Want Scalable Computing? Speculate![*]

Idit Keidar
Technion EE Dept.
idish@ee.technion.ac.il

Assaf Schuster
Technion CS Dept.
assaf@cs.technion.ac.il

Abstract

Distributed computing is currently undergoing a paradigm shift, towards large-scale dynamic systems, where thousands of nodes collaboratively solve computational tasks. Examples of such emerging systems include autonomous sensor networks, data-grids, wireless mesh network (WMN) infrastructures, and more. We argue that *speculative computations* will be instrumental to successfully performing meaningful computations in such systems. Moreover, solutions deployed in such platforms will need to be as *local* as possible.

Emerging distributed computing environments

Distributed systems of enormous scale have become a reality in recent years: Peer-to-peer file-sharing systems such as eMule [11] regularly report over a million online nodes. Grid computing systems like Condor [6] report harnessing tens of thousands of machines world-wide, sometimes thousands for the same application [29]. Publish-subscribe infrastructures such as Gryphon [27] and QuickSilver [26] can employ tens of thousands of machines to disseminate varied information to a large population of users with individual preferences, e.g., trading data to stock brokers, or update propagation in large-scale on-line servers. Wireless sensor networks are expected to span tens of thousands of nodes in years to come [8, 12]; testbeds of close to a thousand nodes are already deployed today [9, 7].

Nevertheless, in currently deployed large-scale distributed systems, nodes seldom collaboratively perform distributed computations or attempt to reach common decisions. The functionality of peer-to-peer and publish-subscribe systems is typically restricted to file searching and content distribution. Their topology does not adapt to global considerations, like network partitions or high traffic load at certain hot spots. Moreover, topic clustering, which has a crucial impact on the performance of publish-subscribe systems, is usually conducted offline, based on expected subscription patterns rather than continuously measured ones [30]. Computation grids serve mostly "embarrassingly parallel" computations, where there is virtually no interaction among the nodes. And current-day sensor networks typically only disseminate information (sometimes in aggregate form) to some central location [23, 12].

But this paradigm can be expected to change, as large scale systems performing non-trivial distributed computations will emerge. Simple file sharing applications are expected to evolve into large data-grids, in which thousands of nodes collaborate to provide complex information retrieval and data mining services [10, 28, 13]. As publish-subscribe systems will be used for larger and more diverse applications, they will need to optimize their topology and topic clustering adaptively.

[*]To appear in SIGACT News, Distributed Computing Column, September, 2006.

Similar demands will arise in emerging wireless computing platforms, such as *mobile ad hoc networks (MANETs)*, *wireless mesh networks (WMNs)*, and sensor networks. In MANETs, a collection of mobile devices self-organize in order to allow communication among them in the absence of an infrastructure. Solving graph problems, like minimum vertex cover [15], is important for supporting efficient communication in such networks. In contrast to MANETs, a WMN provides an *infrastructure* for supporting communication among wireless devices, as well as from such devices to stationary nodes on the Internet. Such networks are expected to be deployed in city-wide scenarios, where potentially thousands of wireless routers will need to collectively engage in topology control, while dynamically adapting to varying loads. The WMN infrastructure will also need to dynamically assign Internet gateways to mobile devices that travel through the mesh. Since mobile devices are expected to run real-time rich-media applications, gateway assignment will need to maximize the quality-of-service (QoS). Hence, such assignments ought to take into account local considerations like proximity to the gateway, as well as global ones like load, since both considerations can impact the QoS. Optimizing a cost comprised of distance and load is called *load-distance balancing* [5]. For example, when the load (user population) and the set of servers (or gateways) are evenly divided across the network, the optimal cost is attained when each user is assigned to its nearest server. But when one area of the network is highly congested, a lower cost can be incurred by assigning some users from the loaded area to lightly-loaded remote servers.

Another example arises in sensor networks, which are expected to autonomously engage in complex decision making in a variety of application domains, ranging from detection of over-heating in data-centers, through disaster alerts during earthquakes, to biological habitat monitoring [24]. The nodes in such a network will need to perform a distributed computation in order to cooperatively compute some function of their inputs, such as testing whether the number of sensors reporting a problem exceeds a certain threshold, or whether the average read exceeds another threshold. More generally, the nodes may need to compute some *aggregation function* of their inputs, e.g., majority vote, AND/OR, maximum, minimum, or average.

The need for local computing

Obviously, centralized solutions are not appropriate for such settings, for reasons such as load, communication costs, delays, and fault-tolerance. At the same time, traditional distributed computing solutions, which require global agreement or synchronization before producing an output, are also infeasible. In particular, in dynamic settings, where the protocol needs to be repeatedly invoked, each invocation entails global communication, inducing high latency and load. In fact, the latency for synchronization over a large WAN, as found in peer-to-peer systems and grids, can be so large that by the time synchronization is finally done, the network and data may well have already changed. Frequent changes may also lead to computing based on inconsistent snapshots of the system state. Moreover, synchronizing invocations that are initiated at multiple locations typically relies on a common sequencer (leader) [20], which by itself is difficult and costly to maintain.

Then how can one cope with the challenge of performing non-trivial computations in large-scale distributed systems? *Locality* is the key– there is a need for *local* solutions, whereby nodes make local decisions based on communication (or synchronization) with some proximate nodes, rather than the entire network.

Various different definitions of locality appear in the literature. But in general, a solution is typically said to be *local* if its costs (usually running time and communication) do not depend on the network size. A local solution thus has unlimited scalability. It allows for fast computing, low overhead, and low power consumption.

Speculation is the key to maximum locality

Unfortunately, although locality is clearly desirable, it is not always attainable. Virtually every interesting problem described above has some instances (perhaps pathological ones) that require global communication in order to reach the correct solution, even if most of its instances are amenable to local computation. Consider, for example, load-distance balancing in WMNs [5]. If the user load is evenly distributed across the entire WMN, then the optimal assignment, namely, assigning each user to its nearest gateway, can be computed locally by each router. But if load in one area of the network may be arbitrarily high, then the optimal solution may involve assigning users from this area to arbitrarily distant gateways, which may even involve communication with the entire network [5].

Locality has been considered in the context of distributed graph constructions, e.g., coloring [21, 25] and minimum vertex cover [15, 22], which are useful in MANETs. Such problems were mostly shown impossible to compute locally, (i.e., in constant time), except when limited to simplistic problems, approximations, or restricted graph models [21, 25, 16][1]. Another example arises in computing aggregation functions, such as majority voting, in sensor networks (or other large graphs) [4, 3, 2]. If the vote is a "landslide victory", e.g., when very few or almost all sensors report of a problem, then this fact can be observed in every neighborhood of the network. But in instances where the votes are at a near-tie, one must inevitably communicate with all nodes to discover the correct solution (if indeed an accurate solution is required).

Fortunately, in many real-life applications, practical instances are highly amenable to local computing [4, 16]. Furthermore, in many important areas, perfect precision is not essential. Consider, for instance, a query in a data mining application, where the top two results have approximately the same score. In such scenarios, it is often immaterial which of the two is returned.

A desirable solution, therefore, should be "as local as possible" for each problem instance: In instances amenable to local computations, e.g., evenly distributed load in the WMN example, the system should converge to the correct result promptly. Moreover, it should become quiescent (stop sending messages) upon computing the correct result. In instances where reaching the correct solution requires communication with more distant nodes, the system's running times and message costs will inevitably be longer, but again, it is desirable that they be proportional to the distance that needs to be traversed in each instance.

Note, however, that each node by itself cannot deduce solely based upon local communication whether it is in a problem instance that can be solved locally or not. For example, a WMN node in a lightly-loaded area of the network cannot know whether some distant node will need to offload users to it. Therefore, in order to be "as local as possible", a solution must be *speculative*. That is, it must optimistically output the result that currently appears correct, even if this result may be later over-ruled because of messages that later arrive from distant nodes. In most instances, the computation will complete (and the system will become quiescent) a long time before any individual node will know that the computation is "done".

Fortunately, in the domains where such computations are applicable, it is not important for nodes to know when a computation is "done". This is because such platforms are constantly

[1]These local solutions typically operate in $O(\log^* n)$ time for a network of size n, which for practical purposes, can be seen as constant.

changing; that is, their computations are never "done". For example, inputs (sensor readings) in a sensor network change over time. Hence, the system's output, which is an aggregation function computed over these inputs, must also change. Regardless of whether speculation is employed, the user can never know whether the output is about to change, due to some recent input changes that are not yet reflected. The system can only ensure that once input changes cease, the output will eventually be correct. Using speculation provides the same semantics, but substantially expedites convergence in the majority of problem instances [2, 4]. At the same time, one should speculate responsibly: if the current output was obtained based on communication with 100 nodes, the algorithm shouldn't speculatively produce a different output after communicating with only 10 nodes.

Such dynamic behavior occurs in various large-scale settings: In data mining applications, the set of data items changes over time, as items are added, deleted, or modified. In topology control applications, nodes and communication links fail and get repaired. In WMNs, users join, depart, and move. Hence, speculative algorithms are useful in all of these platforms. Such algorithms can also be composed– for example, speculative majority voting can be used as a building block for a speculative data mining application [14].

To summarize, speculative computing is a promising approach for designing large-scale distributed systems. It is applicable to systems that satisfy the following conditions:

1. Global synchronization is prohibitive. That is, progress cannot be contingent on communication spanning the entire network.

2. Many problem instances are amenable to local solutions.

3. Eventual correctness is acceptable. That is, since the system is constantly changing, there isn't necessarily a meaningful notion of a "correct answer" at every point in time. But when the system stabilizes for "long enough", the output should converge to the correct one in the final configuration.

What does "as local as possible" mean?

The discussion above suggests that some problem instances are easier than others, or rather more amenable to local solutions. For some problems, one can intuitively recognize problem parameters that impact this amenability, for instance, the load discrepancies in load-distance balancing in WMNs, and the vote ratio in majority voting. Note that these problem parameters are *independent* of the system size– the same vote ratio can occur in a graph of any size. Suggested "local" solutions to these problems have indeed empirically exhibited a clear relationship between such parameters and the costs [4, 5]. However, how can this notion be formally defined?

The first formal treatment of "instance-based locality" was introduced by Kutten and Peleg in the context of local fault mending [18]. They linked the cost of a correction algorithm (invoked following faults) to the number of faults in an instance. Similar ideas were subsequently used in various studies of fault-tolerance [19, 17, 1]. In this context, the number of faults is the problem parameter that determines its amenability to local computation.

However, the problem parameter that renders instances locally computable is not always intuitive to pinpoint. Consider, for example, an algorithm computing *any* given aggregation function on a large graph. What is the problem parameter that governs its running time? In recent work with Birk et al. [3], we provide an answer to this question. We define a new formal metric on problem instances, *veracity radius (VR)*, which captures the inherent possibility to compute them locally. 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, then there is apparently no need for a node to communicate with nodes that are farther than r_0 hops away from it, irrespective of the graph size. The VR is then the minimum radius r_0 so that in all neighborhoods of radius $r \geq r_0$, the value of the aggregation function is the same as for the entire graph[2]. For example, if the aggregation function is majority voting, and the VR is 3, then the same value "wins" the vote in every 3-neighborhood in the graph. This value is clearly the majority over the entire graph, and every node can reach the globally correct result by communicating only with its 3-neighborhood. It is important to note that the nodes do not know the instance's VR, and hence cannot know that the outcome observed in the 3-neighborhood is the right one. However, a *speculative* algorithm can output the correct result once it gathers information within a radius of 3, and never change its output value thereafter.

Indeed, VR yields a tight lower bound on output-stabilization time, i.e., the time until all nodes fix their outputs to the value of the aggregation function, as well as an asymptotically tight lower bound on quiescence time [3]. Note that the output stabilization bound is for speculative algorithms: it is only seen by an external observer, whereas a node that runs the algorithm cannot know when it is reached. The above lower bounds are proven for input *classes* rather than individual instances. That is, for every given value r of the VR, no algorithm can achieve output stabilization or quiescence times shorter than r on *all* problem instances with a VR of r. In this context, the VR is the problem parameter that defines amenability to local computation. Empirical evaluation further shows that the performance of a number of efficient aggregation algorithms [31, 4, 14, 3] is effectively explained by the VR metric.

Although the VR metric is originally defined for a single problem instance, it has also been extended to deal with on-going aggregation over dynamically changing inputs [2]. This extension examines the VRs of all instances in a sliding window of input samples. As in the static case, this metric yields a lower bound on quiescence and output stabilization, and an algorithm whose performance is within a constant factor of the lower bound [2].

Discussion and additional research directions

This collection of examples naturally raises the question of whether the notion of "amenability to local computation" can be further generalized. An appealing way to try and generalize the notion of locality is by linking the per-instance performance of an algorithm directly to the performance of the optimal solution for this instance. For example, it would be desirable for an algorithm to perform within a factor of the best possible algorithm on every problem instance.

Unfortunately, it is only possible to prove such results in restricted special cases (e.g., load-distance balancing when restricted to a line instead of a plane [5]). There is no general way to do so, since it is often not possible to design a single algorithm that is optimal for all instances. Consider again the question of aggregation on a graph. For every given (fixed) problem instance I, (that is, assignment I of input values to nodes), it is possible to design an algorithm as follows: each node locally checks whether its input matches its input in I. If yes, it speculatively decides on the value of the aggregation function on I, and does not send any messages. Otherwise, it initiates

[2]For practical purposes, the formal VR metric does not consider exact neighborhoods, but rather allows for some slack in the subgraphs over which the values of the aggregation function are examined [3].

some well-known aggregation algorithm. If a node hears a message from a neighbor, it joins the well-known algorithm. This algorithm reaches quiescence and output stabilization within 0 time on instance I, which is clearly impossible for a single algorithm to achieve on all instances.

In conclusion, it remains a major challenge to find a general notion of "amenability to local computation" that will capture a wide range of distributed computations, including aggregation problems, fault mending, load-distance balancing, topic clustering in publish-subscribe systems, data mining, and so on. While such a general notion is still absent, appropriate metrics– and efficient speculative algorithms– for specific problems should continue to be sought. Of special interest will be on-going algorithms for dynamically changing systems, and algorithms providing *approximate* results rather than perfectly accurate ones.

Acknowledgments

We are thankful to Danny Dolev and Sergio Rajsbaum for helpful comments.

References

[1] Y. Azar, S. Kutten, and B. Patt-Shamir. Distributed error confinement. In *ACM Symp. on Prin. of Dist. Computing (PODC)*, July 2003.

[2] Y. Birk, I. Keidar, L. Liss, and A. Schuster. Efficient dynamic aggregation. In *Int'l Symp. on DIStributed Computing (DISC)*, Sept. 2006.

[3] Y. Birk, I. Keidar, L. Liss, A. Schuster, and R. Wolff. Veracity radius - capturing the locality of distributed computations. In *ACM Symp. on Prin. of Dist. Computing (PODC)*, July 2006.

[4] Y. Birk, L. Liss, A. Schuster, and R. Wolff. A local algorithm for ad hoc majority voting via charge fusion. In *Int'l Symp. on DIStributed Computing (DISC)*, Oct. 2004.

[5] E. Bortnikov, I. Cidon, and I. Keidar. Load-distance balancing in large networks. Technical Report 587, Technion Department of Electrical Engineering, May 2006.

[6] Condor. High throughput computing. `http://www.cs.wisc.edu/condor/`.

[7] D. Culler. Largest tiny network yet. `http://webs.cs.berkeley.edu/800demo/`.

[8] P. Dutta, M. Grimmer, A. Arora, S. Bibyk, and D. Culler. Design of a wireless sensor network platform for detecting rare, random, and ephemeral events. In *4th Int'l Conf. on Information Processing in Sensor Networks (IPSN'05)*, Apr. 2005.

[9] P. Dutta, J. Hui, J. Jeong, S. Kim, C. Sharp, J. Taneja, G. Tolle, K. Whitehouse, and D. Culler. Trio: Enabling sustainable and scalable outdoor wireless sensor network deployments. In *5th Int'l Conf. on Information Processing in Sensor Networks (IPSN) Special track on Platform Tools and Design Methods for Network Embedded Sensors (SPOTS)*, Apr. 2006.

[10] EGEE. Enabling grids for e-science. `http://public.eu-egee.org/`.

[11] eMule Inc. emule. `http://www.emule-project.net/`.

[12] D. Estrin, R. Govindan, J. Heidemann, and S. Kumar. Next century challenges: Scalable coordination in sensor networks. In *ACM/IEEE Int'l Conf. on Mobile Computing and Networking*, pages 263–270, Aug. 1999.

[13] S. Ghemawat, H. Gobioff, and S.-T. Leung. The Google file system. In *ACM Symposium on Operating Systems Principles (SOSP)*, pages 29–43, Oct. 2003.

[14] D. Krivitski, A. Schuster, and R. Wolff. A local facility location algorithm for sensor networks. In *Int'l Conf. on Distributed Computing in Sensor Systems (DCOSS)*, June 2006.

[15] F. Kuhn, T. Moscibroda, and R. Wattenhofer. What cannot be computed locally! In *ACM Symp. on Prin. of Dist. Computing (PODC)*, July 2004.

[16] F. Kuhn, T. Moscibroda, and R. Wattenhofer. On the locality of bounded growth. In *ACM Symp. on Prin. of Dist. Computing (PODC)*, July 2005.

[17] S. Kutten and B. Patt-Shamir. Time-adaptive self-stabilization. In *ACM Symp. on Prin. of Dist. Computing (PODC)*, pages 149–158, Aug. 1997.

[18] S. Kutten and D. Peleg. Fault-local distributed mending. In *ACM Symp. on Prin. of Dist. Computing (PODC)*, Aug. 1995.

[19] S. Kutten and D. Peleg. Tight fault-locality. In *IEEE Symp. on Foundations of Computer Science (FOCS)*, Oct. 1995.

[20] L. Lamport. The part-time parliament. *ACM Trans. Comput. Syst.*, 16(2):133–169, May 1998.

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

[22] M. Luby. A simple parallel algorithm for the maximal independent set problem. *SIAM Journal on Computing*, 15(4):1036–1055, Nov. 1986.

[23] S. Madden, M. Franklin, J. Hellerstein, and W. Hong. Tag: a tiny aggregation service for ad-hoc sensor networks. In *5th Symp. on Operating Systems Design and Implementation (OSDI)*, Dec. 2002.

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

[25] M. Naor and L. Stockmeyer. What can be computed locally? *ACM Symp. on Theory of Computing (STOC)*, pages 184–193, 1993.

[26] K. Ostrowski and K. Birman. Extensible web services architecture for notification in large-scale systems. In *IEEE International Conference on Web Services (ICWS)*, 2006. To appear.

[27] R. Strom, G. Banavar, T. Chandra, M. Kaplan, K. Miller, and B. Mukherjee. Gryphon: An information flow based approach to message brokering. In *Fast Abstract in Int'l Symposium on Software Reliability Engineering*, Nov. 1998.

[28] TeraGrid. Teragrid project. `http://www.teragrid.org/`.

[29] D. Thain, T. Tannenbaum, and M. Livny. Distributed computing in practice: The Condor experience. *Concurrency and Computation: Practice and Experience*, 17(2–4):323–356, Feb.–Apr. 2005.

[30] Y. Tock, N. Naaman, A. Harpaz, and G. Gershinsky. Hierarchical clustering of message flows in a multicast data dissemination system. In *Parallel and Dist. Comp. and Systems (PDCS)*, Nov. 2005.

[31] R. Wolff and A. Schuster. Association rule mining in peer-to-peer systems. In *IEEE Int'l Conf. on Data Mining (ICDM)*, Nov. 2003.