Veracity Radius - Capturing the Locality of Distributed Computations*

Yitzhak Birk
Elec. Eng. Dept.
Technion
Haifa 32000, Israel
birk@ee.technion.ac.il

Idit Keidar
Elec. Eng. Dept.
Technion
Haifa 32000, Israel
idish@ee.technion.ac.il

Liran Liss
Elec. Eng. Dept.
Technion
Haifa 32000, Israel
liranl@tx.technion.ac.il

Assaf Schuster
Comp. Sci. Dept.
Technion
Haifa 32000, Israel
assaf@cs.technion.ac.il

Ran Wolff
Comp. Sci. Dept.
University of Maryland
Baltimore County (UMBC)
ranw@cs.umbc.edu

ABSTRACT

This paper focuses on *local* computations of distributed aggregation problems on fixed graphs. We define a new metric on problem instances, *Veracity Radius (VR)*, which captures the inherent possibility to compute them locally. We prove that VR yields a tight lower bound on output-stabilization time, i.e., the time until all nodes fix their outputs, as well as a lower bound on quiescence time. We present an efficient aggregation algorithm, I-LEAG, which reaches both output stabilization and quiescence within a time that is proportional to the VR of the problem instance, and is also efficient in terms of per-node communication and memory. We empirically show that the VR metric also effectively captures the performance of previously suggested efficient aggregation protocols, and that I-LEAG significantly outperforms these protocols in several respects.

Categories and Subject Descriptors

C.4 [**Performance of Systems**]: [Performance attributes]; G.2.2 [**Graph Theory**]: [Graph labelling, Network problems]; H.3.4 [**Systems and Software**]: [Distributed systems, Performance evaluation]

General Terms

Algorithms, Theory

*This work was supported in part by a grant from the Israel Ministry of Science.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
PODC'06, July 22-26, 2006, Denver, Colorado, USA.
Copyright 2006 ACM 1-59593-384-0/06/0007 ...$5.00.

Keywords

Aggregation, Locality

1. INTRODUCTION

Background. We consider distributed computation by a large set of network nodes, which must collaboratively compute and come to know the value of a global *aggregation function* [24, 16] of their individual inputs. For example, testing whether the number of sensors reporting a problem exceeds a certain threshold (majority vote with threshold), determining the average processor load in a computing grid, etc. Such networks may comprise millions of nodes, and communication bandwidth as well as energy resources may be limited, requiring good scalability. An appealing approach for achieving scalability is using *local computations*, whereby the computation time and the amount of per-node communication are independent of the size of the network.

Unfortunately, virtually all interesting aggregation functions, e.g., majority, AND/OR, average, and minimum, cannot always be computed locally – they have instances that require some nodes to indirectly communicate with a fixed fraction of the graph nodes to deduce the correct result. Yet, they also have instances that intuitively should require little if any computation (e.g., when no sensor reports a problem). Indeed, recent work presents efficient solutions to interesting aggregation problems, and empirically shows that these solutions *usually* perform local computations [4, 23].

Contributions. We strive to formally quantify the efficiency, or achieved locality, of a distributed aggregation algorithm. We rule out the use of average-case analysis, as such analysis must assume a concrete distribution of the inputs, and we do not believe that it is possible to characterize the "typical" input distribution in a general sensor or peer-to-peer network. Instead, we seek a metric for the inherent amenability of a given *problem instance* to local computation, relative to which proposed algorithms can be evaluated.

The main contribution of this paper is such a new formal metric, the *Veracity Radius (VR)*. It is defined using the

notion of an r-neighborhood of a node v, which is the set of all nodes within a radius r from v. Intuitively, if for all r-neighborhoods with $r \geq r_0$ the aggregation function yields the same value as for the entire graph, then there is apparently no need for a node to ever communicate with nodes that are farther than r_0 hops away from it, irrespective of graph size.

The formal VR metric (see Section 3) captures the notion of a minimal such radius r_0, but allows some slack in the subgraphs over which the values of the aggregation function are examined. Specifically, it is parameterized by some slack function $0 \leq \alpha(r) \leq r$, and is defined based on node environments that include an $\alpha(r)$-neighborhood and are included in an r-neighborhood.

VR provides a fine-grained classification of problem instances, which yields an intuitive tight lower bound on computation time. Specifically, for any given aggregation function F, communication graph G, $d \geq 0$, and algorithm solving F on G, there is an instance with $VR_\alpha \leq d$ for which at least $min\{\lfloor \alpha(d) \rfloor, Radius(G)\}$ steps are required before all nodes decide on the correct result. This bound is tight for every d such that $\alpha(d) \leq Radius(G)$: it is achieved by a simple Full Information (FI) algorithm. Unfortunately, while FI attains optimal output stabilization, nodes send messages for a number of steps equal to graph diameter. In certain systems, such as sensor networks, wherein energy-efficiency is critical and/or communication resources are limited, such a solution is inadequate. The challenge is thus to determine when it is safe to *stop sending messages*, i.e., achieve local *quiescence*.

We present an efficient aggregation algorithm (see Section 4), *I-LEAG (Instance-Local Efficient Aggregation on Graphs)*, that addresses this challenge: both I-LEAG's computation and quiescence times are within a constant factor of the above lower bound. Moreover, I-LEAG is communication efficient and has low memory requirements. I-LEAG runs on a precomputed hierarchy of partitions of the graph with special locality properties, which may also be applicable elsewhere. This hierarchy only depends on the graph and not on the aggregation function or its inputs, so it only needs to be computed once. One salient feature of I-LEAG is that explicit communication (sending of messages) ceases once the output stabilizes, thereby reducing both quiescence time and communication.

Finally, we complement the aforementioned theoretical results with a simulation study (see Section 5), which (i) demonstrates that VR adequately predicts the performance of previous algorithms that were shown to be efficient in common instances [4, 23] (as well as I-LEAG's), and (ii) compares these algorithms with I-LEAG. We find that I-LEAG has superior performance by all tested measures.

Related Work. Recently, there is large interest in applications of large-scale distributed systems such as sensor networks [16, 17], peer-to-peer systems [22], and computational grids [20]. Consequently, many methods for collecting and aggregating the vast amount of data contained (or generated) by these networks have been proposed, e.g., [7, 8, 2, 5]. Often, the aggregation result needs to be diffused throughout the entire network, [24]. However, this body of work has seldom focused on locality.

Locality has been considered in the context of distributed graph constructions, e.g., coloring [15, 18] and minimum vertex cover [9], which are significantly different than aggregation. In this paper, we assume a fixed graph, for which any such construction can be computed off line; specific problem instances are then specified only by the inputs at each node. Moreover, such work has primarily focused on worst-case analysis. This restrictive view of locality often limits the applicability of local computation to simplistic problems or to restricted graph models (e.g., [10]).

The notion of an "instance-based" locality was first mentioned by the seminal work of Kutten and Peleg on Fault-local Mending [12], and was later refined by subsequent work on fault tolerance such as Tight Fault Locality [13], Time-adaptive Self-Stabilization [11], Distributed Error Confinement [1], and more. Harnessing local computation whenever possible has also been suggested for location services [14]. None of this work, however, deals with aggregation. The only previous work that addresses instance-local aggregation deals with majority voting [4, 23]. It empirically shows that even a problem that trivially requires global communication for many instances can still be computed locally in the common case, but provides no formal metric or analysis to justify this.

Finally, there are several works that attempt to find meaningful alternatives to the over-restrictive worst-case analysis as well as the often impractical average-case analysis, e.g., [6] and [21]. However, these consider neither a distributed setting nor locality.

2. PRELIMINARIES

2.1 Model and Problem Definition

We model a distributed system as a graph, in which nodes and edges correspond to computing nodes and communication links, respectively. Nodes can exchange information only by sending messages. We assume a fixed, unweighted, undirected communication graph and synchronous operation. Given a set D, we denote a multi-set over D by $\{d_1^{n_1} ... d_m^{n_m}\}$, where $n_i \in \mathbb{N}$ indicates the multiplicity of $d_i \in D$, and the absence of d_i represents $n_i = 0$. We denote the set of multi-sets over D by $\mathbb{N}^D$. Given a graph $G = G(V, E)$ and an *aggregation function* $F{:}\mathbb{N}^D \rightarrow R$, the aggregation problem $P_{G,F}$ is to compute F for every input assignment $I{:}V \rightarrow D$, and to distribute the result $r \in R$ to all nodes.

More formally, every node v has an input value $I_v \in D$ and an output register O_v. Initially, $O_v = \perp$ and v only knows its own input. An algorithm *solves* $P_{G,F}$ if $\forall v \in V$: $O_v = F(I_V)$ after finite time, where I_X denotes the multi-set induced by the projection of I on a set of nodes $X \subseteq V$. Note that since F is defined over multisets, the identity of the nodes does not play a role. For brevity, we also use I to denote I_V when clear from the context.

Let $\mathcal{M}_A(I)$ denote the performance of an algorithm A on input I under measure $\mathcal{M}$, and consider such an execution of A. We focus on the following three known performance measures: (1) *Quiescence time*, $Q_A(I)$, is the time after which no more messages are sent; (2) *Output-stabilization time*, $OS_A(I)$, is the earliest time after which no node changes its output; and (3) *Message Load*, $ML_A(I)$, is the average number of messages per node sent during the execution.

In this paper, we only consider functions $F{:}\mathbb{N}^D \rightarrow R$ where R is a discrete totally-ordered set with at least two values and F satisfies the following properties:

Convexity $\forall X, Y \in \mathbb{N}^D$: $F(X \cup Y) \in \Big[F(X), F(Y)\Big]$

Onto (in singletons) $\forall r \in R, \exists x \in D$: $F(x) = r$.

Convexity ensures that if the graph is partitioned such that the results of computing the aggregation function over every component (in isolation) are all the same, then these results also equal the correct (global) result. This suggests opportunities for locally computing F, based on the following intuition:

> If the (global) correct result of an aggregation function is also evident in every local environment, then it is likely that every node can arrive at this result based solely on the inputs of nodes in its vicinity.

Many interesting functions have these properties. Examples include: Min, Max, Majority, Median, and Consensus (e.g., by using AND/OR functions). However, some important ones such as computing an average (violates discreteness) and counting the number of nodes that satisfy a certain predicate (violates convexity), do not fall into this category. Nevertheless, even these two functions can practically be included in our framework: an average can be computed by rounding the result to the desired precision (regaining discreteness[1]), and given an estimate of the graph size, any counting problem can be expressed as a rounded average problem (regaining convexity) by computing the fraction of the nodes (up to a fixed precision) that satisfy the desired predicate and then multiplying the result by the graph size. The term "aggregation functions" will henceforth be used to refer to functions that possess these two properties.

2.2 Graph Notions

Let $G = G(V, E)$ be a graph. Denote G's diameter and radius by $Diam(G)$ and $Rad(G)$, respectively. We use the following graph-theoretic notation:

Definition 1. (Cluster) A *cluster* is a subset $S \subseteq V$ of vertices whose induced subgraph $G(S)$ is connected. A *weak cluster* is a subset $S \subseteq V$ that is connected in G but not necessarily in $G(S)$. (If G is connected, any set of nodes is a weak cluster.)

Definition 2. (Distance) For every two nodes $v_1, v_2 \in V$, the distance between v_1 and v_2 in G, $dist(v_1, v_2)$, is the length of the shortest path connecting them.

Definition 3. (Neighborhood) The $r-$neighborhood ($r \in \mathbb{R}^+$) of a node v, $\Gamma_r(v)$, is the set of nodes $\{v' \mid dist(v, v') \leq r\}$. $\widehat{\Gamma}(v) = \Gamma_1(v) - \{v\}$ denotes the neighbors of a node v. For a cluster S: $\Gamma_r(S) = \bigcup_{v \in S} \Gamma_r(v)$ and $\widehat{\Gamma}(S) = \Gamma_1(S) - S$.

3. VERACITY RADIUS: AN INHERENT METRIC FOR LOCALITY

We now introduce the Veracity Radius (VR) metric for quantifying the "attainable locality" of an instance (input) of an aggregation problem. We first give formal definitions of VR and the related notion of local complexity. We then establish a tight lower bound on output-stabilization and quiescence times using VR, thereby proving that VR is inherent to local computation.

[1]Note that an exact average over $\mathbb{R}$ can seldom be computed locally.

3.1 Veracity Radius and Local Complexity

Let α:$\mathbb{R}^+ \rightarrow \mathbb{R}^+$ be a continuous, non-decreasing function such that $\alpha(r) \in [\frac{r}{K}, r]$, for some $K \geq 1$. We call such a function a K-bounded *slack function*.

Definition 4. (α-Veracity Radius) Let $P_{G,F}$ an aggregation problem. For every K-bounded slack function α and input I, the α-veracity radius of I is:

$$VR_\alpha(I) \triangleq min\{r \in \mathbb{R}^+ \mid \forall r' \geq r, v \in V, S \subseteq V \text{ s.t.}$$

$$\Gamma_{\alpha(r')}(v) \subseteq S \subseteq \Gamma_{r'}(v)\text{: } F(I_S) = F(I)\}.$$

If $F(I_v) = F(I)$ for every $v \in V$, then $VR_\alpha(I) = 0$. We refer to such I as a *trivial* input assignment. It is important to note that this metric is general enough to cover any aggregation function and graph; yet, it enables reasoning about algorithms that do not always compute their results based on exact neighborhoods. Also, it would seem that $\Gamma_{\alpha(r')}(v) \subseteq S$ should be a sufficient requirement for selecting S. However, this would allow adding arbitrary nodes to $\Gamma_{\alpha(r')}(v)$ from anywhere in the graph, causing $F(I_S)$ to be any value. The terms VR and VR_α are used synonymously throughout the paper.

Given a graph, VR represents a property of the input rather of the graph size, so it can be used as a "locality metric". Any algorithm whose performance depends solely on VR for any graph is infinitely scalable (in the graph's size) for all problem instances with a similar VR. We say that such algorithms have *local complexity*.

Definition 5. (Local Complexity) Let $\mathcal{G}$ be an infinite family of graphs, and $\mathcal{M}$ a performance measure. An algorithm A on $\mathcal{G}$ has local complexity with respect to $\mathcal{M}$ if there exists a slack function α and a function g such that for every $G \in \mathcal{G}$ and non-trivial input I: $\mathcal{M}_A(I) \leq g(VR_\alpha(I))$. In case $g(x) = cx$ for some $c \geq 1$, we say that A is (c, α)-$\mathcal{M}$ local.

In the next section, we derive lower bounds on output stabilization and quiescence times of $LB_d \triangleq min\{\lfloor \alpha(d) \rfloor, Rad(G)\}$ for instances with $VR_\alpha \leq d$ of any aggregation problem. As proven below, both (c, α)-OS and (c, α)-Q local algorithms thus attain the lower bound to within a constant factor.

THEOREM 1. *Given a K-bounded slack function α, for every (c, α)-OS local algorithm A and every non-trivial input instance I: $OS_A(I) \leq 2cK \cdot LB_{VR_\alpha(I)}$. The same holds for quiescence.*

PROOF. Let $P_{F,G}$ be an aggregation problem. For $v \in V$ and $r' \geq 2KRad(G)$, it holds that $\Gamma_{\alpha(r')}(v) = V$ because $\alpha(r') \geq r'/K \geq 2Rad(G) \geq Diam(G)$. So, for every input instance I, $VR_\alpha(I) \leq 2KRad(G)$ by definition. As a result, for $\alpha(VR_\alpha(I)) > Rad(G)$, it holds that $VR_\alpha(I) \leq 2K \cdot LB_{VR_\alpha(I)}$. For $\alpha(VR_\alpha(I)) \leq Rad(G)$:

$$VR_\alpha(I) \leq K \cdot \alpha(VR_\alpha(I)) \leq K(1 + \lfloor \alpha(VR_\alpha(I)) \rfloor) =$$

$$K(1 + LB_{VR_\alpha(I)}).$$

If in addition I is non-trivial, $\alpha(VR_\alpha(I)) \geq 1$ and thus $LB_{VR_\alpha(I)} \geq 1$. Therefore, for every non-trivial input I we obtain that $VR_\alpha(I) \leq 2K \cdot LB_{VR_\alpha(I)}$. Since $OS_A(I) \leq cVR_\alpha(I)$ and $Q_A(I) \leq cVR_\alpha(I)$ for (c, α)-OS and (c, α)-Q local algorithms, respectively, we have the result. $\square$

We conclude that algorithms satisfying our new notion of locality are optimal within a constant factor in all cases, and are extremely appealing for problems that have many instances for which $c \cdot VR$ is substantially smaller than the graph diameter.

3.2 Lower Bound

We now show that VR provides lower bounds for both quiescence and output stabilization times. The proof is based on the renowned indistinguishability argument and builds upon the next two lemmas.

LEMMA 1. *Let F be an aggregation function. For every $c \in \mathbb{N}$, there exist $a, b \in D$ and $n_1, n_2 \in \mathbb{N}$ such that $n_1 + n_2 = c$ and $F(a) = F(a^{n_1}b^{n_2}) \neq F(a^{n_1}b^{n_2+1}) = F(b)$.*

PROOF. We choose $x, y \in D$ such that $F(x) < F(y)$ are consecutive values in R. By convexity:

$$\forall k \in [0, c-1]\text{: } F(x^{c-k}y^k) = F(x^{c-(k+1)} \cup y^k \cup x) \leq$$

$$F(x^{c-(k+1)} \cup y^k \cup y) = F(x^{c-(k+1)}y^{k+1}).$$

Moreover, because $F(x^c) = F(x) < F(y) = F(y^c)$, we conclude that there exists $\overline{k}$ for which the inequality is strict, i.e., $F(x) = F(x^{c-\overline{k}}y^{\overline{k}}) < F(x^{c-(\overline{k}+1)}y^{\overline{k}+1}) = F(y)$. Convexity ensures that $F(x^{c-\overline{k}}y^{\overline{k}+1})$ is either $F(x)$ or $F(y)$. If it is $F(y)$, let $a = x$ and $b = y$, and $n_1 = c - \overline{k}$ and $n_2 = \overline{k}$. Otherwise, let $a = y$ and $b = x$, and $n_1 = \overline{k}+1$ and $n_2 = c - (\overline{k}+1)$. □

LEMMA 2. *Let $P_{G,F}$ be an aggregation problem. For every slack function α and every $d \geq 1$ such that $1 \leq \alpha(d) \leq Rad(G)$, there exist two inputs I and I', such that $VR_\alpha(I) \leq d$, $VR_\alpha(I') \leq d$, $F(I) \neq F(I')$, and at least one node $v_0 \in G$ cannot distinguish between I and I' in fewer than $\lfloor \alpha(d) \rfloor$ steps.*

PROOF. Choose a node $v_0 \in V$ such that $|\Gamma_{\lfloor \alpha(d) \rfloor - 1}(v_0)| = min_{v \in V}|\Gamma_{\lfloor \alpha(d) \rfloor - 1}(v)|$. Let $S = \Gamma_{\lfloor \alpha(d) \rfloor - 1}(v_0)$. Following Lemma 1, we can choose $a, b \in D$ and $n1, n2 \in \mathbb{N}$ such that $n_1 + n_2 = |S|$ and $F(a) = F(a^{n_1}b^{n_2}) \neq F(a^{n_1}b^{n_2+1}) = F(b)$. Partition S into two sets of nodes, S_1 and S_2, such that $|S_1| = n_1$ and $|S_2| = n_2$. We next construct two input assignments, I and I', as follows:

$$I_v = \begin{cases} a, & v \in S_1 \\ b, & v \in S_2 \\ b, & v \notin S \end{cases} \qquad I'_v = \begin{cases} a, & v \in S_1 \\ b, & v \in S_2 \\ a, & v \notin S \end{cases}$$

Note that $I_S = I'_S$ and $F(I_S) = F(I'_S) = F(a)$.

We first claim that for every $v \in V$, $r \geq d$ and $T \subseteq V$ such that $\Gamma_{\alpha(r)}(v) \subseteq T \subseteq \Gamma_r(v)$, it holds that $F(I_T) = F(b)$. Clearly, the number of a values in I_T is at most n_1. From the choice of S and the fact that S does not cover the entire graph, it holds that $|T| \geq |S| + 1$. As a consequence, the number of b values in I_T is at least $n_2 + 1$. Since $F(a^{n_1}b^{n_2+1}) = F(b)$, it follows from convexity that $F(I_T) = F(b)$.

As a result, we obtain that $F(I) = F(b)$ and $VR_\alpha(I) \leq d$ by definition. Similarly, $F(I') = F(a)$ and $VR_\alpha(I') \leq d$. Since all nodes in S have *exactly* the same input in both assignments, it is obvious that v_0 cannot distinguish between I and I' before $\lfloor \alpha(d) \rfloor$ steps take place. □

THEOREM 2 (LOWER BOUND). *Let $P_{G,F}$ be an aggregation problem. For every $d \geq 0$, every slack function α and every deterministic algorithm A that solves P, there exists an input I with $VR_\alpha(I) \leq d$ for which $OS_A(I) \geq min\{\lfloor \alpha(d) \rfloor, Rad(G)\}$. The same holds for quiescence.*

PROOF. If $\alpha(d) < 1$ the claim holds trivially, so assume that $\alpha(d) \geq 1$. If $\alpha(d) \leq Rad(G)$, let I and I' be the inputs, and $v_0 \in V$ the node from the construction in Lemma 2. We initially focus on output stabilization. Consider the input I. If $OS_A(I) \geq \lfloor \alpha(d) \rfloor$ we are done. Otherwise, there exists a time $t' < \lfloor \alpha(d) \rfloor$ such that for every $t \geq t'$, it holds that $O_{v_0} = F(I)$. Since v_0 cannot distinguish between I and I' before $\lfloor \alpha(d) \rfloor$, for every t such that $t' \leq t < \lfloor \alpha(d) \rfloor$, $O_{v_0} = F(I)$ for I' as well. Therefore, in I', O_{v_0} must change from $F(I)$ to $F(I')$ at $\lfloor \alpha(d) \rfloor$ or later because A correctly solves P.

For quiescence, assume by contradiction that both $Q_A(I)$ and $Q_A(I')$ are less than $\lfloor \alpha(r) \rfloor$. Therefore, v_0 cannot distinguish between I and I' at all times. As a consequence, O_v must be identical upon reaching output-stabilization (as A solves $P_{G,F}$, it must do so in finite time) for both inputs, contradicting the correctness of A.

If $\alpha(d) > Rad(G)$, consider an input I that satisfies the lemma with respect to output stabilization (quiescence) for a value of d' such that $\alpha(d') = Rad(G)$. Since α is a monotone non-decreasing function, the fact that $\alpha(d') < \alpha(d)$ implies that $d' < d$. Therefore, I also satisfies the lemma for d because $VR_\alpha(I) < d$, and A requires at least $Rad(G) = min\{\lfloor \alpha(d) \rfloor, Rad(G)\}$ steps to reach output stabilization (quiescence). □

Clearly, prior knowledge of VR would enable every node to obtain the correct result by collecting only the input values in its VR-neighborhood. Unfortunately, such knowledge is not available and is impossible to obtain without communication. Nevertheless, a simple *Full Information* protocol (FI), in which every node broadcasts its input to all other nodes and outputs the result of the aggregation function over the inputs it heard so far, achieves the output-stabilization lower bound for every d such that $\alpha(d) \leq Rad(G)$, thereby showing that the lower bound is tight. (For a formal description of FI and a proof of its output-stabilization time, see Appendix A.) However, FI does not achieve the quiescence bound: its quiescence time is $Diam(G)$ for every input I, irrespective of $OS_{FI}(I)$. FI also sends $O(|E| \cdot Diam(G))$ messages, and has a high memory consumption of $O(|V|)$ per node.

4. EFFICIENT LOCAL AGGREGATION

Can we do better than FI in terms of memory consumption, communication cost, and quiescence time while still maintaining nearly optimal output stabilization? We now present I-LEAG, an efficient algorithm for solving any aggregation problem, which achieves (c, α)-local output stabilization and quiescence times. In addition, I-LEAG is communication efficient, achieving a local message load in many cases, and has a low memory usage.

We begin by describing a locality-preserving partition hierarchy required by I-LEAG (in Section 4.1), followed by a formal statement of the algorithm itself (in Section 4.2). Next, we prove I-LEAG's correctness (in Section 4.3) and analyze its output-stabilization and quiescence times, message load, and memory usage (in Section 4.4).

4.1 Local Partition Hierarchy (LPH)

Let $G = G(V, E)$ be a graph, and let $\Lambda_\theta = \lceil \log_\theta(Diam(G)) \rceil$. I-LEAG can solve any aggregation function F on G, given a certain pre-construction of partitions, pivot nodes, and some additional information on G. This information can be pre-computed once for G and subsequently used to solve any instance of $P_{G,F}$. Moreover, the pre-construction does not depend on F, so it can be used for other aggregation functions (on the same graph). The pre-construction must form a partition hierarchy:

Definition 6. (Partition Hierarchy) An m-level *partition hierarchy* is a triplet $\langle \{\mathcal{S}_i\}, \{\mathcal{P}_i\}, \{\mathcal{T}_i\} \rangle, 0 \le i \le m$, where:

- $\{\mathcal{S}_i\}$ is a set of partitions, in which for every cluster $S' \in \mathcal{S}_{i-1}$ there exists a cluster $S \in \mathcal{S}_i$ such that $S' \subseteq S$. The topmost level, $\mathcal{S}_m$, contains a single cluster equal to V.
- $\{\mathcal{P}_i\}$ is a set of pivot sets. $\mathcal{P}_i$ includes a single pivot for every cluster $S \in \mathcal{S}_i$. $S_i(p) \in \mathcal{S}_i$ denotes the cluster S for which $p \in S$.
- $\{\mathcal{T}_i\}$ is a set of forests. For every $p \in \mathcal{P}_i$, $\mathcal{T}_i$ contains a directed tree $T_i(p)$ whose root is p and whose leaves are either the nodes $\{p' \in \mathcal{P}_{i-1} \mid S_{i-1}(p') \subseteq S_i(p)\}$ or the nodes in $S_0(p)$ if $i = 0$.

Definition 7. (Logical Tree) For every $i > 0$ and every $p \in \mathcal{P}_i$, denote by $\widetilde{T}_i(p)$ the *logical tree* formed recursively by concatenating $T_i(p)$ and $\widetilde{T}_{i-1}(p')$ at every leaf p' of $T_i(p)$, where $\forall p' \in \mathcal{P}_0$: $\widetilde{T}_0(p') \triangleq T_0(p')$.

Note that a logical tree $\widetilde{T}_i(p)$ spans all nodes in $S_i(p)$, and its underlying physical graph may contain cycles due to multiple roles of the same node in different levels of the hierarchy.

In addition, the partition hierarchy used by I-LEAG must be (θ, α)-local for some θ, α:

Definition 8. ((θ, α)-Local Partition Hierarchy (LPH)) Let $\theta \ge 2$ and let α be a slack function. A Λ_θ-level partition hierarchy $\langle \{\mathcal{S}_i\}, \{\mathcal{P}_i\}, \{\mathcal{T}_i\} \rangle$ is called (θ, α)-local if for every $p \in \mathcal{P}_i$, it holds that $\Gamma_{\alpha(\theta^i)}(p) \subseteq S_i(p) \subseteq \Gamma_{\theta^i}(p)$, and $\widetilde{T}_i(p)$'s height is at most θ^i.

Thus, in contrast to other known partition hierarchies [19], which impose upper bounds on cluster radii, an LPH also enforces lower bounds.

As an example, we present a centralized algorithm, MESH-P, for constructing a $(2, \alpha)$-LPH with minimal slack for mesh neighborhoods, where $\alpha(r) = max\{r-1, r/2\}$ is a 2-bounded slack function. MESH-P is detailed in Algorithm 1. For comparison, the "natural" partition hierarchy for meshes (based on squares) results in a larger slack of $\alpha(r) = r/2$. Given a positive integer L and an arbitrary node $p_0 \in \mathbb{N}^2$ of an infinite mesh G_∞ (node labels are cartesian coordinates), MESH-P constructs recursively an $(L+1)$-level hierarchy, $\langle \{\mathcal{S}_i\}, \{\mathcal{P}_i\}, \{\mathcal{T}_i\} \rangle$ for $i \in [0, L]$, which covers 2^{2L+1} nodes of G_∞ around p_0. More specifically, the graph G covered by this hierarchy is the induced subgraph:

$$G = G_\infty \Big(\Gamma_{2^L-1}(p_0) \ \cup \ \Gamma_{2^L-1}(p_0 + \langle 0, 1 \rangle) \Big),$$

which has a diameter of $2^{L+1} - 1$ and resembles a neighborhood of radius of 2^L on a mesh (some of the border nodes are left out to enable tiling).

Algorithm 1 (MESH-P)

Input: $L \in \mathbb{N}, p_0 \in \mathbb{N}^2$
Output: a $(2, \alpha)$-LPH $\langle \{\mathcal{S}_i\}, \{\mathcal{P}_i\}, \{\mathcal{T}_i\} \rangle, i \in [0, L]$, for $\alpha(r) = max\{r-1, r/2\}$
Variables: partition set $\{\mathcal{S}_i\}$, set of pivot sets $\{\mathcal{P}_i\}$, and forest set $\{\mathcal{T}_i\}$, all initially $\emptyset$

1: mesh-part(L, p_0)
2: **return** $\langle \{\mathcal{S}_i\}, \{\mathcal{P}_i\}, \{\mathcal{T}_i\} \rangle, i \in [0, L]$

Function mesh-part:$(i \in \mathbb{N}, p \in \mathbb{N}^2)$
  $\mathcal{S}_i \leftarrow \mathcal{S}_i \cup \{ \ \Gamma_{2^i-1}(p) \ \cup \ \Gamma_{2^i-1}(p + \langle 0, 1 \rangle) \ \}$
  $\mathcal{P}_i \leftarrow \mathcal{P}_i \cup \{p\}$
  if $i = 0$ **then**
    $T \leftarrow \{ \ \langle p + \langle 0, 1 \rangle, p \rangle \ \}$
  else
    $T \leftarrow \emptyset$
    for all $\Delta \in \{\langle 0, 2^{i-1} \rangle, \langle 0, -2^{i-1} \rangle, \langle 2^{i-1}, 0 \rangle, \langle -2^{i-1}, 0 \rangle\}$ **do**
      mesh-part$(i-1, p + \Delta)$
      $T \leftarrow T \cup \{ \overrightarrow{e} \in \mathbb{N}^2 \times \mathbb{N}^2 \mid \overrightarrow{e}$ is an edge in the direct path from $p + \Delta$ to $p \}$
  $\mathcal{T}_i \leftarrow \mathcal{T}_i \cup \{T\}$

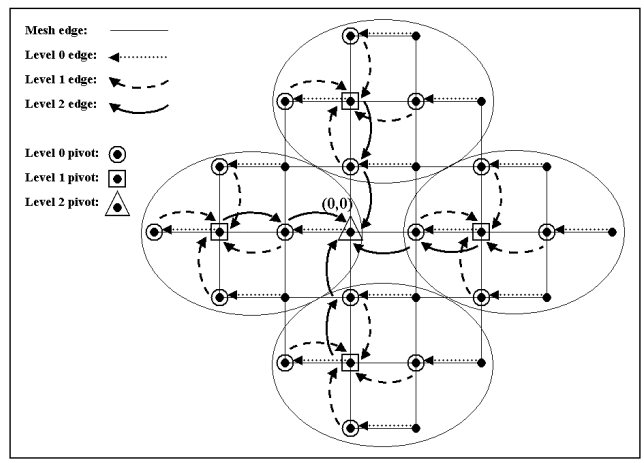

Figure 1: MESH-P partition hierarchy for L=2.

Figure 1 shows the resulting construction for $L = 2$ and $p_0 = \langle 0, 0 \rangle$. The level 2 pivot, which serves as the root of the logical tree that spans the whole graph, is located at $\langle 0, 0 \rangle$. The x and y coordinates of G increase to the right and up, respectively. As can be seen from the figure, the clusters at every level form a partition: level-0 clusters include a pivot and its neighbor to the right; level-1 clusters are marked by ellipses; and the (single) cluster at level 2 comprises the whole graph. In addition, every cluster of level 0 or 1 is fully contained in some cluster of the next (higher) level. Note that $\forall i \in [0, 2], p \in \mathcal{P}_i$: $\Gamma_{2^i-1}(p) \subseteq S_i(p) \subseteq \Gamma_{2^i}(p)$ and the height of $\widetilde{T}_i(p)$ is at most 2^i as required. It is easy to verify that MESH-P generates an LPH for every $L \ge 0$.

Finally, we refer to a hierarchy whose clusters may be weak as a *weak* partition hierarchy. I-LEAG works even with a weak LPH. In Appendix B, we show how to construct a weak $(5, \alpha)$-LPH for *any* graph, for $\alpha(r) = r/5$.

4.2 I-LEAG

I-LEAG is given in Algorithm 2. $O_v[u]$ holds the value of O_u as known to v. $Tree^+$ is the edge set comprising the

Algorithm 2 (I-LEAG) for node $v \in V$

Parameters: $F{:}\mathbb{N}^D \to R$, (θ, α)-local hierarchy: $\langle\{\mathcal{S}_i\}, \{\mathcal{P}_i\}, \{\mathcal{T}_i\}\rangle, 0 \le i \le \Lambda_\theta$ of $G(V, E)$
Input: $I_v \in D$
Output: $O_v \in R$, initially $F(I_v)$
Variables: $\forall u \in \widehat{\Gamma}(v)$: $O_v[u] \in R \cup \{\perp\}$, initially $\perp$; $i \in \mathbb{N}$
Definitions: $Tree^+ \triangleq \bigcup_{i,p\in\mathcal{P}_i} T_i(p)$, ignoring edge directions (i.e., $Tree^+ \subseteq E$)

Synchronous phases:

1: **for** phase $i = 0$ to Λ_θ **do**
2: **set timer** to $4 \cdot \theta^i + 1$
3: **for every** $p \in \mathcal{P}_i$ s.t. $v \in T_i(p)$ **do**
4: **if** $\exists u \in \widehat{\Gamma}(v)$ s.t. u is v's parent in $T_i(p) \wedge O_v[u] \neq O_v$ **then** send $\langle p, \mathsf{conflict}\rangle$ to u
5: **wait** until timer expires

Message handlers:

upon receiving first $\langle p, \mathsf{conflict}\rangle$ message in phase i:
if $v = p$ **then**
$val \leftarrow \mathsf{converge\text{-}cast}(F, \widetilde{T}_i(p))$ /* see Algorithm 3 */
$\mathsf{multicast}$ $\langle p, val\rangle$ to $S_i(p) \cup \{w \in \widehat{\Gamma}(S_i(p)) \mid \exists u \in S_i(p)\colon (u, w) \in Tree^+\}$
else
forward $\langle p, \mathsf{conflict}\rangle$ to v's parent in $T_i(p)$

upon receiving $\langle p, val\rangle$ in phase i:
if $v \in S_i(p)$ **then** $O_v \leftarrow val$
$\forall u \in \widehat{\Gamma}(v)$: **if** $u \in S_i(p)$ **then** $O_v[u] \leftarrow val$

(underlying, undirected) edges of the graph induced by the logical tree. I-LEAG executes Λ_θ sequential phases. Each phase i corresponds to partition $\mathcal{S}_i$ in the hierarchy and consists of three sequential operations, which are executed concurrently in every cluster $S_i(p) \in \mathcal{S}_i$, where $p \in \mathcal{P}_i$:

- Lines 3-4: Check for conflicts among the clusters of $\mathcal{S}_{i-1}$ that constitute $S_i(p)$ by comparing outputs of neighboring nodes along $T_i(p)$[2]. We prove by induction that upon commencing phase i, all nodes within a cluster of level $i-1$ have the same result. Thus, the absence of conflicts indicates that the aggregate value of every $S_{i-1} \subseteq S_i(p)$ is the same, and convexity ensures that this value equals the global result. Detected conflicts are reported to p over the edges of $T_i(p)$. Clearly, nodes in the tree only need forward to p the first notification they receive.
- $\langle\mathsf{conflict}\rangle$ message handler: If a conflict is detected, calculate $F(I_{S_i(p)})$ using a simple converge-cast procedure based on the logical tree of p, $\widetilde{T}_i(p)$. (See detailed description below.) The result is multicast to $S_i(p)$ over the same tree. In addition, every node in $S_i(p)$ forwards the result to its neighbors in $Tree^+$ (excluding nodes in $S_i(p)$).
- $\langle p, val\rangle$ message handler: Finally, if $F(I_{S_i(p)})$ was explicitly calculated, every $v \in S_i(p)$ updates O_v. In addition, every neighbor u of v such that $u \in S_i(p)$ or $(u, v) \in Tree^+$ updates $O_u[v]$ accordingly. This information exchange enables conflict detection in subsequent phases without communication.

[2]If $T_i(p)$ also connects nodes outside $S_i(p)$, conflicts with them will be reported as well.

Algorithm 3 (Converge-cast) given $F \triangleq \langle\widehat{R}, F_I, F_{agg}, F_O\rangle$

Function $\mathsf{converge\text{-}cast}{:}(F, Tree) \to R$
return $F_O(v.\mathsf{converge\text{-}cast\text{-}internal}(F, Tree))$

Function $\mathsf{converge\text{-}cast\text{-}internal}{:}(F, Tree) \to \widehat{R}$
if v is a leaf in $Tree$ **then return** $F_I(I_v)$
let $\{u_1, ..., u_{K_v}\}$ be v's children in $Tree$
for $k = 1, ..., K_v$ **parallel do**
$tmp[k] \leftarrow u_k.\mathsf{converge\text{-}cast\text{-}internal}(F, Tree)$
return $F_{agg}(tmp[1], ..., tmp[K])$

The timer in line 2 and the **wait** instruction in line 5 ensure that the next phase is not started before all messages of the current phase subside (see Lemma 3). If there are no conflicts starting from a certain phase i, then I-LEAG reaches output stabilization at phase i. Moreover, it reaches quiescence at the same time because subsequent conflict checks are done without communication.

The converge-cast procedure assumes that the aggregation function $F{:}\mathbb{N}^D \to R$ can be represented as a tuple $\langle\widehat{R}, F_I, F_{agg}, F_O\rangle$, where: $\widehat{R}$ is some internal representation, and $F_I{:}D \to \widehat{R}$, $F_{agg}{:}\widehat{R}^n \to \widehat{R}$ and $F_O{:}\widehat{R} \to R$ are functions such that for every set of nodes $V = \{v_1, ..., v_n\}$ and an input I:

$$F(I_V) = F_O\Big(F_{agg}\big(F_I(I_{v_1}), ..., F_I(I_{v_n})\big)\Big).$$

For example, a simple majority vote can be represented as follows: $D = R = \{0, 1\}$; $\widehat{R}$ is an ordered pair $\mathbb{N}\times\mathbb{N}$; $F_I(bit)$ returns $\{1, 0\}$ if bit is 1 and $\{0, 1\}$ otherwise; F_{agg} returns the sum of its inputs; and given $\{x, y\} \in \widehat{R}$, F_O returns 1 if $x \ge y$ and 0 otherwise. When invoked at a pivot p, the converge-cast procedure computes F recursively over the inputs of $S_i(p)$ based on the logical tree of p, as described in Algorithm 3 using remote procedure call (RPC) semantics.

4.3 Correctness

THEOREM 3. *I-LEAG correctly solves every aggregation problem $P_{G,F}$, given a (θ, α)-LPH of G.*

The proof follows immediately from the following lemmas:

LEMMA 3. *If a conflict is detected at the beginning of some phase i (line 4), then all messages sent during the phase reach their destination by the end of the phase (line 5). Otherwise, no messages are sent during phase i.*

PROOF. For every $p \in \mathcal{P}_i$, the height of $\widetilde{T}_i(p)$ is at most θ^i. Hence, it takes at most θ^i time for a conflict message to reach its intended pivot, $2\theta^i$ time to conduct a converge-cast [19], and $\theta^i + 1$ time for a multicast to span a cluster in $\mathcal{S}_i$ and its direct neighbors. Since multicasts are initiated at pivot nodes upon completion of converge-cast operations, and these are triggered in response to conflict messages that are sent simultaneously at the beginning of a phase, it takes at most $4\theta^i+1$ time for all messages to subside. Therefore, if a conflict was detected at the beginning of a phase then the **wait** statement in line 5 ensures that there are no messages in flight when the phase ends. Otherwise, no conflict messages would be sent, and thus no converge-cast or multicast operations would be initiated either. □

LEMMA 4. *At the end of every phase i, for every node $v \in V$ and every neighbor u of v such that $(u,v) \in Tree^+$: $O_u[v] = O_v$.*

PROOF. By induction on i. Let $p \in \mathcal{P}_i$ such that $v \in S_i(p)$. If $val = F(I_{S_i(p)})$ is explicitly calculated in phase i, lemma 3 ensures that both u and v receive a message containing a $\langle p, val\rangle$ tuple before the end of the phase. (u receives it either as a member of $S_i(p)$ itself or as a neighbor of v because $(u,v) \in Tree^+$). In this case, v sets $O_v = val$, u sets $O_u[v] = val$, and the lemma holds. Since F is calculated explicitly in all clusters at phase 0, this observation also establishes the basis of the induction. Otherwise, both O_v and $O_u[v]$ are left unchanged and the lemma follows from our assumption on $i-1$. □

LEMMA 5. *At the end of every phase i, for every $p \in \mathcal{P}_i$ and every node $v \in V$: $O_v = F(I_{S_i(p)})$.*

PROOF. By induction on i. We distinguish between two cases. Let $p \in \mathcal{P}_i$. If a conflict was detected in line 4 for p, then $F(I_{S_i(p)})$ is calculated explicitly and assigned to the output register of every $v \in S_i(p)$ by the end of the phase (Lemma 3). Since all clusters experience a conflict in phase 0, this observation also establishes the basis of the induction.

Otherwise, the connectivity of $T_i(p)$ and Lemma 4 ensure that for every $u, v \in T_i(p)$: $O_u = O_v$. According to our assumption on phase $i-1$, for every $S \in \mathcal{S}_{i-1}$ and every $v \in S$ it holds that $O_v = F(I_S)$. We thus also have that for every $S', S'' \in \mathcal{S}_{i-1}$ such that $S', S'' \subseteq S_i(p)$: $F(I_{S'}) = F(I_{S''})$ because $T_i(p)$ contains at least one representative from every such cluster. Therefore, the lemma holds by convexity. □

4.4 Complexity

We begin by analyzing I-LEAG's output stabilization and quiescence times. Given an input I, denote I-LEAG's *veracity phase*, i.e., the last phase in which I-LEAG detects a conflict, by $VP(I)$.

LEMMA 6. *Let $P_{G,F}$ be an aggregation problem. Given a (θ,α)-LPH of G, it holds for every non-trivial input I that: $VP(I) < \log_\theta(VR_\alpha(I)) + 1$.*

PROOF. Let I be a non-trivial input with $VR_\alpha(I) = d$. Denote $VP(I)$ by n and assume in contradiction that $n \geq \log_\theta d + 1$. Lemma 5 ensures that when conflict detection is performed at the beginning of phase n, every node holds the aggregate value of its cluster at level $n-1$. In addition, every two neighboring nodes u, v in $Tree^+$ hold the aggregate value of each other according to Lemma 4.

Since conflicts are only detected between nodes that are neighbors in $Tree^+$, for any such pair of nodes u, v that detect a conflict during phase n, it follows that u and v belong to different clusters at level $n-1$ with different aggregate values. Thus, there exists $p \in \mathcal{P}_{n-1}$ such that $F(I_{S_{n-1}(p)}) \neq F(I)$. Since $\Gamma_{\alpha(\theta^{n-1})}(p) \subseteq S_{n-1}(p) \subseteq \Gamma_{\theta^{n-1}}(p)$, we obtain that $VR_\alpha(I) > \theta^{n-1} \geq \theta^{\log_\theta d+1-1} = d$, a contradiction. □

THEOREM 4. *Let $P_{G,F}$ be an aggregation problem. Given a (θ,α)-LPH of G, for every non-trivial input I, I-LEAG's output stabilization and quiescence times are at most:*

$$\Big(\frac{4\theta^2}{\theta-1}\Big) \cdot VR_\alpha(I) + \log_\theta(VR_\alpha(I)) + 2.$$

PROOF. Let I be a non-trivial input with $VR_\alpha(I) = d$. According to Lemma 3, I-LEAG is quiescent once no more conflicts are detected. Since every phase i takes $4\theta^i + 1$ time, we obtain from lemma 6 that:

$$Q_{I-LEAG}(I) = \sum_{i=0}^{VP(I)} (4\theta^i + 1) = 4\Big(\frac{\theta^{VP(I)+1} - 1}{\theta - 1}\Big) + VP(I) + 1$$

$$< \Big(\frac{4\theta^{\log_\theta d+2} - 1}{\theta - 1}\Big) + \log_\theta d + 2 < \Big(\frac{4\theta^2}{\theta - 1}\Big) d + \log_\theta d + 2.$$

As nodes change their outputs only in response to multicasts, $OS_{I-LEAG}(I) \leq Q_{I-LEAG}(I)$. □

COROLLARY 1. *Let $\mathcal{G}$ be a family of graphs. Given a (θ,α)-LPH for every $G_i \in \mathcal{G}$, I-LEAG is a $(\frac{5\theta^2}{\theta-1}, \alpha)$-OS/Q local algorithm for computing* any *aggregation function F on $\mathcal{G}$.*

PROOF. For every $d \geq 1$ and $\theta \geq 2$, it holds that

$$\Big(\frac{4\theta^2}{\theta-1}\Big) d + \log_\theta d + 2 < \Big(\frac{5\theta^2}{\theta-1}\Big) d.$$

Therefore, the proof is immediate from Theorem 4. □

For any aggregation problem $P_{G,F}$, the number of bits required to represent a single value from the input domain ($\log|D|$), output range ($\log|R|$) or an internal representation ($\log|\widehat{R}|$) obviously depends on the aggregation function F, and in certain cases also on the graph size. However, F is a parameter rather than a property of the algorithm. For brevity, we therefore use the so called "word model" for evaluating both communication (in the form of the average per-node message load ML) and memory usage, and assume that these values practically fit in $O(1)$ memory words. We also make the same assumption for pivot identifiers. Consequently, all of I-LEAG's messages have a size of $O(1)$ memory words. (Results in the "bit model" can be obtained by multiplying by $max\{\log|D|, \log|R|, \log|\widehat{R}|, \log|V|\}$.) Due to lack of space, the proofs are given in [3].

THEOREM 5. *Let $P_{G,F}$ be an aggregation problem. Given a (θ,α)-LPH for G, for every input I, $ML_{I-LEAG}(I) = O(\log^2(VR_\alpha(I)))$ if $Tree^+$ is a tree in G, and $ML_{I-LEAG}(I) = O(\log(VR_\alpha(I)) \cdot \Lambda_\theta)$ otherwise.*

Note that when the (undirected) edge set induced by the LPH's logical tree, $Tree^+$, forms a tree in G, I-LEAG achieves a *local* message load, as the number of messages that a node sends (on average) depends only on VR. Many simple partitions can be constructed with this property, especially for regular graphs (e.g., our mesh construction in Section 4.1).

THEOREM 6. *Let $P_{G,F}$ be an aggregation problem. Given a (θ,α)-LPH for G, I-LEAG requires $O(\Lambda_\theta|V|)$ total memory, and $O(Deg(v))$ memory per node v, where $Deg(v)$ is the number of edges incident to v in the logical tree induced by the hierarchy.*

5. VERACITY RADIUS VS. PRACTICAL PERFORMANCE

In this section, we illustrate that VR explains why previously suggested algorithms are efficient in "common" instances by linking VR with these instances. To this end, we

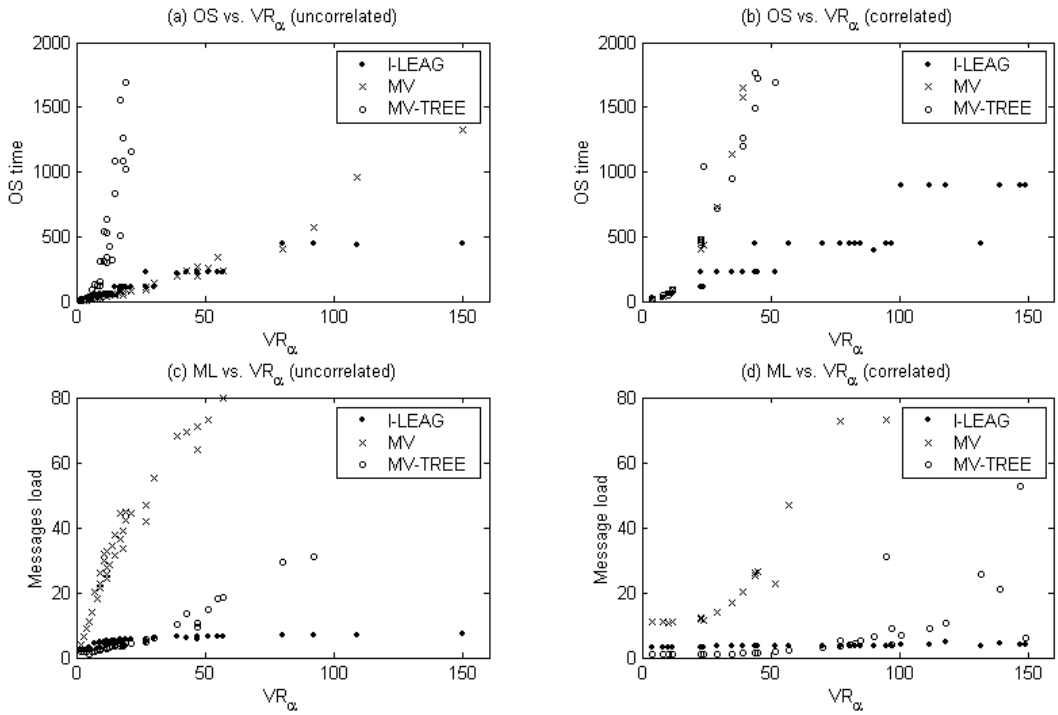

Figure 2: Output stabilization times and message load of I-LEAG, MV, and MV-TREE as a function of VR. All experiments were conducted on a mesh of 128K nodes and a diameter of 511.

conducted an empirical study. We considered binary majority voting (with a threshold of 50%) on a mesh topology, which is typical of sensor networks. We used the construction of Section 4.1 to generate a $(2, \alpha)$-LPH for a graph G of 128K nodes, 255K edges, and a diameter of 511. ($\alpha(r) = max\{r-1, r/2\}$.)

We compared the performance of I-LEAG with that of two previously-suggested algorithms for local majority-voting: MV [4], and MV-TREE [23]. To achieve good coverage, we experimented with two radically different input-distribution models with respect to locality: uncorrelated and correlated. An input assignment was generated as follows. In the *uncorrelated* model, we first chose a probability for voting '1', P_1, and then drew the votes according to P_1 in an i.i.d. fashion (i.e., every node votes '1' with probability P_1 and '0' with probability $1 - P_1$). Thus, votes of neighboring nodes are uncorrelated. In the *correlated* model, we fixed $P_1 = 0.9$, and chose a radius $R \in [0, Rad(G)]$ and a node $v_0 \in V$ uniformly at random. Subsequently, every node $v \in V$ drew its vote according to P_1 if $dist(v, v_0) \leq R$ and according to $1 - P_1$ otherwise. By biasing P_1, this model simulates a scenario in which v_0 is the focal point of an event that arouses every node at distance R away from it, which otherwise would be idle. We ran all algorithms over numerous input assignments that ensured a wide range of VR values.

Output stabilization times are depicted in Fig. 2(a-b). (For all algorithms, quiescence times were practically the same as output stabilization times.) We observe an apparent correlation between output stabilization and VR in all simulations. I-LEAG's operational phases are also evident from the graphs. Clearly, I-LEAG outperforms the other algorithms in both models. Notice that MV, which was designed for i.i.d inputs, achieves good output stabilization times in the uncorrelated model, but performs badly in the correlated model.

Message load is presented in Fig. 2(c-d). In general, for every given VR, all algorithms demonstrate a higher ML in the uncorrelated model, which has considerably more nodes with minority votes than the correlated model. In addition, ML basically increases monotonically with VR. Except for small VRs, in which I-LEAG and MV-TREE have a similar ML, I-LEAG outperforms the other algorithms.

We conclude that VR succeeds in identifying the instances for which the algorithms perform well in practice. Moreover, although the output-stabilization (or quiescence) lower bound presented in Section 3.2 is a *worst case* over over all instances with a certain VR or less, practically none of the algorithms performed better on *any* instance. Intuitively, this can be attributed to the fact that none of the algorithms are biased towards a specific set of instances, which might enable them to perform better than VR for these instances at the expense of others.

6. CONCLUSIONS

This work was motivated by the observed gap between the de facto good performance of efficient local-aggregation algorithms and the theoretical "worst-case" impossibility of local aggregation. We have provided a theoretical instance-based metric, *Veracity Radius (VR)*, which serves as a lower bound on computation time. We have shown the bound to be tight, and provided an efficient algorithm, I-LEAG, whose quiescence and output stabilization times are both within

a constant factor of the lower bound. (The exact factor depends on the LPH pre-construction used by I-LEAG.)

Our notion of local algorithms, namely algorithms whose performance provably depends on an inherent metric of the input instance rather than on the graph size, has implications that go beyond aggregation algorithms. Topology maintenance in wireless mesh networks, load balancing in global computational grids, and cache management in wide-area application services, are all problems that have global instances, but must operate locally whenever possible to achieve good performance. In such systems, VR (or an equivalent metric) can prove invaluable for designing and evaluating efficient algorithms.

7. REFERENCES

[1] Y. Azar, S. Kutten, and B. Patt-Shamir. Distributed error confinement. In *Proc. of the 22nd Annual Symp. on Principles of Distributed Computing*, July 2003.

[2] M. Bawa, H. Garcia-Molina, A. Gionis, and R. Motwani. Estimating aggregates on a peer-to-peer network. Technical report, Stanford University, Database group, 2003. Available from: `http://www-db.stanford.edu/~bawa/publications.html`.

[3] Y. Birk, I. Keidar, L. Liss, A. Schuster, and R. Wolff. Veracity radius - capturing the locality of distributed computations. *CCIT Technical Report 578, EE Department, Technion*, 2006.

[4] Y. Birk, L. Liss, A. Schuster, and R. Wolff. A local algorithm for ad hoc majority voting via charge fusion. In *Proc. of DISC*, October 2004.

[5] J. Considine, F. Li, G. Kollios, and J. Byers. Approximate aggregation techniques for sensor databases. In *Proc. of ICDE*, 2004.

[6] R. Fagin, A. Lotem, and M. Naor. Optimal aggregation algorithms for middleware. *Journal of Computer and System Sciences*, pages 614–656, 2003.

[7] C. Intanagonwiwat, R. Govindan, and D. Estrin. Directed diffusion: A scalable and robust communication paradigm for sensor networks. In *In Proc. of the Sixth Annual Intl. Conf. on Mobile Computing and Networking*, August 2000.

[8] D. Kempe, A. Dobra, and J. Gehrke. Computing aggregate information using gossip. In *Proc. of Fundamentals of Computer Science*, 2003.

[9] F. Kuhn, T. Moscibroda, and R. Wattenhofer. What cannot be computed locally! In *Proc. of the 23rd Symposium on Principles of Distributed Computing (PODC)*, July 2004.

[10] F. Kuhn, T. Moscibroda, and R. Wattenhofer. On the locality of bounded growth. In *Proc. of the 24th Annual Symposium on Principles of Distributed Computing (PODC)*, July 2005.

[11] S. Kutten and B. Patt-Shamir. Time-adaptive self-stabilization. In *In Proc. of the 16th Annual ACM Symposium on Principles of Distributed Computing*, pages 149–158, August 1997.

[12] S. Kutten and D. Peleg. Fault-local distributed mending. In *Proc. of the 14th Annual ACM Symposium on Principles of Distributed Computing*, August 1995.

[13] S. Kutten and D. Peleg. Tight fault-locality. In *Proc. of the 36th IEEE Symposium on Foundations of Computer Science*, October 1995.

[14] J. Li, J. Jannotti, D. D. Couto, D. Karger, and R. Morris. A scalable location service for geographic ad hoc routing. In *Proc. of the 6th ACM Intl. Conf. on Mobile Computing and Networking*, August 2000.

[15] N. Linial. Locality in distributed graph algorithms. *SIAM J. Computing*, 21:193–201, 1992.

[16] S. Madden, M. Franklin, J. Hellerstein, and W. Hong. Tag: a tiny aggregation service for ad-hoc sensor networks. In *Proc. of the 5th Annual Symposium on Operating Systems Design and Implementation (OSDI)*, December 2002.

[17] A. Mainwaring, J. Polastre, R. Szewczyk, and D. Culler. Wireless sensor networks for habitat monitoring. In *Proc. of the ACM Workshop on Sensor Networks and Applications*, 2002.

[18] M. Naor and L. Stockmeyer. What can be computed locally? In *Proc. of the 25th ACM Symposium on Theory of Computing*, 1993.

[19] D. Peleg. *Distributed Computing: A Locality-Sensitive Approach*. SIAM Monographs on Discrete Mathematics and Applications, 2000.

[20] T. C. Project. `http://www.cs.wisc.edu/condor/`.

[21] D. Spielman and S.-H. Teng. Smoothed analysis: Why the simplex algorithm usually takes polynomial time. *Journal of the ACM*, pages 385 – 463, 2004.

[22] R. van Renesse, K. Birman, and W. Vogels. Astrolabe: A robust and scalable technology for distributed system monitoring, management, and data mining. *ACM Transactions on Computer Systems*, 2003.

[23] R. Wolff and A. Schuster. Association rule mining in peer-to-peer systems. In *Proc. of ICDM*, 2003.

[24] J. Zhao, R. Govindan, and D. Estrin. Computing aggregates for monitoring wireless sensor networks. In *Proc. of SNPA*, 2003.

APPENDIX

A. FULL INFORMATION PROTOCOL

In Algorithm 4, we present FI, a full information protocol for computing any aggregation function. Intuitively, FI is optimal in terms of output-stabilization because it actually defines VR:

THEOREM 7. *For every aggregation problem $P_{G,F}$, every slack function α and every $d > 0$ such that $\alpha(d) \leq Rad(G)$, FI has an optimal output stabilization time for instances with $VR \leq d$.*

PROOF. Let I be an arbitrary input with $VR_\alpha(I) = d' \leq d$. It is straightforward from the algorithm that at the end of every time step t, every node v knows the inputs of all nodes in $\Gamma_t(v)$. For every $r \geq d'$ and every $v \in V$, it follows from the definition of VR_α that $F(I_{\Gamma_{\alpha(r)}(v)}) = F(I)$. Since in unweighted graphs $F(I_{\Gamma_{\lfloor\alpha(r)\rfloor}(v)}) = F(I_{\Gamma_{\alpha(r)}(v)})$, FI ensures that $O_v = F(I)$ after time step $t = \lfloor\alpha(d')\rfloor$. Consequently, for every input I such that $VR_\alpha(I) \leq d$:

$$OS_{FI}(I) \leq \lfloor\alpha(VR_\alpha(I))\rfloor \leq \lfloor\alpha(d)\rfloor = LB_d,$$

where LB_d denotes the lower bound. □

B. LPH CONSTRUCTION FOR GENERAL GRAPHS

In Algorithm 5, we present our hierarchal partitioning algorithm, HPART, for constructing a weak LPH on any given graph. HPART is a centralized algorithm that accepts a graph G and a constant $\theta \geq 5$ and returns a (θ, α)-local weak partition hierarchy of G, where $\alpha(r) = r/\theta$ is a θ-bounded slack function. At level 0, every node is defined as its own cluster (line 1). Subsequently, HPART operates in phases, constructing a partition level in each phase. The construction itself is done in two loops:

- Lines 4 - 12: Build a group of clusters that fulfill the requirements for the current level. These clusters do not necessarily cover the whole graph.
- Lines 13 - 18: Expand the clusters until all the entire graph is covered, while maintaining the partition requirements. The resulting clusters may be weak.

During the first loop, $\overline{V}$ holds uncovered nodes, and $\overline{P} \subseteq \overline{V}$ holds only those uncovered nodes that are pivots of the previous level. In every iteration, a new cluster S is added to the current level i based on a completely uncovered θ^{i-1}-neighborhood of some node p (line 5), which serves as the new cluster's pivot. S comprises the nodes of all clusters of level $i-1$ that intersect this neighborhood (lines 6-7), and its spanning tree T is formed by connecting the pivots of these clusters to p using shortest paths while avoiding cycles (lines 8-10). Finally, S, T and p are added to the current level (line 11), and $\overline{P}$ and $\overline{V}$ are updated.

In the second loop, each iteration selects one uncovered cluster (of the previous level) to be covered by one of the newly created clusters of the current level i. Specifically, HPART selects a yet uncovered cluster with pivot $p' \in \mathcal{P}_{i-1}$, and adds its nodes to a cluster $S \in \mathcal{S}_i$ whose pivot $p \in \mathcal{P}_i$ is closet to p' (lines 14-15). Notice that p' cannot be too far from S because otherwise, it would have already been added to a level-i cluster during the first loop. After updating the corresponding tree T (line 16), HPART adjusts level i's cluster and tree sets (line 17), and updates $\overline{P}$. ($\overline{V}$ is not used in this loop.)

It is easy to see that HPART has polynomial execution time. Moreover, devising a corresponding distributed implementation for it is straightforward. HPART's correctness is proved in [3].

Algorithm 4 (FI) for node $v \in V$ given $F{:}\mathbb{N}^D \rightarrow R$

Input: $I_v \in D$
Output: $O_v \in \mathbb{R}$
Variables: $X_v \in 2^{V \times D}$, initially $\{\langle v, I_v \rangle\}$

for step $i = 0$ to $Diam(G)$ **do**
  $\forall X_u$ received from $u \in \widehat{\Gamma}(v)$: $X_v \leftarrow X_v \cup X_u$
  $O_v \leftarrow F(X_v \perp D)$ /* $X_v \perp D$ is the multiset obtained by projecting X_v on D */
  send X_v to $\widehat{\Gamma}(v)$

Algorithm 5 (HPART)

Input: a connected graph $G(V, E)$ and an integer $\theta \geq 5$
Output: a (θ, α)-LPH $\langle \{\mathcal{S}_i\}, \{\mathcal{P}_i\}, \{\mathcal{T}_i\} \rangle, i \in [0, \Lambda_\theta]$, for $\alpha(r) = r/\theta$
Variables: partition set $\{\mathcal{S}_i\}$, set of pivot sets $\{\mathcal{P}_i\}$ and forest set $\{\mathcal{T}_i\}$, all initially $\emptyset$

1: $\mathcal{S}_0 \leftarrow \{\{v\} \mid v \in V\}$, $\mathcal{P}_0 \leftarrow V$, $\mathcal{T}_0 \leftarrow \emptyset$
2: **for** level $i = 1$ to Λ_θ **do**
3: $\overline{P} \leftarrow \mathcal{P}_{i-1}$, $\overline{V} \leftarrow V$

  /* build initial clusters */
4: **while** $\exists v \in \overline{V}$ s.t. $\Gamma_{\theta^{i-1}}(v) \subseteq \overline{V}$ **do**
5: let $p \in \{v \in \overline{V} \mid \Gamma_{\theta^{i-1}}(v) \subseteq \overline{V}\}$
6: $P \leftarrow \{p' \in \overline{P} \mid \mathcal{S}_{i-1}(p') \cap \Gamma_{\theta^{i-1}}(p) \neq \emptyset\}$
7: $S \leftarrow \bigcup_{p' \in P} S_{i-1}(p')$
8: $T \leftarrow \emptyset$
9: **for all** $p' \in P$ **do**
10: $T \leftarrow T \cup \{e \in L \mid L \subseteq E$ is some shortest path from p' to p s.t. $T \cup L$ has no cycles$\}$
11: $\mathcal{S}_i \leftarrow \mathcal{S}_i \cup \{S\}$, $\mathcal{P}_i \leftarrow \mathcal{P}_i \cup \{p\}$, $\mathcal{T}_i \leftarrow \mathcal{T}_i \cup \{T\}$
12: $\overline{P} \leftarrow \overline{P} - P$, $\overline{V} \leftarrow \overline{V} - S$

  /* expand clusters to include all clusters of $\mathcal{S}_{i-1}$ */
13: **while** $\overline{P} \neq \emptyset$ **do**
14: let $p' \in \overline{P}$, $p \in \mathcal{P}_i$ s.t. $dist(p, p') = min_{p \in \mathcal{P}_i}\{dist(p, p')\}$
15: $S \leftarrow S_i(p)$, $S' \leftarrow S \cup S_{i-1}(p')$
16: $T \leftarrow T_i(p)$, $T' \leftarrow T \cup \{e \in L \mid L$ is some shortest path from p' to p s.t. $T \cup L$ has no cycles$\}$
17: $\mathcal{S}_i \leftarrow (\mathcal{S}_i - \{S\}) \cup \{S'\}$, $\mathcal{T}_i \leftarrow (\mathcal{T}_i - \{T\}) \cup \{T'\}$
18: $\overline{P} \leftarrow \overline{P} - \{p\}$

19: **return** $\langle \{\mathcal{S}_i\}, \{\mathcal{P}_i\}, \{\mathcal{T}_i\} \rangle, i \in [0, \Lambda_\theta]$