T-79.4001 Seminar on Theoretical Computer Science
Spring 2007 – Distributed Computation

Distributed selection

Toni Kylmälä
toni.kylmala@tkk.fi

Data

Data set: $\mathcal{D} = \bigcup_x D_x$

Distribution of set to sites D_{xi}: $\{D_{x1}, D_{x2}, \ldots, D_{xn}\}$

Basic operations

1. queries
2. updates

    2.1 insertion

    2.2 deletion

    2.3 change (but this can be seen as a deletion and an insertion).

Distribution of data set to sites x

Partitioning where two sites have no common elements: $D_i \cap D_j = \emptyset, i \neq j$. This is very good for updates but slow for queries.

Multiple-copy where every site has a copy of the entire data set. $\forall_i D_i = \mathcal{D}$. This is very good for queries but bad for updates.

Generally we have *partially replicated* data with problems from both extreme cases but no advantages from either.

Restrictions

IR (Connectivity, Total Reliability, Bidirectional Links, Distinc Identifiers)
For simplicity we assume the data to be sorted locally at each entity.
We also assume that in case of ties with data elements being in multiple sites we use ID:s to brake ties and achieve a totally ordered set. We also assume a spanning tree for communication and a single coordinating site *s*.

For efficiency the coordinator *s* should be the center of the graph and the tree a shortest path spanning tree for *s*.

Selection

The distributed selection problem is the general problem of locating $\mathcal{D}[K]$, the Kth smallest element. Problems of this type are called *order statistics*.

Median

If size of set $\mathcal{D}$ N is odd. There is only one median. $\mathcal{D}[\lceil N/2 \rceil]$. If N is even we have a *lover median* $\mathcal{D}[N/2]$ and an *upper median*. $\mathcal{D}[N/2+1]$.

Property 5.2.1

$\mathcal{D}[K] = \overrightarrow{\mathcal{D}}[N-K+1]$. *K*th smallest is the *(N - K + 1)*th largest element. This fact has important consequences.

Property 5.2.2

If a site has more than *K* elements then only the K smallest elements need be considered. Similarly for (N - K + 1) elements only the (N - K + 1) largest elements need be considered.

Selection in a small set *N = O(n)*

Input collection Collecting all the data to *s* and letting it solve locally is feasible but an overkill. M[Collect] = $O(n^2)$ in the worst case. (e.g. Ring)

Truncated ranking Making the messages depend on the value of K we can reduce the costs. E.g. by using the existing tree ranking protocol (exercise 2.9.4 *).

M[Rank] = n∆.

∆ = Min{K, N - K + 1}. If ∆ is small Rank is much more efficient but as it grows to N/2 the two protocols have the same cost.

Important

The two are generic protocols but it is possible to take advantage of the network topology. This is the case for Ring, Mesh and Complete Binary Tree.

Selection among two sites

When N » n we need a more efficient protocol. Here n = 2.

Median finding

A lower median has exactly $\lceil N/2 \rceil$ - 1 elements smaller than itself and $\lfloor N/2 \rfloor$ larger than itself. Thus comparing the local medians m_x and m_y we can eliminate halt of all the elements.

Assume that $|D_x| = |D_y| = N/2$, $N = 2^i$ and that m_x is larger. Then in D_x all the elements larger than m_x cannot be the median because they have N/4 in D_x and another N/4 in D_y for a total of N/2 elements smaller than themselves. Thus they can all be removed.

The same applies for the elements in D_y smaller or equal than m_y. They have N/4 + 1 elements in D_x and N/4 elements in D_y for a total of N/2 + 1 elements larger than themselves. So they cannot be medians and can be removed.

Consequence: The overall median is the median of the elements left.

Thus we simply reapply the process until only two elements are left and the global median can be determined.

Cost of protocol: Halving

Each iteration halves the data set thus having log N iterations. Only on message per iteration is required.

This can be generalised for arbitrarily sized sets without changing its complexity (Exercise 5.6.5).

Finding Kth smallest element

Assume again that $|D_x| = |D_y|$.

Case K < $\lceil N/2 \rceil$

There are more than K elements. Thus all elements larger than D$_i$[K] can be discarded leaving us with two sets of size K where finding the Kth smallest is finding the median.

Case K > $\lceil N/2 \rceil$

We can now look for the (N - K + 1)th (< $\lceil N/2 \rceil$) largest element thus similarly to the above case we have an upper median finding problem.

Summary

K-selection can be transformed into median finding.

General selection: RankSelect

With 10 to 100 sites and local data $\geq 10^6$ we need something else. Choose an item d_i out of $\mathcal{D}$ and count its rank d*. If d* < K then the item and all items smaller than it can not be the Kth item we want. Similarly for d* > K. This allows us to reduce the size of the search space at each iteration.

Counting the rank is a trivial broadcast in a SP and a convercast to collect the information.

Choosing d_i uniformly at random

It is possible (section 2.6.7 end exercise 2.9.52) to choose uniformly at random an item from the set $\mathcal{D}$ in a tree in the initial set. Also after items have been removed (exercise 2.9.52 and exercise 5.6.10) with the same costs.

Also by choosing from a set of locally uniformly chosen and weighted values at coordinator.

Costs of RandomSelect

Because in the worst case we only remove d_i for N iterations.
$M[\text{RandomSelect}] \leq (4(n - 1) + r(s))N$
$T[\text{RandomSelect}] \leq 5r(s)N$

However on average (Lemma 5.2.1) due to randomness:
$M_{Average}[\text{RandomSelect}] = O(N\log N)$
$T_{Average}[\text{RandomSelect}] = O(N\log N)$

Random choice with reduction

Because Kth smallest = (N - K + 1)th largest each site can reduce its search space to Δ_i = Min{$K_i, N_i - K_i$ + 1} before the random selection occurs.

M[RandomFlipSelect] = $\leq$ (2(n - 1) + r(s))N
T[RandomFlipSelect] = $\leq$ 3r(s)N

However on average (Lemma 5.2.2) due to randomness:
$M_{Average}$[RandomFlipSelect] = O(n(ln(Δ) + ln(N)))
$T_{Average}$[RandomFlipSelect] = O(n(ln(Δ) + ln(N)))

Selection in a Random Distribution - taking advantage of distribution knowledge

If all distributions are equally likely then we can get a representative of the entire set by choosing from the largest site D_z at iteration i the h_ith smallest element where
$h_i = \lceil K_i(\frac{m_i+1}{N+1}) - \frac{1}{2}\rceil$.
This will be used until there are less than n items under consideration and finish with Random-FlipSelect.

Due to randomness (Lemma 5.2.3)
$M_{Average}$[RandomRandomSelect] = $O(n(\log\log(\Delta) + \log(N)))$
$T_{Average}$[RandomRandomSelect] = $O(n(\log\log(\Delta) + \log(N)))$

Filtering

For systems where a guaranteed reasonable cost even in the worst case is required. This can be achieved e.g. with strategy RandomSelect with the appropriate choice of d_i.

Let D_x^i denote the elements of site x in iteration i and $n_x^i = |D_x^i|$ denote its size. Consider the (lower) median $d_x^i = D_x^i[\lceil n_x^i/2 \rceil]$ of D_x^i and let $M_i = \{d_x^i\}$ be the set of these medians. Associate a weight, the size of set x, to each median and choose d_i to be the weighted (lower) median of M_i.

Lemma 5.2.4 (and exercise 5.6.18): Iterations until n elements are left is at most 2.41 log(N/n). At each iteration determining the median of set M_i can be done using protocol Rank because we only have n elements. In the worst case it requires $O(n^2)$ messages in each iteration.

The worst case costs of this then are
M[Filter] = $O(n^2 log\frac{N}{n})$
T[Filter] = $O(nlog\frac{N}{n})$.

Reducing the worst case: ReduceSelect

Combining all the previous techniques and adding a few new ones allows us to reduce the costs further.

Reduction Tools

Reduction tool 1: Local Contraction. If a site has more than Δ items it can immediately reduce its item set to size Δ. Thus N is only nΔ after this tool has been used once. This requires that each site know N and K.

Reduction tool 2: Sites Reduction. If the number of sites n is greater than K (or N - K + 1), then n - N sites (or n - N + K - 1) and all data therein can be removed.

1. Consider the set $D_{min} = D_x[1]$ (or D_{max}).
2. Find the Kth smallest (or (N - K + 1)th largest) element w. For example using Rank.
3. If $D_x[1] > w$ (or respectively $< w$) then the entire set D_x can be removed.

This reduces the number of sites to at most Δ. (What about D_{min} = {1, 1 ,1 ,2 ,3, 3} when looking for the 3rd smallest?)

Combined use

Using the two tools together reduces the selection from N elements among n sites to selection from Min{n,Δ} sites each with at most Δ elements. Thus the search space is at most Δ^2 elements. It is also possible to successfully use them again. Call this protocol REDUCE.

Example

N: search space	K:rank of f* in search space	x_1	x_2	x_3	x_4	x_5
10,032	4096	10,000	20	5	5	2
4126	4096	4096	20	5	5	2
65	33	33	20	5	5	2

In the first round we only reduce the number of elements in site x_1. In the second round we find that “looking for the largest“ has a smaller value than the smallest (4126 - 4096 + 1 = 33). That allows us to again reduce the number of elements in site x_1. Finally our search space has only 65 elements left.

Lemma 5.2.5

After the execution of REDUCE, the number of elements left is at most Δ min{n, Δ}.

Costs

Each execution of local contraction requires a broadcast and a convergecast 2(n - 1) messages and 2r(s) time. Interestingly it will be executed a constant three times. (Exercise 5.6.19).

Cutting tools

For simplicity and with out loss of generality let K = Δ (the case where N - K + 1 = Δ is analogous.

Visualize the data as an n by Δ ($n \leq \Delta$) matrix. With data from site X_i in row i. Thus we have ordered rows and unordered columns. Now let us consider the set C(2) that is all second-smallest elements in each site. Find the kth = $\lceil K/2 \rceil$ smallest element m(2) of this set. It has k - 1 elements smaller than itself in C(2). All these elements including m(2) also have a total of k elements smaller than themselves in C(1). Thus it has a total of k + (k - 1) = 2k - 1$\geq$ K - 1. Thus any element larger than m(2) cannot be the Kth smallest element in the whole set. Thus we can remove them all from consideration.

Now consider the set $C(2^i)$ where $2^i \leq K$. The kth smallest element where k = $\lceil K/2^i \rceil$. By definition it has exactly k - 1 elements smaller than itself in $C(2^i)$. And 2^i - 1 elements smaller in its row. Thus it has at least (k - 1) + $k(2^i - 1) \geq \frac{K}{2^i}2^i - 1$ = K - 1 elements smaller than itself in the global set. Thus similarly as before any element larger than $m(2^i)$ cannot be the Kth smallest and can be removed.

Lemma 5.2.6

After the execution of Cutting Tool on all columns $\{C(2^i) \mid 2^i \leq K\}$. The number of elements left is at most: $\min\{n,\Delta\}\log\Delta$.

Costs

Each of the $\log\Delta$ steps requires a selection from a set of size at most $\min\{n,\Delta\}$. This can be done for example with Rank. Thus the overall worst case is

$M[CUT] = O(n^2\log\Delta)$

$T[CUT] = O(n\log\Delta)$

Putting it all together

Protocol REDUCE composed of reduction tools 1 and 2, reduces the search space from N to at most Δ^2. Protocol CUT consisting of cutting tools further reduces it to at most $\min\{n,\Delta\}$.

Starting from these we build a full selection protocol by continuing from $\min\{n,\Delta\}$ to O(n) (e.g. using protocol Filter) and then a protocol for small sets (e.g. Rank) to finally determine the sought element.

Thus the protocol ReduceSelect consists of executing REDUCE which requires 3 iterations of LocalContractions each using 2(n - 1) messages and 2r(s) time and one execution of SitesReduction that consists in execution of Rank. Protocol CUT is used with $N \leq \min\{n,\Delta\}\Delta$ and requires at most $\log\Delta$ iterations of the CuttingTools, each consisting in an executing of Rank. Protocol Filter is used with $N \leq \min\{n,\Delta\}\log\Delta$ and thus requires at most $\log\log\Delta$ iterations, each costing 2(n - 1) messages and 2r(s) time plus an execution of Rank. For a total cost:

$M[\text{ReduceSelect}] = (\log\Delta + 4.5\log\log\Delta + 2)M[\text{Rank}] + (6 + 4.5\log\log\Delta)(n - 1)$

$T[\text{ReduceSelect}] = (\log\Delta + 4.5\log\log\Delta + 2)T[\text{Rank}] + (6 + 4.5\log\log\Delta)2r(s)$

Summary

For small data sets $N = O(n)$ we have $O(n^2)$ protocols.

In the special case of $n=2$ we can efficiently choose arbitrary Kth smallest element with cost $O(\log K)$ regardles of N.

For the general case we have several ways to reduce the search space and arrive at a somewhat efficient solution on average.

As a special general case we also have a very complex protocol that guarantees a better worst case than the general ones but is slower on average.