

Distributed Top-K Query Processing

by

Demetris Zeinalipour

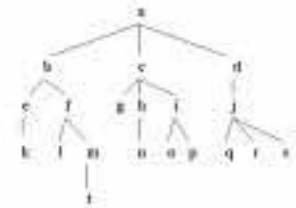
*Wednesday, November 16, 2005
Campus Building 12, 2nd Floor, #202,*

17:30-18:30



Characteristics of Modern Applications

- ***“Data is generated in a distributed fashion”*** e.g. sensor data, file-sharing data, Geographically Distributed Clusters)
- ***“Data is not always relational”***
(e.g. : hierarchical XML data, multimedia data, unstructured text data, streaming data,...)
- ***“Distributed Data is often outdated before it is ever utilized”***
(e.g. CCTV video traces, Internet ping data, sensor readings, weblogs, RFID Tags,...)
- ***“Transferring the Data to a centralized repository is usually more expensive than storing it locally”***



Our Framework

- ***Why design algorithms and systems that a' priori organize information in centralized repositories?***
- **Our Approach: “On-Demand Search and Retrieval”**
 - *Data remains **in-situ** (at the generating site).*
 - *When Users want to search/retrieve some information they perform **on-demand** queries.*
 - *We evenly distribute the load across the remote sites.*
- **Challenges:**
 - ***Minimize the utilization** of the communication medium*
 - *Resolve Queries in **parallel***
 - ***Exploit the network** structure between the distributed nodes*
 - *Number of Answers might be very large → **Focus on Top-K***



Presentation Outline

1. Introduction to Top-K Query Processing
2. Related Work & Algorithms
3. Queries using Distributed Scores
(The Threshold Join Algorithm (TJA))
4. Experimental Evaluation using our
Middleware Testbed.
5. Conclusions & Future Work.



Distributed Top-K Query Processing

TOP-k Query Objectives:

- To find the **k highest ranked** answers to a user defined aggregate similarity (scoring) function.
e.g. **SUM**, MAX, MIN, COUNT, PRODUCT, ...
- Minimize some **cost metric** associated with the retrieval of the correct answers (e.g. I/Os, network traffic).

Cost Metric in a Distributed Environment?

→ **Bandwidth** → **Latency**

In a Sensor Environment the energy consumption of sending 1 byte \approx 1120 CPU instructions) Source: The RISE (Riverside Sensor) (NetDB'05, IPSN'05 Demo, SECON'05)



Distributed Top-K Query Processing

Motivating Example

- Assume that we have a cluster of **n=5 servers**. Each server maintains locally **m=5 web pages**.
- When a web page is accessed by a client, a server increases a local **hit counter** by one.
- **TOP-1 Query:** “Which Webpage has the highest hit rate **Score(o_i)** across all servers?”
- **Score(o_i)** can only be calculated if we combine the readings from all 5 servers.

Local score


URL

	v1	v2	v3	v4	v5	TOP-5
	o3, 99	o1, 91	o1, 92	o3, 74	o3, 67	o3,405
	o1, 66	o3, 90	o3, 75	o1, 56	o4, 67	o1, 363
	o0, 63	o0, 61	o4, 70	o2, 56	o1, 58	o4, 207
	o2, 48	o4, 07	o2, 16	o0, 28	o2, 54	o0, 188
	o4, 44	o2, 01	o0, 01	o4, 19	o0, 35	o2, 175

m

n

Complete Scores



Distributed Top-K Query Processing

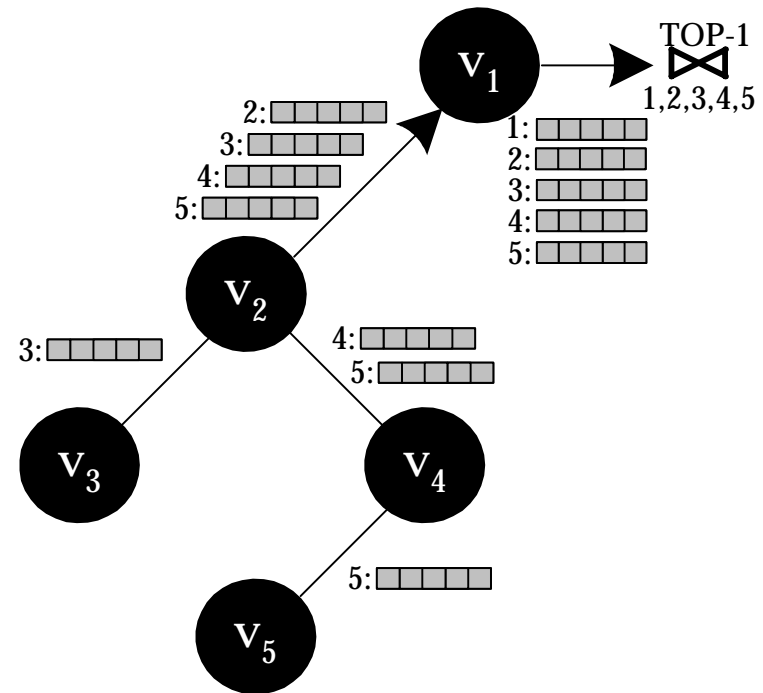
Other Applications

- **Sensor Networks:** Each sensor locally maintains a sliding window of the last m readings (i.e. m (ts, val) pairs).
Q: Find when did we have the highest average temperature across all sensors.
- **Collaborative Spam Detection Networks:** Every peer tags IPs (mail sender:msgID) as {spam | ham}.
Q: Find the 5 IPs with the highest spam reports across all peers.
- **Might be applicable in other distributed domains as well, e.g. Grids, Vehicular Networks, etc...**



Naïve Solution: Centralized Join (CJA)

- Each Node sends all its local scores (list)
- Each intermediate node forwards all received lists
- The Gnutella Approach



Drawbacks

- Overwhelming amount of messages.
- Huge Query Response Time

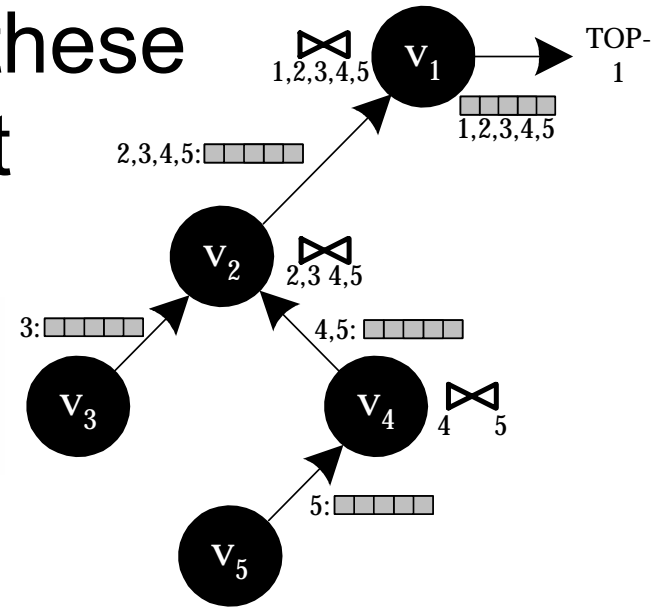


Improved Solution: Staged Join (SJA)

- Aggregate the lists before these are forwarded to the parent using:

$$R(v_i) = list(v_i) \bowtie \left(\bigwedge_{\forall j \in children(v_i)} list(v_j) \right)$$

- This is essentially the TAG approach (Madden et al. OSDI '02)



- Advantage:** Only (n-1) messages
- Drawback:** Still sending everything!

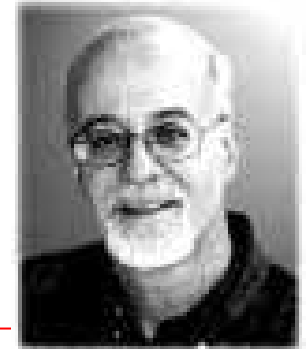


The Threshold Algorithm (Not Distributed)

Fagin's* Threshold Algorithm (TA):

Long studied and well understood.

* Concurrently developed by 3 groups



TA Algorithm

- 1) Access the n lists in parallel.
- 2) While some object o_i is seen, perform a **random access** to the other lists to find the complete score for o_i .
- 3) Do the same for all objects in the current row.
- 4) Now compute the threshold τ as the **sum of scores** in the current row.
- 5) The algorithm stops after K objects have been found with a score above τ .



The Threshold Algorithm (Example)

v1	v2	v3	v4	v5	TOP-K
o3, 99	o1, 91	o1, 92	o3, 74	o3, 67	O3, 405
o1, 66	o3, 90	o3, 75	o1, 56	o4, 67	O1, 363
o0, 63	o0, 61	o4, 70	o2, 56	o1, 58	O4, 207
o2, 48	o4, 07	o2, 16	o0, 28	o2, 54	
o4, 44	o2, 01	o0, 01	o4, 19	o0, 35	

Iteration 1 Threshold

$$\tau = 99 + 91 + 92 + 74 + 67 \Rightarrow \tau = 423$$

Have we found K=1 objects with a score above τ ?

=> NO

Iteration 2 Threshold

$$\tau \text{ (2nd row)} = 66 + 90 + 75 + 56 + 67 \Rightarrow \tau = 354$$

Have we found K=1 objects with a score above τ ?

=> YES!



The Threshold Algorithm (Not Distributed)

Why is the threshold correct?

Because the threshold essentially gives us the maximum Score for the objects not seen ($\leq \tau$)

Advantages:

- The number of object accessed is minimized!

Why Not TA in a distributed Environment?

Disadvantages:

Each object is accessed individually (random accesses)

- A huge number of **round trips** (phases)
- **Unpredictable Latency** (Phases are sequential)
- Also In-network **Aggregation not possible**



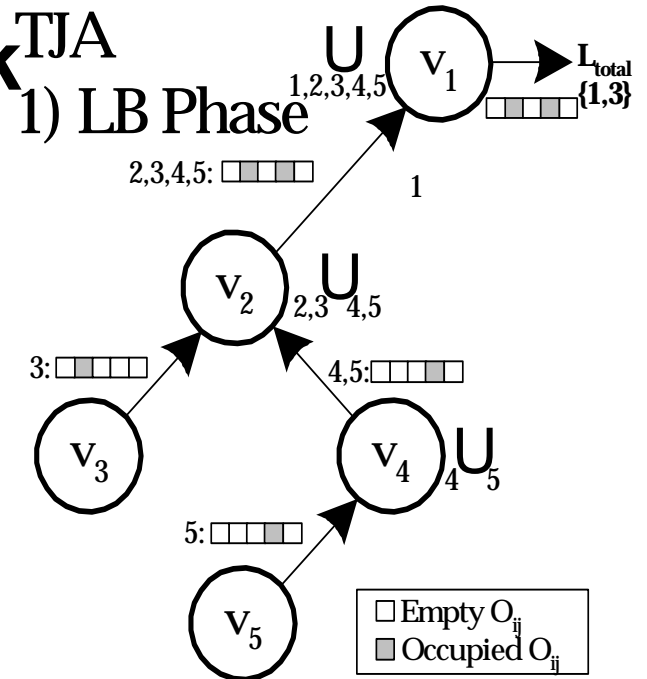
Threshold Join Algorithm (TJA)

- TJA is our **3-phase algorithm** that **minimizes** the number of transmitted objects and hence the **utilization of the communication channel**.
- **How does it work:**
 1. **LB Phase:** Ask each node to send the K (locally) highest ranked results.
The union of these results defines a threshold τ .
 1. **HJ Phase:** Ask each node to transmit everything above this threshold τ .
 2. **CL Phase:** If at the end we have not identified the complete score of the K highest ranked objects, then we perform a cleanup phase to identify the complete score of all incompletely calculated scores.



Step 1 - LB (Lower Bound) Phase

- Each node sends its **top-k**^{TJA} results to its parent.
- Each intermediate node performs a **union** of all received lists (denoted as τ):



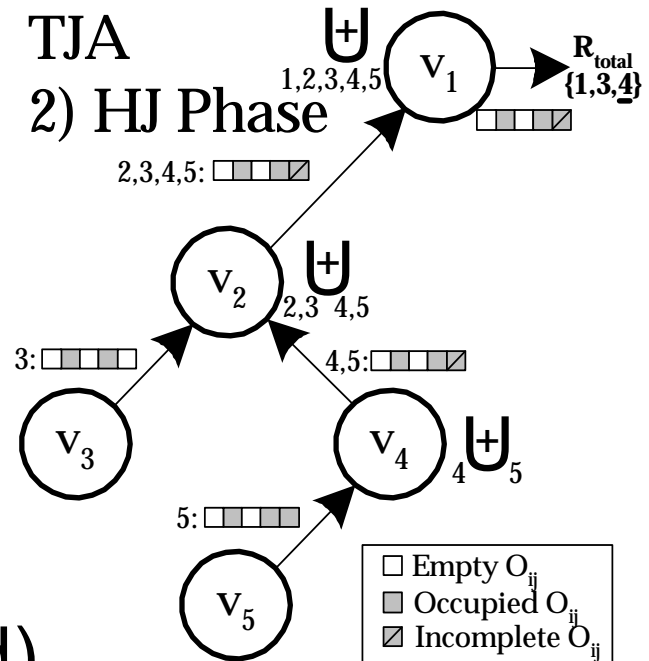
Query: TOP-1

v1	v2	v3	v4	v5	LB
o3, 99	o1, 91	o1, 92	o3, 74	o3, 67	{o3, o1}
o1, 66	o3, 90	o3, 75	o1, 56	o4, 67	
o0, 63	o0, 61	o4, 70	o2, 56	o1, 58	
o2, 48	o4, 07	o2, 16	o0, 28	o2, 54	
o4, 44	o2, 01	o0, 01	o4, 19	o0, 35	



Step 2 – HJ (Hierarchical Join) Phase

- Disseminate τ to all nodes
- Each node sends back everything with score above all objectIDs in τ .
- Before sending the objects, each node tags as incomplete scores that could not be computed exactly (upper bound)



v1	v2	v3	v4	v5
o3, 99	o1, 91	o1, 92	o3, 74	o3, 67
o1, 66	o3, 90	o3, 75	o1, 56	o4, 67
o0, 63	o0, 61	o4, 70	o2, 56	o1, 58
o2, 48	o4, 07	o2, 16	o0, 28	o2, 54
o4, 44	o2, 01	o0, 01	o4, 19	o0, 35

HJ

o3, 405	} →	Complete
o1, 363		
o4', 354		



Step 3 – CL (Cleanup) Phase

Have we found K objects with a complete score?

Yes: The answer has been found!

No: Find the *complete score* for each incomplete object (all in a single batch phase)

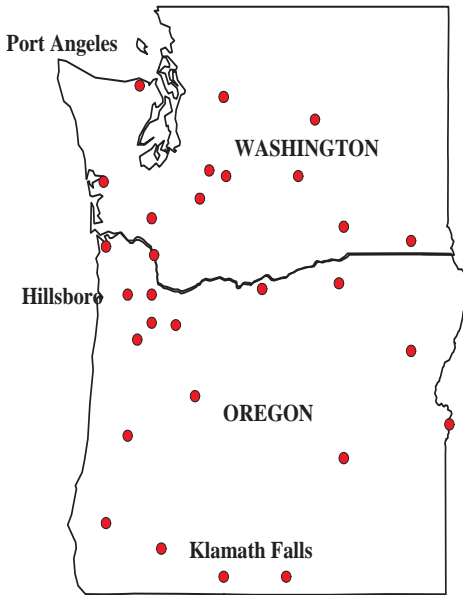
- **CL ensures correctness!**
- **This phase is rarely required in practice.**

v1	v2	v3	v4	v5	TOP-5
o3, 99	o1, 91	o1, 92	o3, 74	o3, 67	o3, 405
o1, 66	o3, 90	o3, 75	o1, 56	o4, 67	o1, 363
o0, 63	o0, 61	o4, 70	o2, 56	o1, 58	o4, 207
o2, 48	o4, 07	o2, 16	o0, 28	o2, 54	o0, 188
o4, 44	o2, 01	o0, 01	o4, 19	o0, 35	o2, 175



Experimental Evaluation

- We implemented a real middleware system (using a binary protocol)
- We tested our implementation with a network of 1000 real nodes using 75 Linux workstations.
- We use a trace driven experimentation methodology.



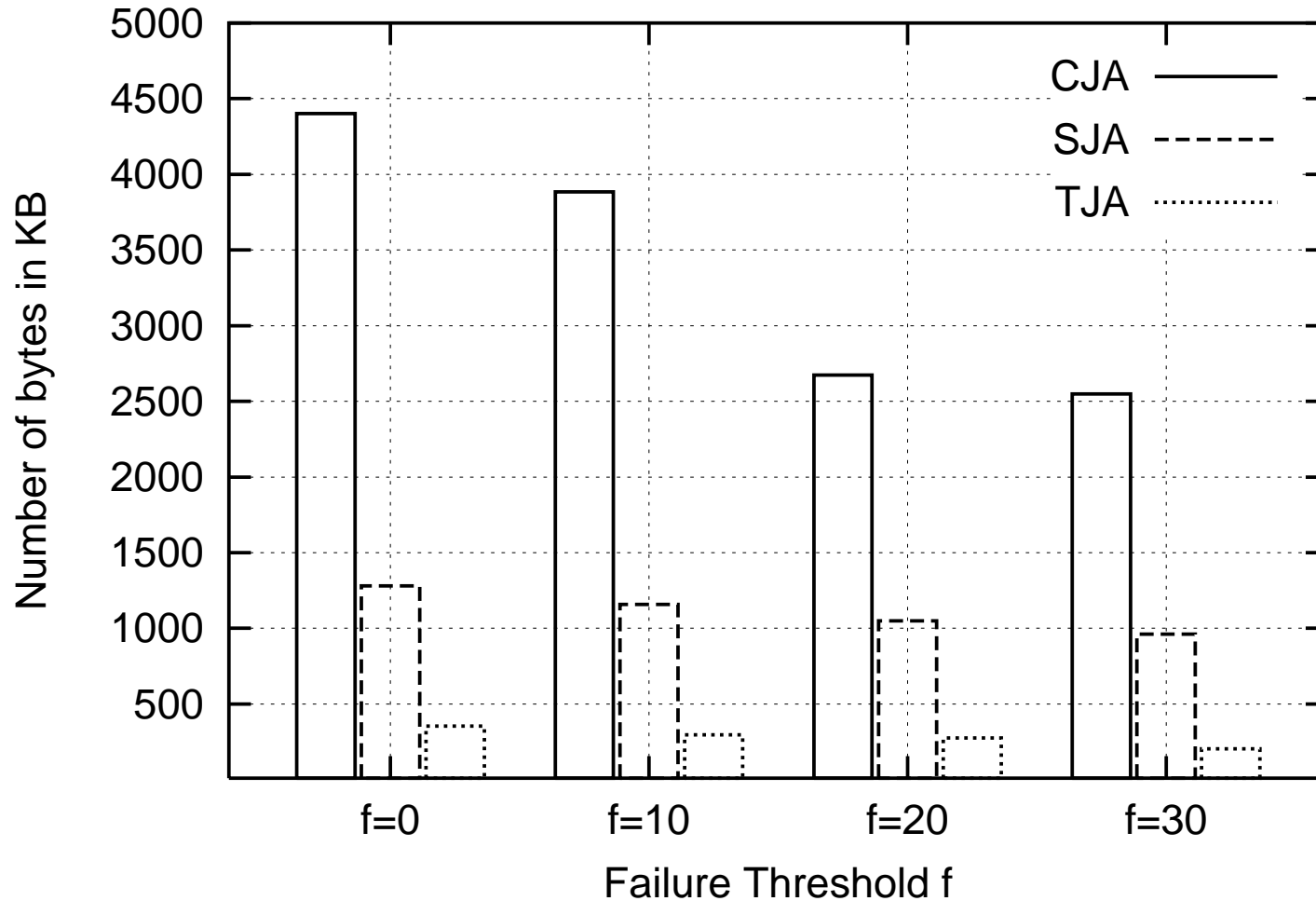
For the results presented in this talk:

- **Dataset:** Environmental Measurements from 32 atmospheric monitoring stations in Washington & Oregon. (2003-2004)
- **Query:** K timestamps on which average temperature across all stations was maximum
- **Network:** Random Graph (degree=4, diameter 10)
- **Evaluation Criteria:** i) **Bytes**, ii) **Time**, iii) **Messages**



Experimental Results

Bytes using the Atmon Dataset

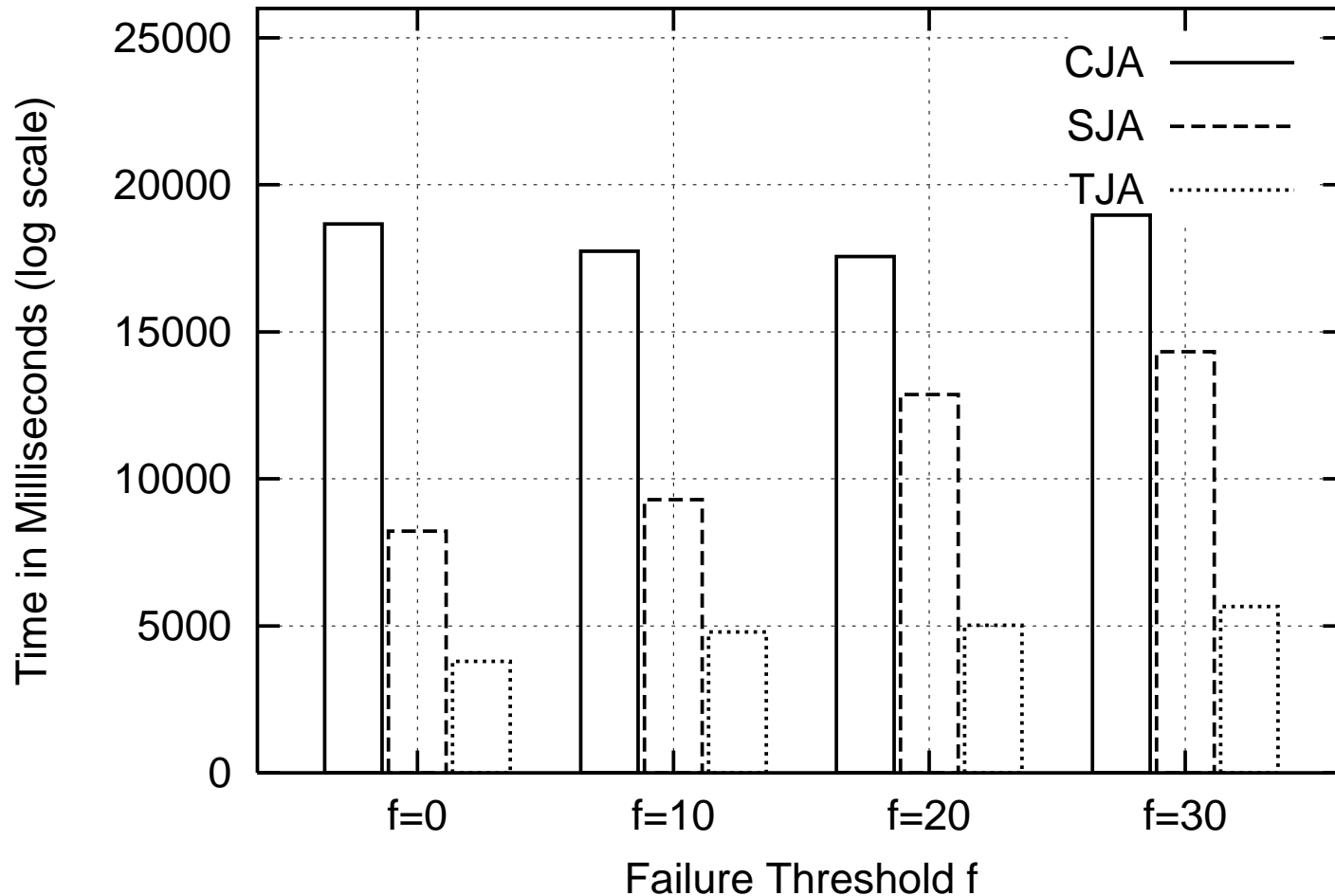


TJA One order of magnitude less bytes than the Centralized Algorithm!



Experimental Results

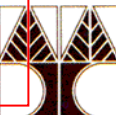
Time using the Atmon Dataset



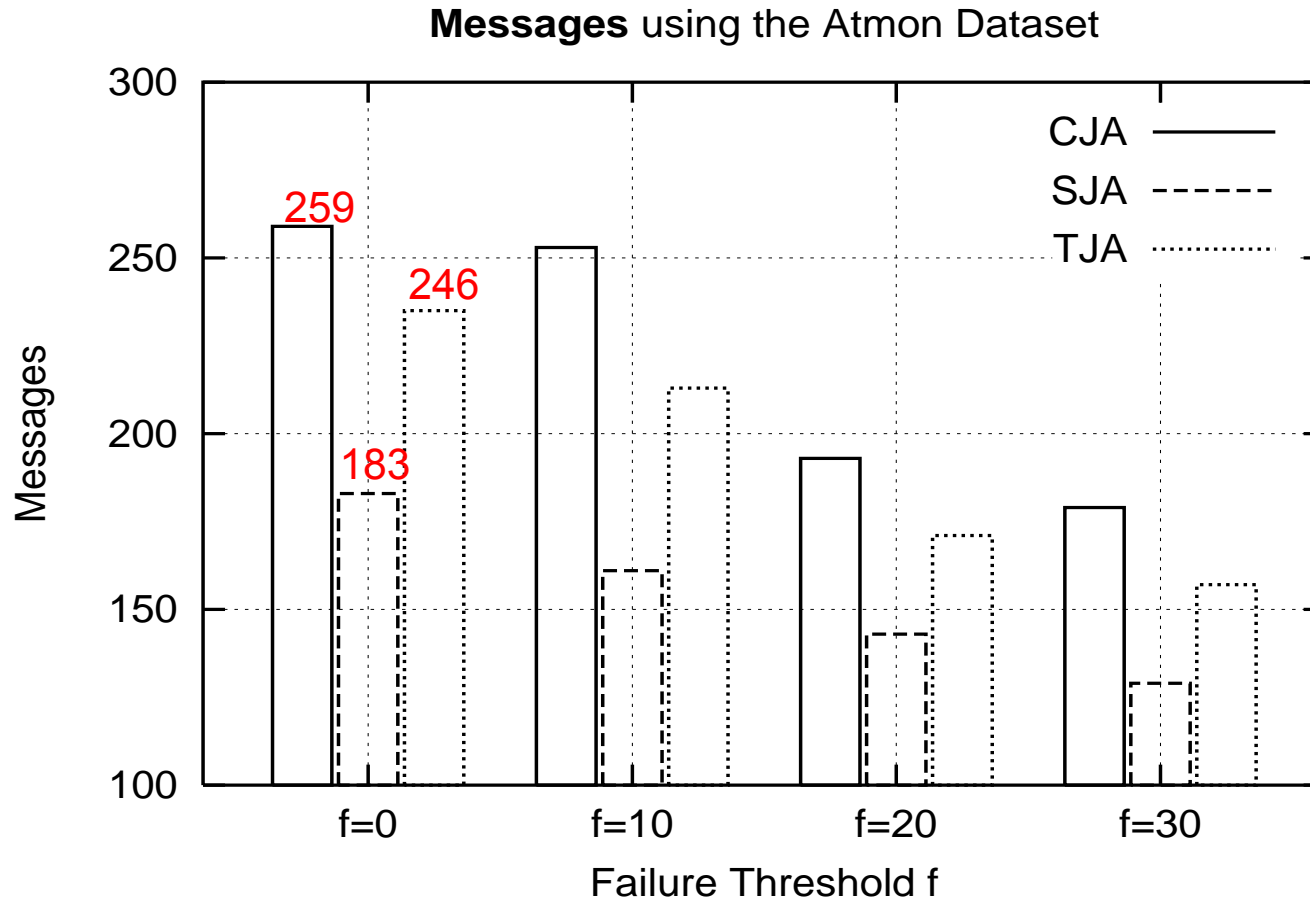
TJA: 3,797ms [LB:1059ms, HJ:2730ms, CL:8ms]

SJA: 8,224ms

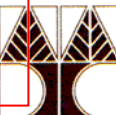
CJA: 18,660ms



Experimental Results



Although TJA consumes more messages than SJA. However these are smaller messages



The TPUT Algorithm (Distributed)

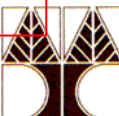
The TPUT Algorithm (Cao et al, PODC'04)

(developed concurrently with TJA for a client/server environment – was also published earlier ☹)



TPUT (Three Phase Uniform Threshold)

- 1) Fetch K first entries from all n lists. Define threshold τ as $\tau = (K\text{th lowest partial score} / n)$.
 τ (the uniform threshold) is then disseminated to all nodes.
- 2) Each node sends any pair which has a score above τ .
- 3) If we found the complete score for less than K objects then we perform a random access for all incomplete objects (in a single batch phase - same with TJA)



The TPUT Algorithm



v1	v2	v3	v4	v5	TOP-1
o3, 99	o1, 91	o1, 92	o3, 74	o3, 67	o1=183, o3=240 o1=363 o2'=158 o4'=137 o0'=124
o1, 66	o3, 90	o3, 75	o1, 56	o4, 67	
o0, 63	o0, 61	o4, 70	o2, 56	o1, 58	
o2, 48	o4, 07	o2, 16	o0, 28	o2, 54	
o4, 44	o2, 01	o0, 01	o4, 19	o0, 35	

Q: TOP-1 - - - P1 **— —** P2 P3

Phase 1 : $o1 = 91+92 = 183$, $o3 = 99+67+74 = 240$

$\tau = (Kth \text{ highest score (partial)} / n) \Rightarrow 240 / 5 \Rightarrow \tau = 48$

Phase 2 : Have we computed K exact scores ?

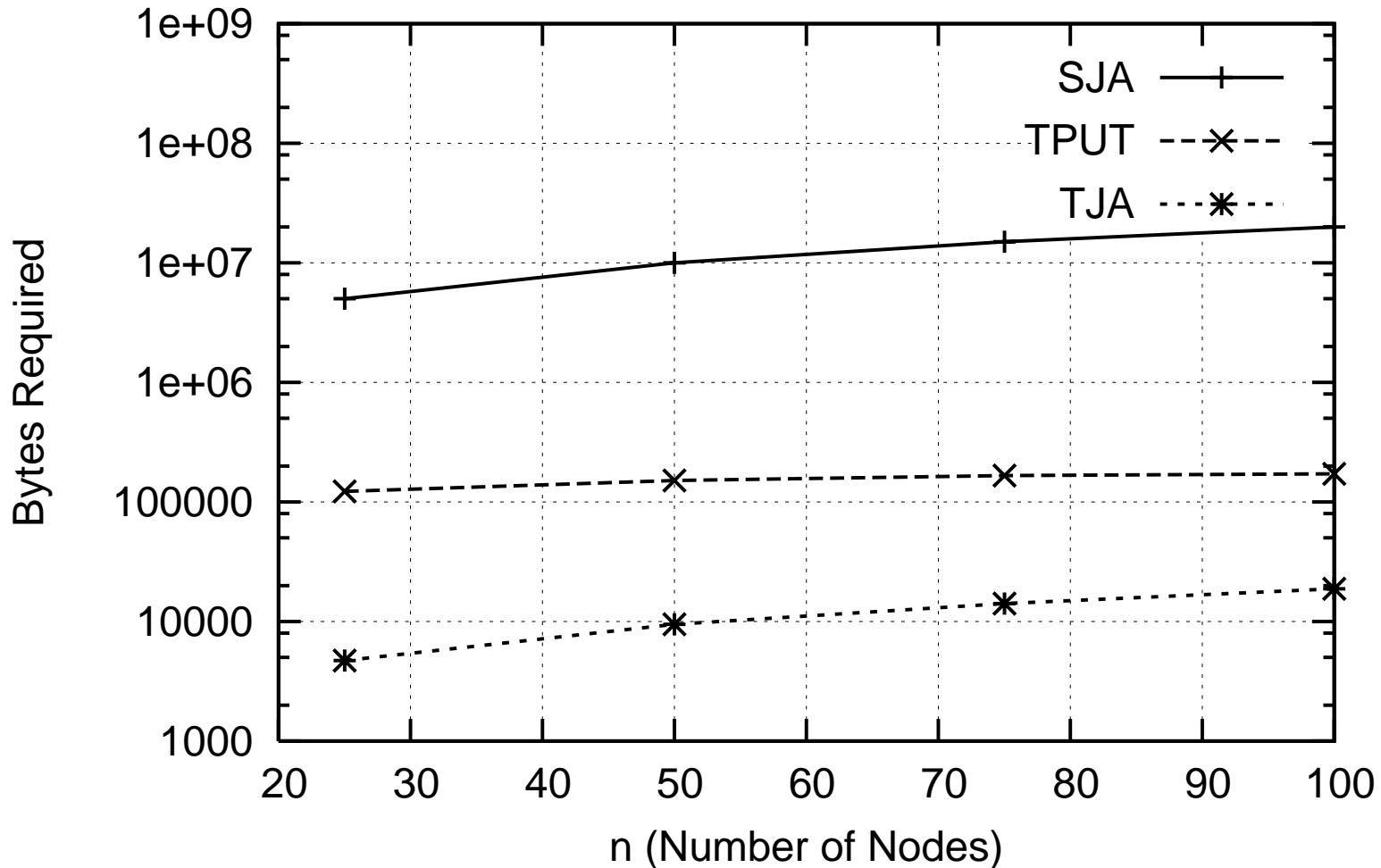
Computed Exactly: [**o3**, o1] Incompletely Computed: [o4,o2,o0]

Drawback: The threshold is too coarse (uniform)



TJA vs. TPUT

Bytes Required for Distributed Top-K Algorithms
(Star Topology, K=5, m=25K)



Conclusions

- Distributed Top-K Query Processing is a new area with many new challenges and opportunities!
- We showed that the TJA provides an order of magnitude improvement over its competitors.
- We believe that our algorithm will be a useful component in Query Optimization engines of future Database systems.



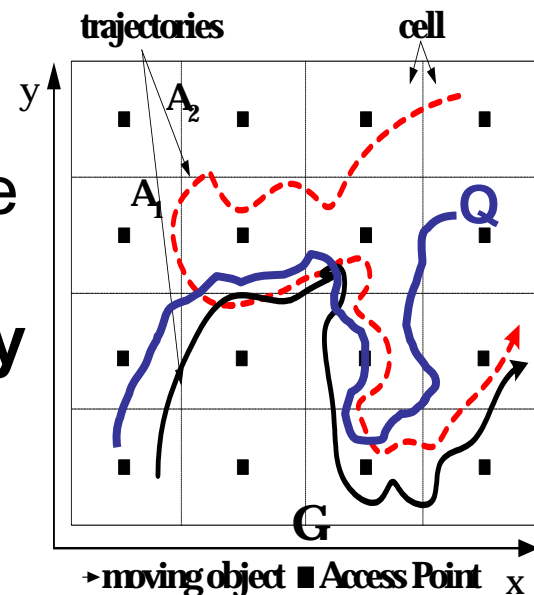
Future Work

- Deployment of TJA algorithm using the Riverside Sensor
- Provide the implementation of TJA as an extension of our Open Source P2P Information Retrieval Engine :

<http://www.cs.ucr.edu/~csyiazti/peerware.html>

- Distributed Top-K Queries using score bounds (which finds application in distributed Spatio-Temporal **Similarity & Pattern** Queries).

- Investigate whether these algorithms are useful in other domains, such as grids, vehicular networks, etc.



Distributed Top-K Query Processing

Thanks!

by

Demetrios Zeinalipour

<http://www.cs.ucy.ac.cy/~dzeina/>

***Wednesday, November 16, 2005
Campus Building 12, 2nd Floor, #202,
17:30-18:30***

