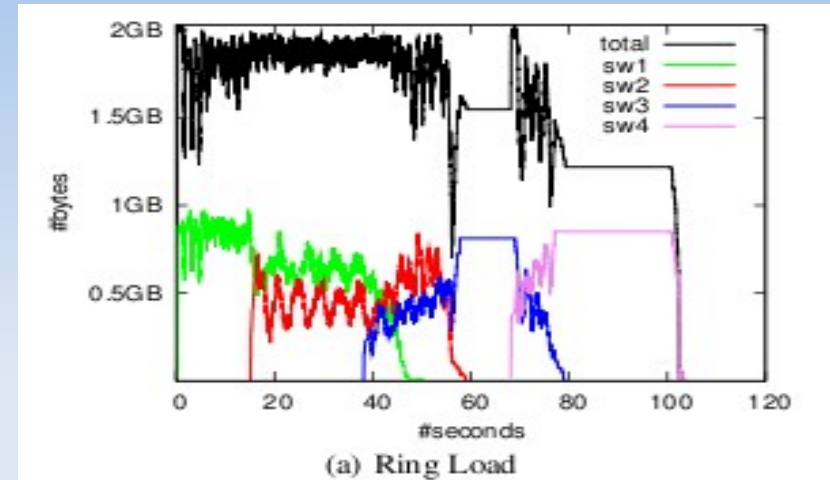
Experiment:

- 4 skewed workloads
- SW_i–D_i(disjont D_{H_i}), except D_{H4} is contained in D_{H1}

workload	SW1	SW2	SW3	SW4
skewed	3	5	7	9
start(sec)	0	15	37.5	67.5
end(sec)	30	45	67.5	97.5
queries/sec	200	300	400	500

Assumption:

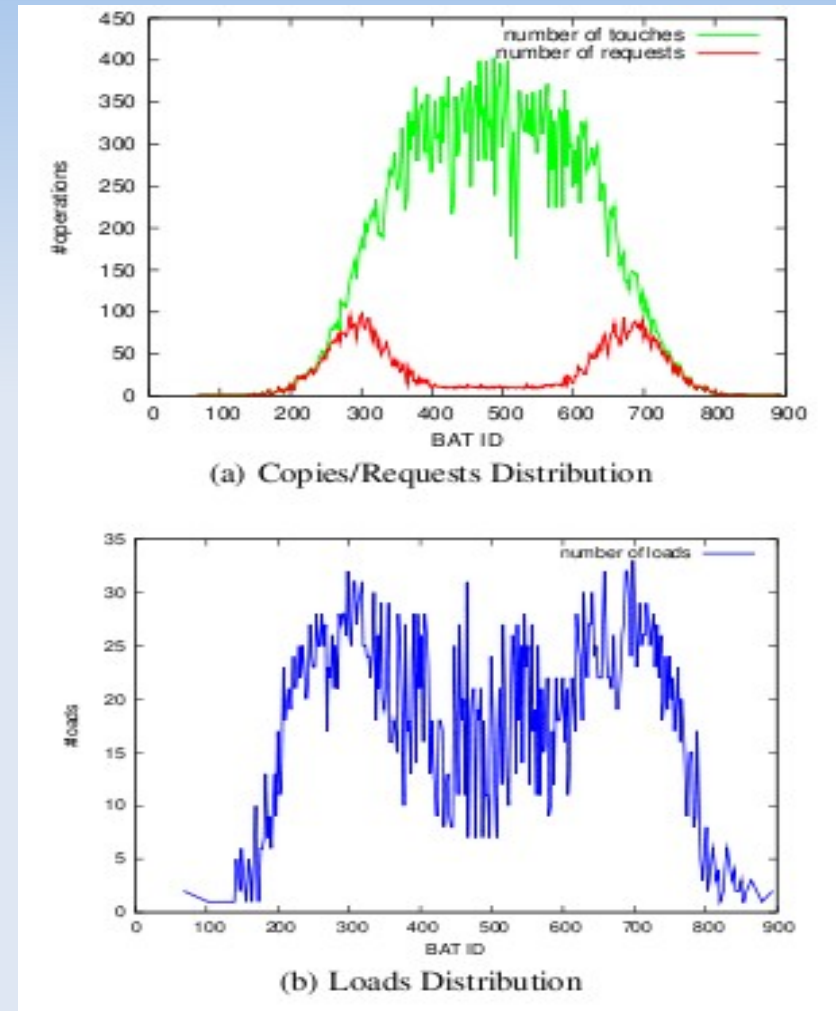
- If bufferload > 80% * capacity, increase LOIT_n one level
- If bufferload < 40% * capacity, decrease LOIT_n one level



Non-uniform Workloads

Experiment:

- 80queries/s on each node for 60s
- *Gaussian* data access distribution, centered around BAT id 500 with *standard deviation* of 50
- Uniform distribution for BAT sizes
- All nodes use the same distribution



Outline

- Motivation
- Background Knowledge
- Data Cyclotron Architecture
 - System Structure
 - Interaction between layers
 - Algorithms
- Evaluation
- ***Summary***
- Outlook

Summary

- What is Data Cyclotron ?
 - Novel architecture to implement distributed query processing
 - Self-organizing architecture to manage hot data set
- Highlights ?
 - Continuous data movement
 - Accomodate current workloads by adjusting hot data set
 - Load Balancing
- Integration with DBMS ?
 - simply by injecting calls into query execution plan

Outline

- Motivation
- Background Knowledge
- Data Cyclotron Architecture
 - System Structure
 - Interaction between layers
 - Algorithms
- Evaluation
- Summary
- ***Outlook***

Outlook

- Query chases requests
- Shared-nothing intra-query parallelism
 - sub-queries
 - reuse of query intermediate results
- Pulsating Rings

Thank you !