Distributed Function Computation in Asymmetric Communication Scenarios

Samar Agnihotri†, Rajesh Venkatachalapathy§, H. S. Jamadagni†
CEDT, Indian Institute of Science, Bangalore - 560012, India
§Systems Science Graduate Program, Portland State University, Portland, OR 97207
Email: samar@cedt.iisc.ernet.in, venkatr@pdx.edu, hsjam@cedt.iisc.ernet.in

Abstract—We consider the distributed function computation problem where the sink computes some function of the data split among N correlated informants, in asymmetric communication scenarios. The distributed function computation problem is addressed as a generalization of distributed source coding (DSC) problem. We are interested in computing the minimum number of informant bits required, in the worst-case, to allow the sink to exactly compute the function. We provide a constructive solution for this in terms of an interactive communication protocol and prove its optimality. The proposed protocol also allows us to compute the worst-case achievable rate-region for the computation of any function. We introduce two equivalence classes of functions: lossy and lossless and show that, in general, the lossy functions can be computed with fewer informant bits than the DSC problem, while computation of the lossless functions requires as many informant bits as the DSC problem.

I. INTRODUCTION

Let us consider a distributed function computation scenario, where a sink node is interested in exactly computing some function \( f = f(X) \) of data-vector \( X \) that is split among \( N \) correlated informants. The correlation in informants’ data is modeled by discrete and finite distribution \( \mathcal{P} \), known only to the sink (asymmetric communication, [11]). The sink and informants interactively communicate with each other, with communication proceeding in rounds, as in [2]. We are concerned with minimizing the number of bits that the informants send, in the worst-case, to allow the sink to compute \( f \).

We consider the distributed function computation problem as a generalization of distributed source coding (DSC) problem\(^1\). The particular distributed function computation problem we consider is a generalization of DSC problem in asymmetric communication scenarios we addressed in [3]. The motivation for this work comes from sensor networks, particularly from our efforts to address the distributed function computation problem in single-hop data-gathering wireless sensor networks, while maximizing the worst-case operational lifetime of the network. In a typical data-gathering sensor network, it is reasonable to assume that the base-station has large resources of energy, computation, and communication as well as the knowledge of correlations in sensor data, whereas a sensor node is resource-limited and only knows its sampled data-values. Therefore, we argue that in such communication scenarios, the onus should be on the base-station to bear most of the burden of computation and communication in the network. Allowing interactive communication between the base-station and sensor nodes lets us precisely do this: base-station forms and communicates smart queries to sensor nodes, which they respond to with short and easily computable messages. This reduces the communication and computation effort at sensor nodes, hence enhancing their lifetime, which in turn leads to increased network lifetime.

The distributed function computation problem was first addressed by Yao in [2] and later by other researchers in different setups, as we discuss in Section II. However, our work mainly differs from the extant work in one or more aspects as follows. First, we approach the distributed function computation problem as a generalization of DSC problem. This allows us to exploit the correlation in informants’ data to solve the function computation problem at the sink. Second, we are concerned with asymmetric communication (only sink knows the joint distribution of informants’ data) and asymmetric computation (only sink computes the function). Third, we are concerned with the worst-case analysis. Fourth, we are interested in distributed function computation with a single instance of data at informants (one-shot computation problem). Finally, we consider a more powerful model of communication where the sink and informants interactively communicate with each other. This work clearly delineates the roles played in optimally solving the distributed function computation problem over arbitrary networks by correlation in informants’ data, the properties of the function to be computed, communication model, network connectivity graph, and routing strategies. In this sense, it paves the way for the development of hitherto elusive general theory of distributed function computation over arbitrary networks.

In Section III, we revisit the notion of information ambiguity, a worst-case information measure we proposed in [4], and extend it to a form useful in the present context. In Section V, we introduce the communication model we assume and our formulation of the distributed function computation problem. The next section introduces a communication protocol to compute any given function at the sink, proves its optimality, and provides the bounds on its performance. Finally in Section VII, we discuss some properties of distributed function computation problem and propose a dichotomous classification of functions, based on the number of informant bits required, in general, to compute those at the sink.

\(^1\)DSC problem is a special case of distributed function computation problem where the function to be computed is identity map, \( id_X \).
II. RELATED WORK

There are three major existing approaches to address the distributed function computation problem, as follows:

Communication complexity: The seminal paper by Yao [2] introduced the problem of computing the minimum number of bits exchanged between two processors when both the processors compute a function of the input that is split between them. Numerous variants of this problem have been explored in the field of communication complexity, [1]. This work provides insights into developing efficient communication protocols for function computation. However, it is mainly interested in estimating the order-of-magnitude of the bounds on communication and computation costs. Also, it is not obvious how to extend this work, when for example, one or more nodes in the network compute some function of source nodes’ data or the source data is split among more than two nodes and is possibly correlated.

Scaling laws: Recently in [5]–[7], the distributed function computation problem has been addressed to characterize the scaling behavior of the rate of function computation with the network size, under simple models of communication and computation in the network. This approach however is not amenable to easily incorporate stronger models of computation and communication, such as interactive communication, data-buffers, and correlated and cooperating sources.

Information theory: Slepian and Wolf in [8] introduced the DSC problem. However, it was many years before distributed function computation problem was seriously addressed in information-theoretic setup, [9]–[12]. Still, there is very little such work that comprehensively addresses the distributed function computation problem over any given network, function, and model of communication and computation.

III. INFORMATION AMBIGUITY FOR DISTRIBUTED FUNCTION COMPUTATION

To perform the worst-case analysis of distributed function computation problem, we revise and generalize some relevant definitions and properties of information ambiguity, a worst-case information measure we introduced in [4].

Note: The logarithms used in this paper are to the base 2.

Consider a $N$-tuple of random variables $(X_1, \ldots, X_N) \sim P = p(\mathcal{X})$, where $\mathcal{X} = (x_1, \ldots, x_N), i \in \{1, \ldots, N\}, X_i \in \mathcal{X}$, and $\mathcal{X}$ is discrete and finite alphabet of size $|\mathcal{X}|$. The support set of $(X_1, \ldots, X_N)$ is defined as:

$$S_{X_1, \ldots, X_N} \text{ def } = \{\mathcal{X} | p(\mathcal{X}) > 0\} \quad (1)$$

We also call $S_{X_1, \ldots, X_N}$ the ambiguity set of $(X_1, \ldots, X_N)$. The cardinality of $S_{X_1, \ldots, X_N}$ is called ambiguity of $(X_1, \ldots, X_N)$ and denoted as $\mu_{X_1, \ldots, X_N} = |S_{X_1, \ldots, X_N}|$. So, the minimum number of bits required to describe an element of $S_{X_1, \ldots, X_N}$, in the worst-case, is $\lceil \log \mu_{X_1, \ldots, X_N} \rceil$.

The support set $S_{X_i}$ of $X_i, i \in \{1, \ldots, N\}$, is the set

$$S_{X_i} \text{ def } = \{x_i : \text{for some } x_{-i}, (x_{-i}, x_i) \in S_{X_1, \ldots, X_N}\}, \quad (2)$$

of all possible $X_i$ values, with $x_{-i} = \{x_1, \ldots, x_N\} \setminus x_i$. We also call $S_{X_i}$ ambiguity set of $X_i$. The ambiguity of $X_i$ is defined as $\mu_{X_i} = |S_{X_i}|$. The conditional ambiguity set of $(X_1, \ldots, X_N)$, when $X_i$ assumes the value $x_i, x_i \in S_{X_i}$, is

$$S_{X_1, \ldots, X_N|X_i}(x_i) \text{ def } = \{\mathcal{X} : \mathcal{X} \in S_{X_1, \ldots, X_N}, x_i \in S_{X_i}\}, \quad (3)$$

the set of possible $(X_1, \ldots, X_N)$ values when $X_i = x_i$. The conditional ambiguity in that case is $\mu_{X_1, \ldots, X_N|X_i}(x_i) = |S_{X_1, \ldots, X_N|X_i}(x_i)|$, the number of possible values of $(X_1, \ldots, X_N)$ when $X_i = x_i$. The maximum conditional ambiguity of $(X_1, \ldots, X_N)$ is

$$\mu_{X_1, \ldots, X_N|X_i}(x_i) \text{ def } = \sup \{\mu_{X_1, \ldots, X_N|X_i}(x_i) : x_i \in S_{X_i}\}, \quad (4)$$

the maximum number of $(X_1, \ldots, X_N)$ values possible with any value that $X_i$ can take.

In fact, for any two subsets $A_X$ and $B_X$ of $\{X_1, \ldots, X_N\}$, such that $A_X \cup B_X \subseteq \{X_1, \ldots, X_N\}$ and $A_X \cap B_X = \phi$, we can define for example, ambiguity set $S_{A_X}$ of $A_X$, conditional ambiguity set $S_{A_X|B_X}(x_B)$ of $A_X$ given the set $x_B$ of values that $B_X$ can take, and maximum conditional ambiguity set $S_{A_X|B_X}(x_B)$ of $A_X$ for any set of values that $B_X$ can take, with corresponding ambiguity, conditional ambiguity, and maximum conditional ambiguity given by $\mu_{A_X}, \mu_{A_X|B_X}(x_B)$, and $\mu_{A_X|B_X}(x_B)$, respectively. However, for the sake of brevity, we do not introduce these definitions here.

Further, let us represent each of $\mu_{X_i}$ values that random variable $X_i$ can take in $[\log \mu_{X_i}]$ bits as $b_i \ldots b_{[\log \mu_{X_i}]}$. Let binary $(x_i)$ represent the value of $j$th, $1 \leq j \leq [\log \mu_{X_i}]$, bit-location in the bit-representation of $x_i$. Then, knowing that the value of $j$th bit-location is $b$, we can define the set of possible values that $X_i$ can take as

$$S_{X_i|b_j}(b) \text{ def } = \{x_i : x_i \in S_{X_i} \text{ and binary }_j(x_i) = b\}, \quad (5)$$

with corresponding cardinality denoted as $\mu_{X_i|b_j}(b)$. We can similarly define $S_{X_i|b_j}(b)$ with $X_i \in A_X$ as

$$S_{X_i|b_j}(b) \text{ def } = \{x_A : x_A \in S_{A_X} \text{ and binary }_j(x_i) = b\}, \quad (6)$$

with corresponding cardinality denoted as $\mu_{A_X|b_j}(b)$. The definitions in (5) and (6) can be easily extended further to the situations where the values of one or more bit-locations in one or more random variable’s bit-representation are known.

Next, we introduce the notion of the ambiguity set and ambiguity of the function output values. The support-set of output values of some function $f$, also called ambiguity set of function output values of function $f$, is defined as:

$$S_f \text{ def } = \{f(\mathcal{X}) : \text{ for some } \mathcal{X} \in S_{X_1, \ldots, X_N}\} \quad (7)$$

The cardinality of $S_f$ is called ambiguity of output values of function $f$ and denoted as $\mu_f = |S_f|$. So, the minimum number of bits required to describe an element in $S_f$ is $[\log \mu_f]$. The conditional ambiguity set of function output values when $X_i = x_i, x_i \in S_{X_i}, i \in \{1, \ldots, N\}$, is defined as

$$S_f|X_i(x_i) \text{ def } = \{f(\mathcal{X}) : \text{ for some } \mathcal{X} \in S_{X_1, \ldots, X_N|X_i}(x_i)\} \quad (8)$$

The corresponding cardinality is called conditional ambiguity of function output values when $X_i = x_i$ and denoted as.
We can further define the maximum conditional ambiguity of function output values as
\[
\hat{\mu}_{f|X_i}(x_i) \overset{\text{def}}{=} \sup \{\mu_{f|X_i}(x_i) : x_i \in S_{X_i}\}
\] (9)
maximum number of function output values possible over any value that \(X_i\) can take over \(S_{X_i}\). The definitions in (8) and (9) can be similarly extended to the situations where the conditioning is carried out over a subset \(A\) of \(X_1, \ldots, X_N\).

Further, when the value of \(f\) in the binary-representation of \(x_i, x_i \in S_{X_i}\), is known, that is \(b_{j_i} = b\), we can define corresponding conditional ambiguity set of function output values as follows
\[
S_{f|b_{j_i}}(b) \overset{\text{def}}{=} \{f(\bar{X}) : \bar{X} \in S_{X_1}, \ldots, X_N, x_i \in S_{X_i|b_{j_i}}(b)\},
\] (10)
with corresponding cardinality denoted as \(\mu_{f|b_{j_i}}(b)\).

Finally, if for every \(X_A, X_A \subset \{X_1, \ldots, X_N\}, f(X_A)\) is defined, then for the support-set \(S_{X_1}, \ldots, X_N\) of data-vectors, the functional \([\log \mu_f]\) is a valid information measure as it then satisfies various axioms of such measures, such as expansibility, monotonicity, symmetry, subadditivity, and additivity, [13]. However, we omit the details of proof for the sake of brevity.

IV. NOTATION

We summarize the notation used frequently in the rest of this paper.

\(N\): number of informants.
\(\mathcal{X}\): discrete and finite alphabet set of cardinality \(|\mathcal{X}|\).
\(\mathcal{P}\): \(N\)-dimensional discrete probability distribution, \(\mathcal{P} = p(x_1, \ldots, x_N), x_i \in \mathcal{X}\).
\(X_i\): random variable observed by \(i^{th}\) informant. \(X_i \in \mathcal{X}\).
\(S_{X_i}\): ambiguity set at the sink of \(i^{th}\) informant’s data, with corresponding ambiguity \(\mu_{X_i} = |S_{X_i}|\).
\(S_{X_1, \ldots, X_N}\): ambiguity set at the sink of all informants’ data, with ambiguity \(\mu_{X_i} = |S_{X_1, \ldots, X_N}|\).
\(S_{X_1, \ldots, X_N|I}\): conditional ambiguity set at the sink of all informant’s data, when sink has information \(I\), with conditional ambiguity \(\mu_{X_i|I} = |S_{X_i|I}|\). The exact nature of \(I\) will be obvious from the context.
\(S_f\): ambiguity set at the sink of the output values of function \(f\), with corresponding ambiguity \(\mu_f = |S_f|\).
\(S_{f|I}\): conditional ambiguity set at the sink of the output values of function \(f\) when sink has information \(I\), with conditional ambiguity \(\mu_{f|I} = |S_{f|I}|\).
\(#f\): minimum number of informant bits required in the worst-case to compute the function \(f\) at the sink.
\(#DSC\): minimum number of informant bits required in the worst-case to solve the DSC problem at the sink.

V. DISTRIBUTED FUNCTION COMPUTATION IN ASYMMETRIC COMMUNICATION SCENARIOS

Let us consider a distributed function computation scenario, where a sink computes some function of the data of \(N\) correlated informants. We assume the asymmetric communication, where the joint distribution \(\mathcal{P}\) of informants’ data is known only to the sink. The Figure 1 depicts this scenario for \(N = 2\).
Assume that \( X_1 \) and \( X_2 \) are random variables. Let \( S_1 \) and \( S_2 \) be the support sets of \( X_1 \) and \( X_2 \), respectively, with \( (X_1, X_2) \) derived from the support-set in first column. Let the function \( f \) being computed at the sink be ‘bitwise OR’ of the instance of \( (X_1, X_2) \) revealed to the informants. For the given support-set, at least \( \lfloor \log_2 \mu_{X_1, X_2} \rfloor = 4 \) bits are required to describe any element of \( S_{X_1, X_2} \) and at least \( \lfloor \log_2 \mu_f \rfloor = 3 \) bits are required to describe any element of \( S_f \). Also, to individually describe any value assumed by \( X_1 \) and \( X_2 \), it requires 3 bits.

For any given support-set of data-vectors, sink can construct a problem-encoding as in Figure 2. It knows that one string, hitherto unknown, from the fourth column is drawn, with first \( \lfloor \log_2 \mu_{X_1} \rfloor \) bits given to informant 1, next \( \lfloor \log_2 \mu_{X_2} \rfloor \) bits given to informant 2, and so on.

**Note on the terminology:** We call a bit-location in the bit-string at an informant (as well as in the bit-representation of \( \overline{X} \) in encoding scheme defined above) defined, if the sink knows its value unambiguously, otherwise it is called undefined. For example, until the sink learns of the actual \( \overline{X} \) revealed to the informants, one or more bits in the \( \sum_{i=1}^{N} \lfloor \log_2 \mu_{X_i} \rfloor \) bits long representation of \( \overline{X} \), remain undefined. Similarly, a bit-location in the bit-representation of the output of the function \( f \) is called evaluated if the sink can exactly compute its value based on the values of one or more bits in informant strings.

VI. COMMUNICATION PROTOCOL FOR DISTRIBUTED FUNCTION COMPUTATION

We address the distributed function computation problem of the last section, in *bit-serial* communication scenario, where in each communication round, only one informant can send only one bit to the sink. This is an example of scenarios where communication takes place over a channel with uplink throughput constrained to one bit per channel use. We are interested in this communication model as it allows us to compute the minimum number of informant bits (total and individual) required to compute \( f(\overline{X}) \) at the sink when any number of rounds and sink bits can be used. Further, it also enables us to compute the worst-case achievable rate-region for this problem, as we show later in this section.

We provide a constructive solution of the distributed function computation problem of the last section, based on interactive communication. The proposed protocol optimally solves this problem and computes the worst-case achievable rate-region. We call the proposed protocol “bit-serial function Computation (bSerfComp)” protocol and describe it next.

A. The bSerfComp protocol

**Algorithm: bSerfComp**

1. \( l = 0 \)
2. Let \( S^l_{X_1, \ldots, X_N} = S_{X_1, \ldots, X_N} \)
3. Let \( S^l_f = S_f, \mu^l_f = |S^l_f| \)
4. Let \( V = \{1, \ldots, \sum_{i=1}^{N} \lfloor \log_2 \mu_i \} \}
5. Let \( U \) be the set of undefined bits in \( V \), \( U \subseteq V \), over all \( \overline{X} \in S^l_{X_1, \ldots, X_N} \)
6. **while** \( (\mu^l_f > 1) \)
7. \( K^{l+1} = \arg \min_{j \in U} \max_{b(j) \in \{0, 1\}} \mu^l_f[b(j)] \)
8. Choose the bit-location corresponding to \( k^{l+1} \), where \( k^{l+1} \) is a randomly chosen element of \( K^{l+1} \)
9. The sink asks the informant corresponding to bit-location \( k^{l+1} \) to send the bit-value \( b(k^{l+1}) \)
10. Set \( S^{l+1}_{X_1, \ldots, X_N} = S^l_{X_1, \ldots, X_N} | b(k^{l+1}) \)
11. Set \( S^{l+1}_f = S^l_f[b(k^{l+1})] \)
12. Compute \( U \subseteq V \), the set of undefined bits
13. \( l = l + 1 \)
In the bSerfComp protocol, in each communication round only one bit is sent by the informant chosen to communicate with the sink. The chosen bit has the property that it divides the size of the current conditional ambiguity set of function output values, at the sink, closest to half. Formally, in terms of the problem statement and encoding introduced in the last section, if $U$ is the set of undefined bits in $\sum_{i=1}^{N}[\log \mu_{X_i}]$ bits long representation of $X$, then the bit chosen in $l^{th}, l \geq 0$, round is the one that solves $\arg\min_{j, b^{l} \in \{0, 1\}} \max_{(j) \in \{0, 1\}} \mu_{f}\left(b^{l},(j)\right)$. The sink, after receiving the value of the chosen bit, recomputes the set of undefined bits $U$. This is carried out iteratively until all $[\log \mu_f]$ bits in the representation of $f(X)$ are not evaluated.

The sink can perform the worst-case performance analysis of the bSerfComp protocol by selecting on the line 9, $b^*(l+1)$ that solves: $b^*(l+1) = \arg\max_{s \in \{0, 1\}} \mu_f\left(b^{(l+1)} \right)$. Note that there are two versions of the bSerfComp protocol. In the online version, the sequence of queries from the sink to the informants is determined adaptively depending on the informant response in the previous rounds, while in the offline version, for a given support-set of data-vectors the entire sequence of queries is determined before actual querying starts. For example, the sequence of queries for the worst-case analysis of the protocol corresponds to the offline version.

B. Optimality of bSerfComp protocol

The binary representations of the elements of $\mathcal{S}_f$, as in Figure 2.e, can be arranged as the leaves of a binary tree, where ambiguity set of function output values $\mathcal{S}_f$ forms the root and conditional ambiguity sets of function output values form internal nodes and leaves. The set of function output values corresponding to a child node is obtained by conditioning the set of function output values corresponding to its parent node on the value $b, b \in \{0, 1\}$ of $b^j$: $j$th bit-location in the binary string revealed to $i$th informant, with ‘$b = 0$’ leading to the left subtree and ‘$b = 1$’ leading to the right subtree. Such a binary tree with $\mu_f$ leaves will have a minimum-height of $[\log \mu_f]$, requiring at least $[\log \mu_f]$ bits to describe any leaf.

Now, we state without proof three lemmas which together allow us to prove the optimality of the bSerfComp protocol.

**Lemma 1:** bSerfComp protocol computes all minimum-height binary trees corresponding to the given support-set to exactly evaluate a given function $f$.

**Lemma 2:** bSerfComp protocol computes $b_i$, the minimum number of bits that the $i^{th}, i \in \{1, \ldots, N\}$, informant must send to the sink exactly evaluate the function $f$.

**Lemma 3:** For a given support-set, each corner point of the worst-case achievable rate-region for computing $f$ corresponds to at least one minimum-height binary tree, with height $\#f$.

**Theorem 1:** For a given support-set, bSerfComp protocol computes the worst-case achievable rate-region for function $f$.

**Proof:** Combining the statements of Lemmas 1 and 3, we can state that bSerfComp protocol computes each corner point of the worst-case achievable rate-region, thus establishing the worst-case achievable rate-region for computing $f$.

The worst-case achievable rate-region for distributed computation of function $f$ in asymmetric communication scenarios is given by the following corollary to Theorem 1. For the sake of notational simplicity, we state it only for $N = 2$.

**Corollary 1:** For $N = 2$, if $b_i$ is the minimum number of bits that an informant $i$, $1 \leq i \leq 2$, sends over all solutions of bSerfComp protocol and $\#f$ is the number of bits sent by all informants, then the worst-case achievable rate region is:

$$
\begin{align*}
R_1 &\geq b_1 \\
R_2 &\geq b_2 \\
R_1 + R_2 &\geq \#f
\end{align*}
$$

In Figures 3-4, using bSerfComp protocol, we compute the worst-case achievable rate-regions for functions: ‘bitwise OR’, ‘bitwise AND’, and ‘bitwise XOR’, evaluated at sink over two support-sets of data-vectors for two correlated informants.

C. Performance bounds for bSerfComp protocol

To compute the bounds on the performance of bSerfComp protocol, we make use of an interesting and important observation based on its working.

**Observation:** A bit in the binary-representation of the function output values can be evaluated, without all bits in the concatenated bit-representation of $X$ being defined.

Let $\#f$ and $\#_{DSC}$ denote the minimum number of informant bits required, in the worst-case, to evaluate the function $f$ and to solve the DSC problem, respectively, at the sink for a given support-set of data-vectors.

**Loose Bounds:** As we discussed before, $\#f$ is bounded from below by $[\log \mu_f]$, that is: $[\log \mu_f] \leq \#f$.

Let $\mu_s$ be the number of data-vectors or informant strings which evaluate to the function output value $s, s \in \mathcal{S}_f$. Also, let us define $\mu_s^{\min} = \min_{s \in \mathcal{S}_f} \mu_s$. Then, assuming that $f$ evaluates to the output that corresponds to $\mu_s^{\min}$, we obtain an upper bound on $\#f$ as: $\#f \leq \#_{DSC} - [\log \mu_s^{\min}]$.

Therefore, we have the following loose bound on $\#f$:

$$
[\log \mu_f] \leq \#f \leq \#_{DSC} - [\log \mu_s^{\min}] \tag{11}
$$

**Tight bounds:** For a given support-set of data-vectors, we know that $[\log \mu_f] \leq \#f$. Let us assume that the sink has obtained $[\log \mu_f]$ informant bits using bSerfComp protocol. Let assume that the size of conditional ambiguity set of data-vectors at the end of $l^{th}$ round, $1 \leq l \leq [\log \mu_f]$, is $1/\epsilon^l$ of its size at the beginning of this round. Define $\epsilon_{\max} = \max\{\epsilon_1, \ldots, \epsilon_{[\log \mu_f]}\}$. Then, the size of conditional ambiguity set of data-vectors after $[\log \mu_f]$ rounds satisfies:

$$
\frac{\mu_{X_1 \ldots X_N}}{2^{\sum_{l=1}^{[\log \mu_f]}(1-\epsilon_l)}} \leq \frac{\mu_{X_1 \ldots X_N}}{2^{1-\epsilon_{\max}[\log \mu_f]}}
$$

Now, if $\frac{\mu_{X_1 \ldots X_N}}{2^{1-\epsilon_{\max}[\log \mu_f]}} \leq \mu_s^{\min}$, then the function output evaluation finishes with $[\log \mu_f]$ to $[\log \mu_f] + [\log \mu_s^{\min}]$ informant bits. So, we have

$$
[\log \mu_f] \leq \#f \leq [\log \mu_f] + [\log \mu_s^{\min}] \tag{12}
$$

2Computation of bitwise functions over two binary strings is well-known, [17]. Let $B(X_1, X_2)$ denote the output of the bitwise function $B$ evaluated over binary-strings corresponding to $X_1$ and $X_2$, $i, j \in \{1, \ldots, N\}, i \neq j, N \geq 2$. Define $B(X_1) = X_1$. Then, $B(X_1, \ldots, X_N)$ can be recursively defined, for example, as: $B(X_1, \ldots, X_N) = B(B(X_1, \ldots, X_{N-1}), X_N)$.
and in this case the lower bound in (11) is actually tight.

Otherwise, if \( \frac{\mu_{X_1, \ldots, X_N}}{2(1+\epsilon_{max})|\log \mu_f|} > \mu_{\text{min}} \), then the function computation finishes in \( [\log \mu_f] + [\log \mu^*] \) to \( [\log \mu_f] + [\log \frac{\mu_{X_1, \ldots, X_N}}{2(1+\epsilon_{max})|\log \mu_f|}] \) bits, where \( \mu^* \) is the size of smallest subset of function output values that satisfies

\[
\frac{\mu_{X_1, \ldots, X_N}}{2(1+\epsilon_{max})|\log \mu_f|} \leq \sum_{S \in \mathbb{S}, S \subseteq S_f, |S| = \mu^*} \mu_s
\]

Therefore, in this case we have:

\[
[\log \mu_f] + [\log \mu^*] \leq \#f \leq [\log \mu_f] + [\log \frac{\mu_{X_1, \ldots, X_N}}{2(1+\epsilon_{max})|\log \mu_f|}]
\]

VII. SOME PROPERTIES OF DISTRIBUTED FUNCTION COMPUTATION

We discuss some of the significant results, properties, and observations based on our work on distributed function computation problem in asymmetric communication scenarios.

A. Two Classes of Functions

Let us consider two functions \( g = \max\{X_1, X_2\} \) and \( h = X_1 + X_2^2 \). For two or more data-vectors derived from any discrete and finite support-set, the function \( g \) may evaluate to the same output value, while the function \( h \) assigns, in general, a unique output value to each of its input pairs. Generalizing this to the functions of \( N, N \geq 2 \), variables computed over corresponding discrete and finite support-sets, there are various functions whose behavior is like either \( g \) or \( h \) above.

The common statistical functions, such as ‘max’, ‘min’, ‘majority’, ‘mean’, ‘median’, and ‘mode’ and logical functions, such as ‘parity’, ‘bitwise OR’, ‘bitwise AND’, and ‘bitwise XOR’ belong to a class of functions, which we call lossy functions. Similarly, the functions such as ‘identity map’, ‘iterated exponentiation’, belong to a class of functions, which we call lossless functions. Formally, for the lossy functions the cardinality of their range is smaller than the cardinality of their domain, while for the lossless functions two cardinalities are equal. In fact, it is easy to prove that the equality of the sizes of domain and range of a function is an equivalence relation and classes of lossy and lossless functions are equivalence classes.

These two equivalence classes of functions are relevant in the discussion of distributed function computation because, in general, the computation of lossy functions at the sink requires fewer number of informant bits than computation of lossless functions. The DSC problem belongs to the equivalence class of lossless functions (DSC is distributed function computation with function being the identity map: \( id_x \)). This implies that, in general, for a given support-set the computation of lossless functions requires as many informant bits, in the worst-case, as the solution of DSC problem, while the computation of lossy functions requires fewer bits than DSC.

The \textbf{bSerfComp} protocol of the last section can be used to compute both, the lossy and lossless functions. However, as for the lossless functions, the \textbf{bSerfComp} protocol reduces to much simpler \textbf{bSerCom} protocol of [3] for computing DSC.
in the corresponding communication scenario, the latter can be deployed at the sink for their computation.

Also, for the \textit{lossless} functions the knowledge of function output allows us to uniquely determine the input data-vector revealed to the informants (reversible function computation), while for the \textit{lossy} functions this is not possible (irreversible function computation). This \textit{apparent} loss of information accompanying the computation of \textit{lossy} functions results in their computation with fewer number of informant bits, but at the cost of sacrificing our ability to recover the input data-vector from their output. For the \textit{lossless} functions, there is no such information loss in their computation, allowing the unambiguous recovery of the input data-vectors from their output, but at the cost of larger number of informant bits.

It should be noted that above classification of functions holds true for any given support-set of data-vectors, in general. However, one can always concoct exceptions where the cardinality of the support-set of function output values for some \textit{lossy} function is same as the cardinality of the corresponding support-set of data-vectors. Similarly, some exception for \textit{lossless} functions can be constructed, where the cardinality of the support-set of function output values is smaller than the cardinality of corresponding support-set of data-vectors. We state without proof that the number of such instances of support-sets is small for any given cardinality of the support-sets. Further, in all situations the following lemma, which we state without proof, always holds for any function \( f \).

\textbf{Lemma 4}: If \([ \log \mu_f ] = [ \log \mu_{\text{DSC}} ]\), then \( \# f = \# \text{DSC} \), otherwise \( \# f \leq \# \text{DSC} \).

This brings us to relating our work on function classification with Han and Kobayashi’s work along similar lines, [9]. We establish two equivalence classes of functions: \textit{lossy} and \textit{lossless}. Given that DSC problem belongs to the class of \textit{lossy} functions, the worst-case achievable rate-region of \textit{lossless} functions coincides with the worst-case rate-region of DSC problem, while for \textit{lossy} functions it is correspondingly larger. In [9] too, the authors have introduced such dichotomy of functions of correlated sources: for one class of functions the achievable rate-region coincides with Slepian-Wolf rate-region and for another class it does not. However, in spite of apparent similarities in results, there are some basic differences in approach. First, we are interested in the worst-case information-theoretic analysis while [9] is concerned with average-case analysis. Second, our results pertain to \textit{one-shot} function computation, while [9] deploys block-encoding.

It is interesting to ask if for a given communication scenario, we can always construct two or more equivalence classes of functions based on the communication cost of their computation. This appears to be a largely unexplored problem and a systematic answer that addresses it and also unifies various previous attempts to solve it, warrants our attention. Further, we can refine proposed two classes of the functions based on finer details of the functions and we actually expect the classification structure to be richer than just the dichotomous classification in the paper.

B. Dependence of \( \# f \) on \( \mu_f \) and \( \mu_{\text{DSC}} \)

In subsection VI-C, we established how for a given support-set of data-vectors \( \# f \), the minimum number of informant bits needed to compute the function \( f \) in the worst-case, depends on \( \mu_f \), the ambiguity of function output values and \( \mu_{X_1, \ldots, X_N} \), the ambiguity of data-vectors. Now, let us consider how for a given function \( f \), \( \# f \) for two different support-set of data-vectors depends on corresponding \( \mu_f \) and \( \mu_{X_1, \ldots, X_N} \).

Let \( \mu_{\text{DSC}}^1 \) and \( \mu_{\text{DSC}}^2 \) denote the cardinality of the set of function output values for first and second support-set, respectively.

Let \( \mu_{\text{DSC}}^1 \) and \( \mu_{\text{DSC}}^2 \) denote the cardinality of the set of data-vectors for first and second support-set, respectively.

Finally, let \( \# f^1 \) and \( \# f^2 \) denote the minimum number of informant bits required to compute the function \( f \) for first and second support-set, respectively.

We establish the relation between \( \# f^1 \) and \( \# f^2 \), given the relations between \( \mu_{\text{DSC}}^1 \) and \( \mu_{\text{DSC}}^2 \), and \( \mu_{\text{DSC}}^1 \) and \( \mu_{\text{DSC}}^2 \). We provide an exhaustive list of various possibilities and provide an example for each when the sink computes \( \max \{ X_1, X_2 \} \) over data values of two informants for a given support-set.

\textbf{Property 1}: \( \mu_1^1 = \mu_2^1, \mu_{\text{DSC}}^1 = \mu_{\text{DSC}}^2 \Rightarrow \# f^1 = \# f^2 \). This deficiency of the notion of ambiguity is illustrated with an example in Figures 5.(a)-(b) for which \( \# f^1 \neq \# f^2 \). In this case, it is clear that ambiguities corresponding to function output values and data-vectors alone cannot be used to establish the relation between \( \# f^1 \) and \( \# f^2 \). This deficiency of the notion of ambiguity is addressed in greater detail in one of our related papers, [18].

\textbf{Property 2}: \( \mu_1^1 = \mu_2^2, \mu_{\text{DSC}}^1 \neq \mu_{\text{DSC}}^2 \Rightarrow \# f^1 \neq \# f^2 \). This property follows the ordering of \( \mu_1^1 \) and \( \mu_2^2 \) as in Figures 6.(a)-(b).

\textbf{Property 3}: \( \mu_1^1 \neq \mu_2^2, \mu_{\text{DSC}}^1 = \mu_{\text{DSC}}^2 \Rightarrow \# f^1 = \# f^2 \). This property follows the ordering of \( \mu_1^1 \) and \( \mu_2^2 \) as in Figures 6.(c)-(d).

\textbf{Property 4}: \( \mu_1^1 < \mu_2^2, \mu_{\text{DSC}}^1 < \mu_{\text{DSC}}^2 \Rightarrow \# f^1 \leq \# f^2 \). Support-sets in Figures 6.(c) and 5.(c) illustrate this.

\textbf{Property 5}: \( \mu_1^1 < \mu_2^2, \mu_{\text{DSC}}^1 > \mu_{\text{DSC}}^2 \Rightarrow \# f^1 \leq \# f^2 \). Support-sets in Figures 6.(d) and 5.(d) illustrate this.

VIII. CONCLUSIONS AND FUTURE WORK

We address the distributed function computation problem in asymmetric and interactive communication scenarios, where the sink is interested in computing some function of input data that is split among \( N \) correlated informants and is derived from some discrete and finite distribution. We consider the distributed function computation as a generalization of distributed source coding problem. We are mainly interested in computing \( \# f \), the minimum number of informant bits required in the worst-case, to allow the sink to exactly compute the given function. We provide \textbf{bSerfComp} protocol to optimally compute the functions at sink for any given support-set of data-vectors, prove that it computes the worst-case achievable rate-region for computing any given function, and illustrate this with examples. Also, we provide a set of bounds on the performance of the proposed protocol.

We define two equivalence classes of functions: \textit{lossy} and \textit{lossless}. We show that the \textit{lossy} functions can be computed, in general, with fewer number of informant bits than\textit{lossless}.
function, such as DSC. Further, we establish the dependence of $\#f$, when the function $f$ is computed over two different support-sets, on the relation between their respective ambiguities of function output values and data-vectors.

In future, we want to extend this work in three interesting directions. First, in this paper we have assumed that the sink and informants directly communicate with each other. Allowing the sink and informants to indirectly communicate with each other over one or more intermediate nodes (as in multihop networks), offers many more opportunities of reducing the number of bits carried over the network to compute a function at the sink. Second, allowing the sink to tolerate certain amount of error in the computation of the function may reduce the number of informant bits required. Finally, we want to come up with a generic framework to classify the functions based on the communication costs of their computation over arbitrary networks with any given model of communication and computation.

REFERENCES

[1] E. Kushilevitz and N. Nisan, Communication Complexity, Cambridge Univ. Press, Cambridge, UK, 1997.
[2] A. C. Yao, “Some complexity questions related to distributed computing,” Proc. ACM STOC 1979, Atlanta, GA, April-May 1979.
[3] S. Agnihotri and H. S. Jamadagni, “Worst-case asymmetric distributed source coding,” Proc. Allerton 2008, Monticello, IL, September 2008.
[4] S. Agnihotri and H. S. Jamadagni, “Information Ambiguity,” Proc. ISITA 2008, Auckland, NZ, December 2008.
[5] A. Giridhar and P. R. Kumar, “Computing and communicating functions over sensor networks,” IEEE JSAC, vol. 23, April 2005.
[6] N. Khude, A. Kumar, and A. Karnik, “Time and energy complexity of distributed computation of a class of functions in wireless sensor networks,” IEEE Trans. Mob. Comput., vol. 7, May 2008.
[7] S. Kamath and D. Manjunath, “On distributed function computation in structure-free random networks,” Proc. IEEE ISIT 2008, Toronto, Canada, July 2008.
[8] D. Slepian and J. K. Wolf, “Noiseless coding of correlated information sources,” IEEE Trans. Inform. Theory, vol. IT-19, July 1973.
[9] T. S. Han and K. Kobayashi, “A dichotomy of functions $f(X, Y)$ of correlated sources $(X, Y)$ from the viewpoint of the achievable rate region,” IEEE Trans. Inform. Theory, vol. IT-33, January 1987.
[10] R. Gallager, “Finding parity in a simple broadcast network,” IEEE Trans. Inform. Theory, vol. IT-34, March 1988.
[11] L. J. Schulman, “Coding for interactive communication,” IEEE Trans. Inform. Theory, vol. IT-42, November 1996.
[12] A. Orlitsky and J. R. Roche, “Coding for computing,” IEEE Trans. Inform. Theory, vol. IT-47, March 2001.
[13] G. J. Klir, Uncertainty and Information: Foundations of Generalized Information Theory, John Wiley & Sons, 2006.
[14] J. Chou, D. Petrovic, and K. Ramchandran, “A distributed and adaptive signal processing approach to exploiting correlation in sensor networks,” Journal of Ad Hoc Networks, vol. 2, October 2004.
[15] M. Adler, “Collecting correlated information from a sensor network,” Proc. SODA 2005, Vancouver, Canada, January 2005.
[16] A. Orlitsky, “Worst-case interactive communication I: Two messages are almost optimal,” IEEE Trans. Inform. Theory, vol. IT-36, September 1990.
[17] B. W. Kernighan and D. M. Ritchie, The C Programming Language, 2nd ed., Prentice-Hall Inc., 1988.
[18] S. Agnihotri and V. Rajesh, “Worst-case compressibility of discrete and finite distributions,” Available as arXiv:0907.1723.