Lecture 20: Distributed Computing and Streaming Algorithms

Lecturer: *Sanjeev Arora* Scribe: *Yuri Pritykin*

This lecture is devoted to two independent topics on algorithms.

1 Distributed Computing

Distributed computing deals with situations where there is no unique computing center, but there are many centers where computation happens, and these centers should compute something together. For instance, on aircraft there are many sensors of different kind that should agree on what is happening around, or there is distributed network that wants for instance to agree on time.

We consider the following example from this wide area.

1.1 Byzantine Generals Problem

The problem is as follows [2]: there are n processors, t of which may be faulty and all the rest are good. Each processor i keeps a bit $b_i \in \{0, 1\}$. Every processor can send its bit to every other processor, as well as can ask other processor about its bit. Good processors always answer honestly, while faulty processors may answer arbitrarily to different requests. The problem is to give a protocol such that:

- at the end all good processors agree on some bit;
- if all good processors in the beginning have the same bit b, then the bit they agree on should be b.

Some observations/facts:

1. If $t \geqslant n/3$, then there is no such a protocol (consider the simplest case of three processors with one faulty processor).
2. The simple majority protocol, that is, everybody asks bits from everyone and then takes the majority, does not work (consider the situation with one faulty processor and all others divided equally on having 0 and 1).
3. There exists a deterministic polynomial time algorithm that solves the problem for $t < n/3$ (this is difficult).

1.2 Rabin's Randomized Algorithm

We describe a simple randomized algorithm due to Rabin [3] which assumes that there is a global *random coin* that when tossed, is visible to all processors.

Let $n = 3t + 1$. The processors maintain a bit `vote` which is initially b_i for processor i. Let N be some natural number. The algorithm for each processor is the following.

Do N times:

1. Send `vote` value to every other processor.
2. Examine all n values of `vote` received (including your own). Identify `maj` = the majority bit among these values and `tally` = the number of times you saw `maj` bit among the `vote` values. (Notice that the values of `maj` and `tally` can vary widely among the processors, since faulty processors could try to confuse things by sending different values of `vote` to different processors.)
3. If `tally` $\geqslant 2t + 1$, then set `vote` = `maj`.
4. If `tally` $\leqslant 2t$, then look at the current global coin toss. If it is "Heads", then set `vote` $= 1$, else set `vote` $= 0$.

Proof of correctness. Note that if all the good processors have the same initial value, then they all set their votes to this value in the first round. In all other cases, we show that with probability at least $1/2$, all the processors assign the same value to `vote`. (Note that as soon as this happens, then always `tally` $\geqslant 2t + 1$ for all processors, and therefore all processors will continue executing step 3 in the algorithm.)

There are two cases.

1. Some processor has `tally` $\geqslant 2t + 1$, and `maj` $= b$ for some $b \in \{0, 1\}$. Since only t processors are faulty, we conclude that at least $t + 1$ good processors must have sent b as their value of `vote`. Thus no other processor will see both `tally` $\geqslant 2t + 1$ and `maj` $= 1 - b$ in the same round. Hence regardless of whether the other processors execute step 3 or 4 in the above algorithm, the probability is at least $1/2$ that they all set `vote` to b.

2. No good processor has `tally` $\geqslant 2t + 1$. Then all of them execute step 4, and with probability 1 set `vote` to the same value.

Thus after N steps with probability at least $1 - \frac{1}{2^N}$ all processors successfully agree on some bit. □

2 Streaming Algorithms and Algorithms for Large Data Set

The general situation here is that the data we are given is so large that we even do not have enough space to store it. Nevertheless we have an access to the data stream and want to compute something about the data on a fly. Examples are some astronomic telescope systems data, particle acceleration data. With Internet it is still not the case, since it is much smaller compared with the former examples of data.

2.1 Computing Frequency Moments

As an example, we consider the following problem from streaming algorithms. We are given a stream of data. Each data item has some type from the set $\{1, \dots, n\}$. Let m_i be the number of times the item of type i appeared so far in the stream. Let

$$F_k = \sum_{i=1}^{n} m_i^k$$

to be the frequency moments of a data stream. The goal is to compute F_k for $k = 1, 2, \ldots$

It is easy to compute F_1 keeping a counter and incrementing it after each new item appears. This requires $O(\log N)$ space where $N = F_1$ is the length of the data stream. How can we compute F_2? A trivial approach is to maintain a counter for each m_i, but that requires $O(n \log N)$ space which might be too large.

Now we describe a $(1+\delta)$-approximation algorithm for computing F_2 that requires much less space [1].

The idea is to use 4-wise independent random variables $\varepsilon_1, \ldots, \varepsilon_n \in \{-1, 1\}$ with $\mathbf{E}[\varepsilon_i] = 0$ for every i. Recall that random variables $\varepsilon_1, \ldots, \varepsilon_n$ are k-wise independent if every k of them are truly independent. If random variables are k-wise independent, then they are l-wise independent for every $l < k$. To construct k-wise independent variables, one is sufficient to use $O(k \log n)$ truly independent random bits (take values of a random k-degree polynomial modulo 2 and adjust to the range $\{-1, 1\}$).

The algorithm is as follows. In the beginning let $\texttt{counter} = 0$. Then whenever an item of type i appears in the stream, set $\texttt{counter} \leftarrow \texttt{counter} + \varepsilon_i$. Thus at the end we have $\texttt{counter} = \sum_{i=1}^{n} m_i \varepsilon_i$. Output $x = (\texttt{counter})^2$.

We have

$$\begin{aligned}
\mathbf{E}[x] = \mathbf{E}\left[\left(\sum_i m_i \varepsilon_i\right)\right] &= \mathbf{E}\left[\sum_{i,j} m_i m_j \varepsilon_i \varepsilon_j\right] \\
&= \sum_{i,j} m_i m_j \, \mathbf{E}[\varepsilon_i \varepsilon_j] = \sum_{i,j} m_i m_j \, \mathbf{E}[\varepsilon_i] \, \mathbf{E}[\varepsilon_j] = \sum_i m_i^2,
\end{aligned}$$

where the last equality follows from the fact that ε_i and ε_j are pairwise independent for $i \neq j$. That is, the average of x is what we wanted to get.

Furthermore,

$$\begin{aligned}
\mathbf{Var}[x] = \mathbf{E}[x^2] - (\mathbf{E}[x])^2 &= \mathbf{E}\left[\left(\sum_i m_i \varepsilon_i\right)^4\right] - \left(\sum_i m_i^2\right)^2 \\
&= \mathbf{E}\left[\sum_{i_1, i_2, i_3, i_4} m_{i_1} m_{i_2} m_{i_3} m_{i_4} \varepsilon_{i_1} \varepsilon_{i_2} \varepsilon_{i_3} \varepsilon_{i_4}\right] - \left(\sum_i m_i^4 - 2 \sum_{i \neq j} m_i^2 m_j^2\right).
\end{aligned}$$

However,

$$\begin{aligned}
\mathbf{E}\left[\sum_{i_1, i_2, i_3, i_4} m_{i_1} m_{i_2} m_{i_3} m_{i_4} \varepsilon_{i_1} \varepsilon_{i_2} \varepsilon_{i_3} \varepsilon_{i_4}\right] &= \mathbf{E}\left[\sum_i m_i^4 \varepsilon_i^4\right] + 6 \, \mathbf{E}\left[\sum_{i \neq j} m_i^2 m_j^2 \varepsilon_i^2 \varepsilon_j^2\right] \\
&= \sum_i m_i^4 + 6 \sum_{i \neq j} m_i^2 m_j^2,
\end{aligned}$$

where we used the fact that $\mathbf{E}[\varepsilon_{i_1}\varepsilon_{i_2}\varepsilon_{i_3}\varepsilon_{i_4}] = 0$ if any of indices i_t appears here odd number of times.

Hence, we have

$$\begin{aligned}\mathbf{Var}[x] = \sum_i m_i^4 + 6\sum_{i\neq j} m_i^2 m_j^2 - \left(\sum_i m_i^4 - 2\sum_{i\neq j} m_i^2 m_j^2\right) \\ = 4\sum_{i\neq j} m_i^2 m_j^2 \leqslant 2\left(\sum_i m_i^4 + 2\sum_{i\neq j} m_i^2 m_j^2\right) = 2(\mathbf{E}[x])^2 = 2F_2^2.\end{aligned}$$

The variance is quite large, but we can reduce it by repeated sampling. Take T independent copies $x_1, \ldots, x_T$ of x, that is, obtained from T different independent blocks $(\varepsilon_i)_{i=1}^n$ of random variables. Let $Y = \sum_{i=1}^T x_i$. Then $\mathbf{E}[Y] = T\,\mathbf{E}[x] = TF_2$ and $\mathbf{Var}[Y] = 2TF_2^2$, since x_i are independent.

By Chebyshev's inequality,

$$Y/T \approx O\left(F_2 \pm \frac{\sqrt{2T}F_2}{T}\right),$$

therefore to get a $(1\pm\delta)$-approximation, we are sufficient to take $T \geqslant 2/\delta^2$. This procedure requires $O(\frac{1}{\delta}\log n)$ space.

References

[1] N. Alon, Y. Matias, and M. Szegedy. The space complexity of approximating the frequency moments. In *Proc. 26th STOC*, pages 20–29, 1996.

[2] L. Lamport, R. Shostak, and M. Pease. The byzantine generals problem. *ACM Trans. Program. Lang. Syst.*, 4(3):382–401, 1982.

[3] M. O. Rabin. Randomized byzantine generals. In *Proc. 24th FOCS*, pages 403–409, 1983.