Sharing random bits with no process coordination

Marius Zimand
School of Computer & Applied Sciences
Georgia Southwestern State University
Americus, GA 31709, USA

Abstract

We present a method by which any polynomial-time randomized distributed algorithm is transformed in such way that each participating process needs only *polylog* local random bits and access to a server providing random strings. The method assumes no coordination among the processes. The error probability increases by only an additive negligible term, and the time complexity of each process increases by at most a *polylog* factor. The main contribution of the paper is in reducing the length of the local random string from (roughly) quadratic (as reported in [Zim97]) to (roughly) linear in the logarithm of the length of the input.
Keywords: randomized distributed algorithm, random bits, pseudo-random generator.

1 Introduction

It is common wisdom that randomness is a useful computational resource in distributed computing (as well as in other contexts). It is less clear how to obtain random bits. A good solution is to use a natural source, such as a Geiger counter or a Zener diode, which produces an output that is indeed unpredictable. The disadvantage of this approach is that it is not realistic to assume that every processor that is ever to run a probabilistic routine is hooked to, say, a Geiger counter. On the other hand it is reasonable to set up a public server that provides good random strings to anyone in need. In this paper we investigate how to use such a server for implementing randomized distributed algorithms.

We allow maximum generality in that we assume that all the processes that participate in a randomized distributed algorithm should possess their own *independent* random bits. An obvious solution to sharing a random string provided by a public server (such a string is called a *global random string*) is to make each of the participating processes grasp a distinct block from the global random string. The downside is that this approach needs a lot of coordination in order to organize the processes to access disjoint blocks of random bits. We propose a solution that does not require any coordination among the participating processes. The price to be paid is that each process has some *local* independent random bits obtained in an unspecified way (say, off-line). However the number of such bits is only *polylog* in the input length. Moreover, under the assumption that the time needed to access one bit from the global random string counts as one unit, the time complexity of each participating process increases by at most a polylog factor, and the error probability increases by only a negligible additive term.

Our solution uses as a main ingredient a randomized pseudo-random generator. More precisely, we build a function that has access to the global random string and that, on an input of size $\log^{2+\epsilon} m$, outputs a string of length m that, with high probability of the global random string, cannot be distinguished from a real random string by any circuit of size polynomial in m even if the circuit has access to the global random string as well. The construction of the randomized pseudo-random generator uses as a first step the randomized one-way function built by Impagliazzo [Imp96]. In a landmark paper, Håstad et al. [HILL91] have presented a general technique to construct a pseudo-random generator given any one-way function. In this paper, we take advantage of some features of Impagliazzo's randomized one-way function and of the setting of our problem to design a much more efficient construction than the general one mentioned above.

2 Formal model

A randomized distributed algorithm on an input x consists of a protocol P among some processes, $Proc_1, Proc_2, \ldots, Proc_k$, where each process $Proc_i$ has access to the common input x, to a local private string r_i, and to a global random string R. The result of the randomized distributed algorithm as well as the randomized distributed algorithm itself are denoted by $P(Proc_1(x; y_1, R), \ldots, Proc_k(x; y_k, R))$. We impose only a feasibility restriction, namely that the number k of participating processes is polynomial in $|x|$ (the length of the input) and that for each input length there exists an oracle circuit C, with inputs $x, y_1, \ldots, y_k$, whose sizes are polynomial in $|x|$, such that when working with oracle R, $C^R(x, y_1, \ldots, y_k) = P(Proc_1(x; y_1, R), \ldots, Proc_k(x; y_k, R))$. We say that a randomized distributed algorithm computes a given function $f(x)$ with error probability $(\delta(n), \beta(n))$, if for all x of length n, with probability of $R \geq 1 - \delta(n)$,

$Prob_{y_1,\ldots,y_k}(P(Proc_1(x; y_1, R), \ldots, Proc_k(x; y_k, R)) = f(x)) \geq 1 - \beta(n)$.

The notation is standard. By Σ we denote the binary alphabet and Σ^n is the set of binary strings of length n. U designates (generically) the uniform distribution (the sample space should be clear from the context). The notation $x \in_D S$, where D is a distribution on the sample space S means that x is chosen from S according to the distribution D. All probabilities, denoted by $Prob(\cdot)$, are with respect to the uniform distribution unless specified otherwise.

3 The simple approaches are not efficient

One simple approach is to let the server give distinct blocks of random strings to each client. This implies that the server has a mutual exclusive routine that calculates for each client an address from where the client can download the block of bits it has requested. It has been reported that such a blocking approach has a significant cost for many data structures such as queues, stacks, and heaps (see [MS97] and the papers cited there). More to the point, we ran some experiments to evaluate the relative performance of synchronized and nonsynchronized access to a buffer (supposedly containing the global random string). In our experiments, a number of k threads had to copy 800000 bytes in batches that were b bytes long. In the synchronized version, the addresses from where the threads are taking the current batch are computed by a routine that has a mutual exclusive lock. In the nonsynchronized version, the above addresses are simply incremented by b units before each copy of the current batch. The following table shows the performance results. We have computed the average (over all the threads) time in milliseconds needed by a thread to complete the copying.

	Batch size	No. of threads	Avg. time
Nonsync	50000	40	6197
Sync			8359
Nonsync	50000	80	6223
Sync			12109
Nonsync	50000	120	8297
Sync			13178
Nonsync	25000	40	7846
Sync			39532
Nonsync	25000	80	7470
Sync			84325
Nonsync	25000	120	7600
Sync			101852

Table 1: Relative performance for nonsynchronized and synchronized access to a buffer.

As it can be seen, when the contention to access the buffer is high, the synchronized version is significantly slower than the nonsynchronized version.

Another simple approach is possible in the case in which the processes $Proc_1, \ldots, Proc_k$ do not use the global random string R. In the notation we drop R and the distributed randomized algorithm is denoted by $P(Proc_1(x, y_1), \ldots, Proc_k(x, y_k))$. In this case, simply taking at random a block from the global random string does the job. "At random" means that the address of the block is given by the local random string. More specifically, suppose $|y_1| = |y_2| = \ldots = |y_k| = m$. Then, by using the Chernoff bounds, the folowing holds (for a proof see [Zim97]).

Proposition 3.1 *The above randomized distributed algorithm can be simulated using a global random string R such that the length of the local random strings is $O(\log^2 m)$. With high probability of R the error probability does not increase by more than $2^{-\Omega(\log^2 m)}$.*

The above proof dose not work for the general case in which the processes use a global random string R. However, we will show that a different approach yields a solution having even better parameters (shorter local random string).

4 Technical tools

We first need some definitions to capture the idea that two distributions are close in the absolute sense or from an algorithmical point of view. Let X and Y be two distributions on the same sample space S. X and Y are statistically ϵ-close (or just ϵ-close) if $\Delta(X, Y) = \sum_{a \in S} |Prob(X = a) - Prob(Y = a) < \epsilon$. $\Delta(X, Y)$ is called the distance between X and Y. It is easy to see that $\Delta(X, Y) = 2 \max_{A \subseteq S} |Prob_X(A) - Prob_Y(A)|$. If we look only at those subsets of S that are computed by circuits of some given size s, we get the notion of algorithmically closeness between two distributions. Let X and Y be two distributions on the same sample space S. X and Y are computationally ϵ-close for circuits of size s, if for any circuit C of size s, $\Delta_{comput}(X, Y) = |Prob_{x \in_X S}(C(x) = 1) - Prob_{y \in_Y S}(C(y) = 1)| \leq \epsilon$. This concept is extended in the natural way for the case in which X and Y are ensembles of distributions and C is a family of circuits.

Let D be a distribution on Σ^n. The *maximum mass* of D is defined to be $max - mass(D) = \max_{x \in \Sigma^n} D(x)$.

At some point in our construction we will use extractors. An extractor is a bipartite graph that can be used to reduce the maximum mass of distributions. More precisely, an (n, k, d, m, ϵ)-extractor is a regular bipartite graph having 2^n nodes in the left-hand side and 2^m nodes in the right-hand side, with the degree of each node in the left-hand side equal to d, and such that if node x is randomly chosen in the left-hand side according to a distribution that has maximum mass k and y is uniformly at random chosen among the edges going out from x, the distribution of $E(x, y)$ is ϵ-close to the uniform distribution on Σ^m. ($E(x, y)$ denotes the node on the right-hand side that is reached from x following y.)

Ideally, we would like to be able to effectively and efficiently build extractors. In fact, quite good extractors have been constructed in recent years [TS96], [Zuc96] (see also the survey paper [Nis96]). However, in the best such extractors, the length of y is polylog in x and $1/\epsilon$ and this is too large for our purposes. In our setting, y will be part of the local random string, and thus, we would like to use an as short y as possible.

On the other hand, since we allow the use of a global random string, we do not necessarily need a deterministic construction of an extractor. Indeed, it turns out that a random regular $(2^n, 2^m)$ bipartite graph with degree $d = O(n2^m/\epsilon^2)$ is with high probability an (n, m, d, m, ϵ)-extractor. This follows from the following two lemmas whose proofs will be given in the full version of this paper.

Lemma 4.1 *Let $G = (V_1, V_2, E)$ be a random regular bipartite graph with the left-hand side $V_1 = \Sigma^n$, the right-hand side $V_2 = \Sigma^m$ and degree $D = 2^d = 9n2^m 1/\epsilon^2$. Then with probability of G at least $1 - 2^{-(n+1)}$, for all $a, a' \in V_1$ with $a \neq a'$, $Prob_{y\in\Sigma^d}(E(a, y) = E(a', y)) \leq 2^{-m}(1+\epsilon)$.*

Lemma 4.2 *Let $G = (V_1, V_2, E)$ be a random regular bipartite graph with the left-hand side $V_1 = \Sigma^n$, the right-hand side $V_2 = \Sigma^m$ and degree $D = 2^d = 9n2^m 1/\epsilon^2$. Then, with probability of G at least $1 - 2^{-(n+1)}$, the distribution of $(E(x, y), y)$, when x is chosen in V_1 according to a distribution with maximum mass bounded by $p/2^n$ and y is uniformly at random chosen in Σ^d, is $\sqrt{\epsilon + \frac{p}{2^{n-m}}}$-close to the uniform distribution.*

5 Main result

The main result is as follows.

Theorem 5.1 *Let $P(Proc_1(x; y_1, R), \ldots, Proc_k(x; y_k, R))$ be a randomized distributed algorithm computing a function $f(x)$ with error probability $(\delta(n), \beta(n))$. Let poly be a polynomial and $\epsilon > 0$. Then there exists processes $Proc'_1, \ldots, Proc'_k$ such that:*

(1) With probability of R greater than $1-(2^{-(\log|x|)^{1+\epsilon}} + \delta(|x|))$,

$$Prob_{r_1,\ldots,r_k}(P(Proc'_1(x; r_1, R), \ldots, Proc'_k(x; r_k, R)) = f(x)) \geq 1 - (\beta(|x|) + 2^{-\Omega((\log|x|)^{1+\epsilon})}),$$

(2) For each i, $|r_i| = \Theta((\log|y_i|)^{1+\epsilon})$,

(3) For each i, the time spent by $Proc'_i(x; r_i, R)$ is at most the time spent by $Proc_i(x; y_i, R)$ times a factor that is polylog in $|x|$.

It should be remarked that the protocol P governing the processes interaction remains the same and, thus, our solution assumes no additional coordination among the processes $Proc'_i$. The theorem states that each process $Proc_i(x; y_i, R)$, with y_i a local random string, can be transformed into an equivalent process $Proc'_i(x; r_i, R)$, using a local random string r_i and having access to a global random string R, supposedly provided by a server specialized in producing good random strings. Since $Proc_i$ performs a polynomial-time computation, the number of bits of y_i can be polynomial in $|x|$, while the number of bits of r_i is only polylog in $|x|$. With high probability on the global random string R, the error probability increases only slightly, as well as the time complexity of each process. However, under the realistic assumption that obtaining good local random bits is difficult and expensive, the saving in the number of local random bits more than compensates for these minor drawbacks.

Sketch of proof. We provide first an overview of the proof. Each of the processes $Proc'_i$ simulates $Proc_i$ with the only modification that accesses to the local random string y_i of the latter are replaced by accesses to the bits of the string $Y_i(r_i, R)$, where r_i is local and R is global. The bulk of the proof consists in the construction of some random variables $Y_1(r_1, R), \ldots, Y_t(r_t, R)$, where t is an arbitrary integer, that depends on a parameter m and an arbitrary polynomial $poly$, and satisfies the following properties:

(1) each local sample space $\mathcal{R}_i$ is the set of $6(\log m)^{1+\epsilon}$ long binary strings,

(2) for each i, $Y_i(r_i, R)$ is an m long binary string,

(3) $Y_i(r_i, R)$ is computable in quasilinear time in m,

(4) with probability of R at least $1 - 2^{-(\log m)^{1+\epsilon}}$, for any choice of a polynomial number of random variables $Y_{i_1}, \ldots, Y_{i_{p(m)}}$ from $\{Y_1, \ldots, Y_t\}$, the joint distribution $< Y_{i_1}, \ldots, Y_{i_{p(m)}} >$ (with R fixed and the local r_i's uniformly at random chosen from the local sample spaces $\mathcal{R}_i$) is at most $2^{-\Omega((\log m)^{1+\epsilon})}$ computationally distinguishable from $< U_m, \ldots, U_m >$ (i.e., m independent copies of the uniform distribution over strings of length m) by circuits of size polynomial in m.

It is then easy to check that the processes $Proc'_i$ simulating as above the processes $Proc_i$ verify the conditions in the Theorem 5.1.

We now move to the construction of the random variables $Y_1(r_1, R), \ldots, Y_t(r_t, R)$. It is a modification of the construction of the randomized pseudo-random generator from [Zim96] based on the probability one one-way function of Impagliazzo [Imp96].

In the initial randomized distributed algorithm $P(Proc_1(x; y_1), \ldots, Proc_k(x; y_k))$, we can assume without loss of generality that $|y_1| = |y_2| = \ldots = |y_k| = m$, where m is a polynomial in $|x|$. It is easier to describe the construction below in terms of a unique parameter n, where $n = (\log m)^{1+\epsilon}$. The construction can be viewed as consisting of five main steps.

Step 1 (The goal of this step is the construction of a randomized one-way function) Let $\mathcal{R}$ be the set of all functions $R : \Sigma^n \to \Sigma^n$. For each $R \in \mathcal{R}$, let $f_R(x) = R(x)$. Impagliazzo [Imp96] has shown that for most R f_R behaves like a one-way function. More precisely, let $q(n)$ be an arbitrary polynomial and $size$ be a bound on the size of adversary circuits with $size \leq 2^{an}$, where $a < 1/2$. Then, with probability of R in $\mathcal{R}$ at least $1 - 2^{-q(n)}$

(a) all strings in Σ^n have at most $p = 2eq(n) + n$ preimages under f_R, and

(b) for $t = 2size \log 2size + 2q(n)$, there are at least $2^n - 2e^2 p \cdot size \cdot t$ strings x in Σ^n that map via f_R into strings that are nonvinvertible by any circuit C of size $size$, i.e., $f_R(C^R(f_R(x))) \neq f_R(x)$.

Let $\mathcal{R}_0$ be the subset of $\mathcal{R}$ consisting of the elements R that satisfy (a) and (b).

Step 2 (The randomized one-way function is expanded with some hidden bits.) Let $g_R(x, s) = (f_R(x), s)$, where $|x| = n$ and $|s| = 2n$. Let $b_i(x, s)$ be the inner product modulo 2 of x and $(s_i, \ldots, s_{i+n-1})$, where s_i is the i-th bit of s. Let $l = 1 + bn$, where b is a positive constant that will be specified later, and define $b(x, s) = (b_1(x, s), b_2(x, s), \ldots, b_l(x, s))$. Then by using the techniques of Goldreich and Levin [GL89], it can be shown that the function $b(x, s)$ provides hidden bits for $g_R(x, s)$. More precisely, there are positive constants b and c_1 such that for all $R \in \mathcal{R}_0$, $(g_R(X), b(X))$ and $(g_R(X), Y)$ are $2^{-c_1 n}$ computationally close for circuits of size $size$ working with oracle R, where X is the uniform distribution on Σ^{3n} and Y is the uniform distribution on Σ^l.

Step 3 (We apply a random extractor to $\overline{g}_R$ to obtain a randomized pseudo-random generator that expands its input by one bit.) From (a) in Step 1, it follows that the maximum mass of $g_R(x)$ is $p/2^{3n}$. We consider a random regular bipartite graph H as in Lemma 4.2 with $V_1 = \Sigma^{3n}$, $V_2 = \Sigma^{3n-bn}$, and $\epsilon = 2^{-n}$. It follows that with probability of G at least $1 - 2^{-(3n+1)}$, $(E(g_R(x), y), y)$ is $\sqrt{2} \cdot 2^{-0.4bn}$-close to the uniform distribution. The graph H is part of the global random string. (In fact, the server can first check that H is a "good" graph that has the above property.) Let $\mathcal{H}_0$ be the set of such graphs.

Now take $G_{R,H}(x, y) = (E(g_R(x), y), y, b_R(x))$. We have that

$$\begin{aligned}&\Delta_{comput}(G_{R,H}(x, y), u_{3n+1})\\ &\leq \Delta_{comput}((g_R(x), b_R(x)), (g_R(x), u_l) +\\ &\Delta_{comput}((E(g_R(x), y), y), u_{3n-bn+|y|}),\end{aligned}$$

where u_j, for various values of j, denotes an element of length j chosen according to the uniform distribution.

The first term is bounded by $2^{-c_2 n}$, and the second term is bounded by $\sqrt{2} \cdot 2^{-0.4bn}$. Thus, there is some constant c_3 such that for all $R \in \mathcal{R}_0$ and for all $H \in \mathcal{H}_0$, $G_{R,H}(x, y)$ is $2^{-c_3 n}$ computationally close to the uniform distribution. Observe that $|x| + |y| \leq 6n$ and that $G_{R,H}$ outputs a string that is one bit longer than its input. For simplicity, we assume that $|x| + |y| = 6n$.

Step 4 (Double the extension of the randomized pseudo-random generator.) We define $I_{R,H} : \Sigma^{6n} \to \Sigma^{2 \cdot 6n}$ by $I_{R,H}(x) = (s_1, s_2, \ldots, s_{2|x|})$, where $s_1, \ldots, s_{2|x|}$ are bits defined inductively as follows: $x_0 = x$ and for $i = 1, \ldots, 2|x|$, $s_i = \text{ the first bit of } G_{R,H}(x_{i-1})$ and $x_i = \text{ the last } |x| \text{ bits of } G_{R,H}(x_{i-1})$. Again, by an application of the hybrid method, there exists a positive constant c_4 such that, for all $R \in \mathcal{R}_0$ and for all $H \in \mathcal{H}_0$, $I_{R,H}(X)$ is $2^{-c_4 n}$ close to the uniform distribution for circuits of size $size$, where X is the uniform distribution on Σ^{6n}.

Step 5 (Get a randomized pseudo-random generator with more expansion) Define $F_{R,H} : \Sigma^{6n} \to \Sigma^{2^{n^\delta}}$ as follows, where $\delta = 1/(1+\epsilon)$. Let $I_0(x)$ and $I_1(x)$ be the first and respectively the second half of the string $I_{R,H}(x)$. Let $j = n^\delta$ and $\alpha_1\alpha_2\ldots\alpha_j$ be a j-bit string. The $\alpha_1\alpha_2\ldots\alpha_j$ bit of $F_{R,H}(x)$ is the first bit of $I_{\alpha_1}(I_{\alpha_2}(\ldots(I_{\alpha_j}(x))\ldots)$. The techniques of Goldreich, Goldwasser, and Micali [GGM86] show that there is a positive constant c_6 such that, for all $R \in \mathcal{R}_0$ and for all $H \in \mathcal{H}_0$, $F_{R,H}(X)$ is $2^{-c_6 n}$ computationally close to the uniform distribution for circuits of size $size$ working with oracle (R, H), where X is the uniform distribution on Σ^{6n}.

Observe that for all $R \in \mathcal{R}_0$ and for all $H \in \mathcal{H}_0$, $F_{R,H}$ takes an input of size $6n$ and produces an output of size 2^{n^δ} that cannot be distinguished by any circuit of size bounded by 2^{an} working with oracle (R, H), with $a < 1/2$, and in particular by any oracle circuit of size polynomial in 2^{n^δ}. (End of Step 5, and of construction.)

Recall that the initial randomized distributed algorithm is $P(Proc_1(x; y_1), \ldots, P(Proc_k(x; y_k))$. For $i = 1, \ldots, k$, let $\mathcal{R}_i = \Sigma^{6n} (= \Sigma^{6 \log^2 m})$ be the local sample spaces and $\mathcal{R} = \{f : \Sigma^n \to \Sigma^n\} \times \{G \mid G \text{ is a bipartite graph as in Step 3}\}$ be the global sample space.

We take, for $i = 1, \ldots, k$, $Y_i(r_i, R) = F_R(r_i)$, where r_i is taken uniformly at random from $\mathcal{R}_i$ and R is taken uniformly
at random from $\mathcal{R}$. Finally, we take $Proc'_i(x; r_i, R) = Proc_i(x; Y_i(r_i, R), R)$ and the new randomized distributed algorithm is $P(Proc'_1(x; r_1, R), \ldots, Proc'_k(x; r_k, R))$. Let us check that the new randomized distributed algorithm satisfies the purported conditions.

Observe first that each bit of $Y_i(r_i, R)$ can be computed in time that is polynomial in n. Indeed, by looking at Step 5, each bit of $Y_i(r_i, R)$ is computed by invoking $j = n^\delta$ times the functions I_0 or I_1 on inputs of size $6n$ and it can be seen that each such computation can be done in time that is polynomial in n. Since $n = (\log m)^{1+\epsilon}$, $Y_i(r_i, R)$ is computed in time $m \log^\gamma m$, for some constant γ and, thus, point (3) from the statement of Theorem 5.1 is verified.

Since $|r_i| = 6 \cdot n$, point (2) is true as well.

It remains to check point (1). The complete proof is provided in the complete version of this paper. The idea is that the error probability of $P(Proc'_1(x; r_1, R), \ldots, Proc'_k(x; r_k, R))$ cannot increase by more than $2^{-(\log m)^{1+\epsilon}}$ because, otherwise, a polynomial-size circuit would be able to simulate the new randomized distributed algorithm and use the simulation to distinguish between the distribution of $Y(r, R)$ and the uniform distribution and this would contradict the conclusion of Step 5. ∎

References

[GGM86] O. Goldreich, S. Goldwasser, and S. Micali. How to construct a random functions. *Journal of the ACM*, 33(4):792–807, 1986.

[GL89] O. Goldreich and L. Levin. A hard-core predicate for all one-way functions. In *Proceedings of the 21st ACM Symposium on Theory of Computing*, pages 25–32, 1989.

[HILL91] J. Hastad, R. Impagliazzo, L. Levin, and M. Luby. Construction of a pseudo-random generator from any one-way function. Technical Report 91-68, ICSI, Berkeley, 1991.

[Imp96] R. Impagliazzo. Very strong one-way functions and pseudo-random generators exist relative to a random oracle. (manuscript), January 1996.

[MS97] M. Michael and M. Scott. Relative performance of preemption-safe locking and non-blocking synchronization on multiprogrammed shared memory multiprocessors. In *Proceedings of the 11th International Parallel Processing Symposium, Geneva, Switzerland*, April 1997.

[Nis96] N. Nisan. Extracting randomness: how and why. A survey. In *Proceedings of the 11th Computational Complexity Conference*, pages 44–58, 1996.

[TS96] A. Ta-Shma. On extracting randomness from weak random sources. In *Proceedings of the 26th ACM Symposium on Theory of Computing*, pages 276–285, 1996.

[Zim96] M. Zimand. How to privatize random bits. Technical Report 616, Department of Computer Science, Univ. of Rochester, Rochester, NY, April 1996.

[Zim97] M. Zimand. How to share random bits. In *Proceedings 10th International Conference on Parallel and Distributed Computing Systems, New Orleans*, October 1997.

[Zuc96] D. Zuckerman. Randomness-optimal sampling, extractors, and constructive leader election. In *Proceedings of the 26th ACM Symposium on Theory of Computing*, pages 286–295, 1996.