Tight Bounds for Distributed Selection

Fabian Kuhn, ETH Zurich
Thomas Locher, ETH Zurich
Roger Wattenhofer, ETH Zurich

Distributed Computing Group

19th ACM Symp. on Parallelism in Algorithms and Architectures
San Diego, CA, USA, June 2007

Motivation: Distributed Aggregation

Growing interest in distributed aggregation!

→ Sensor networks, distributed databases...

Aggregation functions?
→ *Distributive* (max, min, sum, count)
→ *Algebraic* (plus, minus, average)
→ *Holistic* (median, k^{th} smallest/largest value) ⟸ Distributed selection

Combinations of these functions enable *complex queries*!
→ „What is the average of the 10% largest values?"

What cannot be computed using these functions?

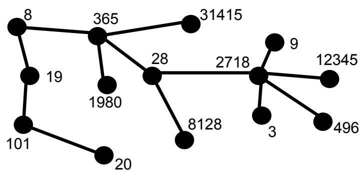
Motivation: Model

How difficult is it to compute these aggregation primitives?

Model:
- Connected graph G = (V,E) of diameter D_G, |V| = n.
- Nodes v_i and v_j can communicate directly if $(v_i,v_j) \in E$.
- A spanning tree is available (diameter $D \le 2 \cdot D_G$)
- Asynchronous model of communication.
- All nodes hold a single element.
- Messages can contain only a constant number of elements.

Simple breadth-first construction!

Can easily be generalized to an arbitrary number of elements!



Motivation: Distributive & Algebraic Functions

How difficult is it to compute these aggregation primitives?

→ We are interested in the time complexity!

Worst-case for every legal input and every execution scenario!

→ *Distributive* (sum, count...) and *algebraic* (plus, minus...) functions are easy to compute:
Use a simple *flooding-echo* procedure → ***convergecast***!

Slowest message arrives after 1 time unit!

Time complexity: $\Theta(D)$

What about holistic functions (such as k-selection)???
Is it (really) harder...?
Impossible to perform in-network aggregation?

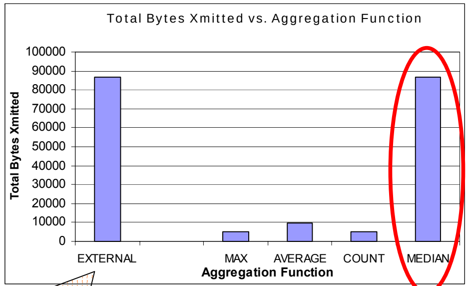
Motivation: Holistic Functions

It is widely believed that *holistic* functions are hard to compute using in-network aggregation.

Example: TAG is an aggregation service for ad-hoc sensor networks

→ It is fast for other aggregates, but not for the MEDIAN aggregate:

„Thus, we have shown that (...) in network aggregation can reduce communication costs by an order of magnitude over centralized approaches, and that, even in the worst case (such as with MEDIAN), it provides performance equal to the centralized approach.“



Taken from keynote by M. J. Franklin at PODC'03

2500 nodes in a 50x50 grid!

Motivation: Really so Difficult?

However, there is quite a lot of literature on distributed k-selection:

A straightforward idea: Use the sequential algorithm by Blum et al. also in a distributed setting → Time Complexity: $O(D \cdot n^{0.9114})$.

Not so great...

A simple idea: Use binary search to find the k^{th} smallest value → Time Complexity: $O(D \cdot \log x_{max})$, where x_{max} is the maximum value.

→ Assuming that $x_{max} \in O(n^{O(1)})$, we get $O(D \cdot \log n)$...

We do not want the complexity to depend on the values!

A better idea: Select values *randomly*, check how many values are smaller and repeat these two steps!

→ Time Complexity: $O(D \cdot \log n)$ in expectation!

Nice! Can we do better?

Outline

I. Motivation/Model

II. Algorithms

III. Lower Bound

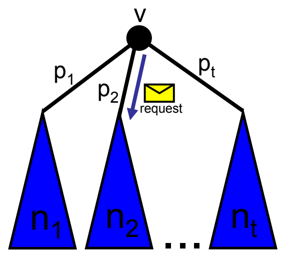
IV. Conclusion

Algorithms: Randomized Algorithm

Choosing elements uniformly at random is a good idea...

How is this done?

→ Assuming that all nodes know the sizes $n_1,\dots,n_t$ of the subtrees rooted at their children $v_1,\dots,v_t$, the request is forwarded to node v_i with probability:



$$p_i := n_i / (1+ \Sigma_k n_k).$$

With probability $1 / (1+ \Sigma_k n_k)$ node v chooses itself.

Key observation: Choosing an element randomly requires O(D) time!

Use pipe-lining to select *several random elements*!

D elements in O(D) time!

Algorithms: Randomized Algorithm

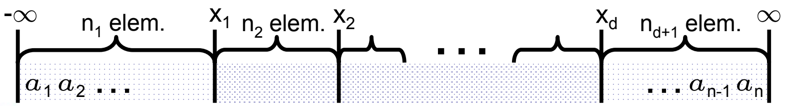
Our algorithm also operates in phases → The set of *candidates* decreases in each phase!

A *candidate* is a node whose element is possibly the solution.

A phase of the randomized algorithm:

1. Count the number of candidates in all subtrees
2. Pick O(D) elements $x_1, \ldots, x_d$ uniformly at random
3. For all those elements, count the number of smaller elements!

Each step can be performed in O(D) time!



Algorithms: Randomized Algorithm

Using these counts, the number of candidates can be reduced by a factor of D in a constant number of phases with high probability.

With probability at least $1-1/n^c$ for a constant $c \geq 1$.

We get the following result:

Theorem: The time complexity of the randomized algorithm is $O(D \cdot \log_D n)$ w.h.p.

More on that later...

We further proved a time lower bound of $\Omega(D \cdot \log_D n)$.

→This simple randomized algorithm is asymptotically optimal!

The only remaining question: What can we do deterministically???

Algorithms: Deterministic Algorithm

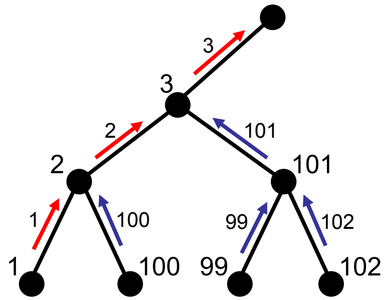
Why is it difficult to find a good deterministic algorithm???

→ Hard to find a good selection of elements that provably reduces the set of candidates!

Simple idea: Always propagate the median of all received values!

Problem: In one phase, only the h^{th} smallest element is found if h is the height of the tree...

→ Time complexity: $O(n / h)$



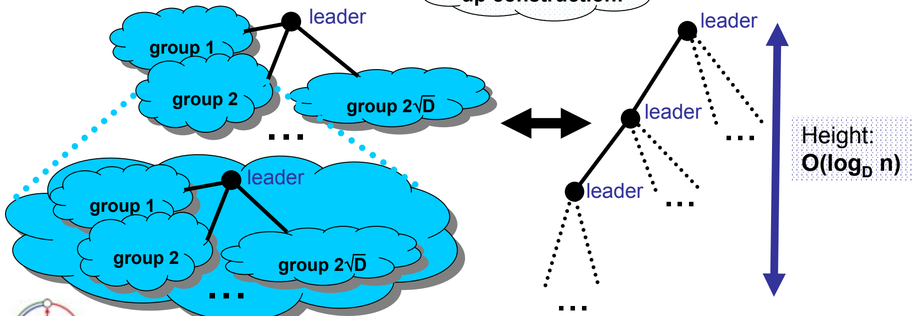
We can do a lot better!!!

Algorithms: Deterministic Algorithm

Idea: Split the graph into at most $2\sqrt{D}$ groups, each containing at most $\lceil n / \sqrt{D} \rceil$ candidates. Do this recursively!

Each group has a leader → „Virtual tree" consisting of leaders!

Simple bottom-up construction!

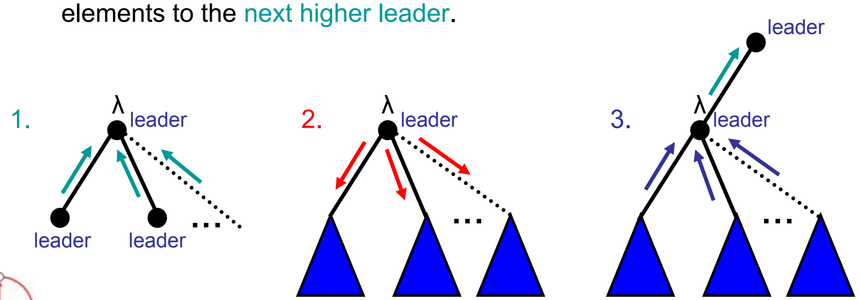


Algorithms: Deterministic Algorithm

A phase of the algorithm (at leader λ):

All steps require O(D) time!

1. Receive $\leq 2\sqrt{D}$ elements from each of $\leq 2\sqrt{D}$ leader children.
2. Count the number of smaller elements for all $\leq 4 \cdot D$ received elements (in all subtrees).
3. Use those counts to find $\leq 2\sqrt{D}$ elements (locally) that partition all elements into sets of size at most $\lceil n / \sqrt{D} \rceil$ and report those elements to the next higher leader.



Algorithms: Deterministic Algorithm

The number of candidates reduces by a factor of $O(\sqrt{D})$ in each phase, thus $O(\log_D n)$ phases are required.

Each phase costs $O(D \cdot \log_D n)$ time.

We get the following result:

> **Theorem**: The time complexity of the deterministic algorithm is $O(D \cdot \log_D^2 n)$.

Only a factor $O(\log_D n)$ worse than the randomized algorithm!

In a grid network ($D = \sqrt{n}$), the time complexity is $\Theta(D)$, asymptotically the same complexity as when computing „easy“ aggregates!!

Outline

I. Motivation

II. Algorithms

III. Lower Bound

IV. Conclusion

Lower Bound

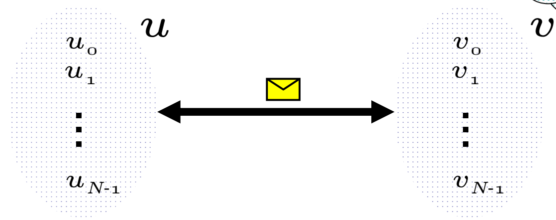
The proof of the lower bound of $\Omega(D \cdot \log_D n)$ consists of two parts:

Part I. Find a lower bound for the case of two nodes u and v with N elements each.

Let $u_0 < u_1 < \ldots < u_{N-1}$ and $v_0 < v_1 < \ldots < v_{N-1}$.

How are the 2N elements distributed on u and v?

What is the order between all u_i and v_j?



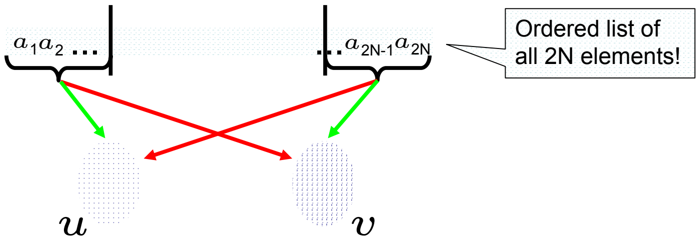
Lower Bound

Assume $N = 2^b$. We use b independent Bernoulli variables $X_0,\ldots,X_{b-1}$ to distribute the elements!

If $X_{b-1} = 0$ → N/2 smallest elements go to u and the N/2 largest elements go to v.

If $X_{b-1} = 1$ it is the other way round.

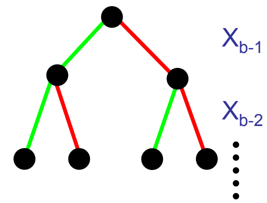
The remaining N elements are recursively distributed using the other variables $X_0,\ldots,X_{b-2}$!



Lower Bound

Crucial observation: For all 2^b possibilities for $X_0,\ldots,X_{b-1}$, the median is a different element.

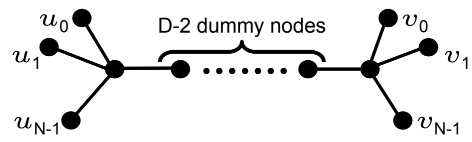
→ Determining all X_i is equivalent to finding the median!



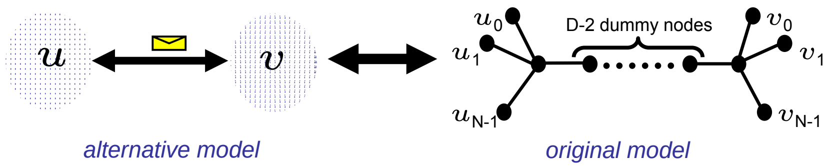
We showed that at least $\Omega(\log_{2B} n)$ rounds are required if B elements can be sent in a single round in this model!

Part II. Find a lower bound for the original model.

Look at the following graph G of diameter D:



Lower Bound



We showed that a time lower bound for the alternative model implies a time lower bound for the original model!

Theorem: $\Omega(D \cdot \log_D \min\{k,n-k\})$ rounds are needed to find the k^{th} smallest element.

$\Omega(D \cdot \log_D n)$ lower bound to find the median!

Outline

I. Motivation/Model

II. Algorithms

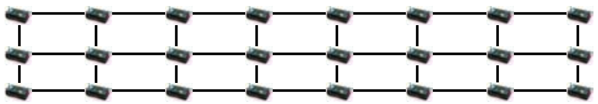
III. Lower Bound

IV. Conclusion

Conclusion

- Simple randomized algorithm with time complexity $O(D \cdot \log_D n)$ w.h.p.
  - Easy to understand, easy to implement...
  - Even asymptotically optimal! Our lower bound shows that no algorithm can be significantly faster!
- Deterministic algorithm with time complexity $O(D \cdot \log_D^2 n)$.
  - If $\exists c \leq 1$: $D = n^c$ → k-selection can be solved efficiently in $\Theta(D)$ time even deterministically!

Recall the 50x50 grid used to test out TAG!



Questions and Comments?

Thank you for your attention!

Thomas Locher
Distributed Computing Group
ETH Zurich, Switzerland
lochert@tik.ee.ethz.ch
http://dcg.ethz.ch/members/thomasl.html

Additional Slide: Deterministic Algorithm

A phase of the deterministic algorithm „step by step“:

1.a Count the number of candidates in all subtrees starting at the leaves.

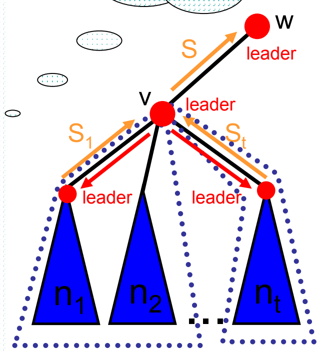
1.b Build groups at the same time → Link children together as long as each group contains at most $\lceil n / \sqrt{D} \rceil$ candidates. One node in each group becomes its leader.

2. The leaders split their group recursively into at most $t \leq 2\sqrt{D}$ groups.

3. Groups of size at most $2\sqrt{D}$ report all values S_i immediately.

4. Once all $\approx 2\sqrt{D} * 2\sqrt{D} = 4D$ values from all groups have arrived, count the elements in each interval and send a selection S of at most $\approx 2\sqrt{D}$ values to the next higher leader.

Each final interval contains at most $n / \sqrt{D}$ values!



All in O(D) time!