

Challenges in Scheduling Aggregation in CyberPhysical Information Processing Systems

James Horey
Computational Sciences and Engineering
Oak Ridge National Laboratory
Oak Ridge, TN, USA
email: horeyjl@ornl.gov

Abstract—Data aggregation is an important element in information processing systems, including MapReduce clusters and cyberphysical networks. Unlike simple sensor networks, all the data in information processing systems must be eventually aggregated. Our goal is to lower overall latency in these systems by intelligently scheduling aggregation on intermediate routing nodes.

In order to understand the potential challenges associated with constructing a distributed scheduler that minimizes latency, we developed a simple model of wireless information processing systems and simulation of our model. Unlike previous models, our model explicitly takes into account link latency and computation time. Our model also considers heterogeneous computing capabilities. We tested the latency while randomly assigning aggregation computation to nodes in the network. Preliminary results indicate that in cases where the computation time is greater than transmission time, in-network aggregation can have a large effect (reducing latency by 50% or more). However, naive scheduling can have a detrimental effect. Specifically, when the root node (a.k.a the basestation) is faster than the other nodes, the latency can increase with increased coverage, and these effects vary with the number of nodes present.

Keywords-sensor networks; scheduling; information processing; aggregation

I. INTRODUCTION

In this paper, we explore the challenges of scheduling data aggregation (i.e. reduce operations) in information processing systems, with the goal of minimizing the resultant latency. We define an information processing system to be one in which the primary goal is to perform some sort of computation over the data (as opposed to simply relaying the raw data). Examples include clusters that operate over big data (MapReduce [1], LINQ [2]) and public resource computing [3] (SETI@home [4], Einstein@home [5], Folding@home [6]) that operate over volunteered computing resources. More recent examples include cyberphysical networks that are designed to process and analyze sensor data. We assume that an important goal for these systems is to lower overall latency. For example, a surveillance network designed to track vehicles via ground sensors must perform the computation before the vehicle is out of range, and in a traffic monitoring network driving conditions must be

calculated in a timely manner.

In information processing systems, computation can be expressed as a type of reduce operation (along with some local operations). Typically reduce operations are more complex than simple aggregation functions [7], but can still be parallelized across a set of machines. Each partial computation outputs a partial aggregate, which can be merged at another node. Although some programming interfaces, such as MapReduce, do not parallelize these functions, there is no inherent reason why the interface couldn't be extended to support parallel partial aggregation.

In this paper we focus on cyberphysical networks that employ a wireless, multi-hop routing scheme. Even when nodes have access to a long-range communication capability (e.g. cellular), users may prefer to disable such communications (due to monetary costs, poor signal, battery power, etc.). Tree-like routing structures are quite common for these networks and easily enable in-network data aggregation [8], [9]. Given such systems, the key question is: where should reduce operations be scheduled so that the overall latency is minimized? Previous works on this question [10], [11], have only considered homogeneous environments (where the relative speed of the basestation is not considered) and do not explicitly model computation and transmission time. In addition, these works also consider actively selecting the routing path; here we assume that the routing path is determined by the operating system and is not directly controlled by the aggregation scheduler.

Note that our scenario addresses a slightly different problem than the one posed in typical real-time sensor networks. We assume that the reduce operations eventually take place *somewhere* in the network; simply relaying the raw data as fast as possible is not an option. Work by Abdelzaher et al. [12] assume that the goal is to saturate the network without introducing additional delays and that aggregation is an important, but optional tool. The question we address assumes that a routing mechanism is already in place [13], and that the challenge lies in *scheduling* the reduce operations given a routing tree. In order to investigate the possible tradeoffs associated with different scheduling algorithms, we've developed a simple model and simulation

of a cyberphysical information processing system.

II. MODEL

We employ a high-level model of the information processing network. Our model is differentiated from previous models by explicitly modeling computation, node computation time, and node transmission time. Our model is similar to a finite-state model in which nodes generate data (or equivalently access data from a local storage device), route this data up towards a final destination (i.e. the basestation), and perform reduce operations over the data. These actions occur in discrete timesteps. Elements of our model include *nodes* and *computation*. Nodes are connected via some routing topology (typically some tree structure), and can be assigned computation.

A single computational job is modeled by the time it takes to perform work over a unit of data (c_d), the units of data the computation consumes (input), the units of data the computation outputs, and the minimum units of data it takes to trigger the work. Since we are modeling reduce operations, the computation will output fewer units than the input. In addition, the time it takes for a single computational job to process the input scales linearly with the amount of input. In our model, data is modeled as an integer that can be interpreted as the size of the data.

Nodes are modeled by the computational time to process a data element (c_n), packet transmission size (p), and transmission time (t_n). For c_n and t_n , lower values indicate a faster node or link. Nodes also contain a local data buffer. Since data is modeled by the size of the data, the buffer is also modeled by a value indicating the number of bytes it contains (b). Both computation speed and transmission model the number of timesteps it takes for the node to perform computation or transmit data. If a node is assigned computation, the time it takes to execute the computation is a product of c_n , c_d and the amount of data in the buffer. Transmission speed is also an integer value indicating the number of timesteps it takes to deliver a single packet to the routing parent. Packet size is an integer value indicating the number of data units the node can transfer during a single transmission period.

At each timestep, nodes can be in one of three states: *computing*, *transmitting*, or *idle*. Nodes can receive data while in any of these states. Initially, each node generates (or accesses) some amount of data that must be routed and processed. If a node is in an *idle* state, it first attempts to transition into the computing state. If this fails, the node then attempts to transition into the transmitting state. Finally, if both transitions fail (perhaps there is no data in the buffer), the node remains in the idle state.

A node enters the *computation* stage if it has been assigned computation and has sufficient input. Once in the computation stage, the node will wait an appropriate number of timesteps for the computation to complete ($c_n * c_d * b$).

After the computation is complete, the output is added to the data buffer. If there are no other pending computational jobs, the node will then enter the transmitting state and transfer the output. The root of the routing tree is special in that it is always assigned a reduce operation (so that at least one node in the network will perform a reduce).

Upon entering the *transmitting* state, the node waits the appropriate number of timesteps for the transmission to complete ($t_n * b$). At the end of the transmission, the data present at the start of the transmitting state is transferred to the routing parent. During any of these stages, additional data may accumulate in the buffer. At the completion of the transmitting stage, the node will re-enter the *idle* stage, and thus restart the computation cycle over the new data.

By changing the relative values of c_d , c_n and t_n , our model can emulate the different properties of various networks and applications. For example, a MapReduce-like system running on a mobile network can increase the value of c_d so that the system becomes bounded by computation time. Likewise a sensor network may set t_n to be higher relative to c_n to emulate a slower network. To model lossy radio networks, different t_n can be assigned to nodes depending on the number of routing siblings. To model heterogeneous networks, c_n and t_n can be assigned different values for certain nodes (such as assigning a very low c_n to the basestation). One current limitation is that these parameter values are set at the beginning of the simulation, and do not change over time. However, we expect to implement and evaluate dynamic settings in the future.

III. EVALUATION

We explore the effects of *randomly* assigning reduce computations for multiple network scenarios. These results can be used to guide future algorithm development and can serve as a null model for other more interesting algorithms. Because our model considers an abstract view of the communication, we opt to use a custom simulator instead of a low-level packet simulation [14]. We simulate two types of nodes: regular *nodes* and a single *root* node. The root and the nodes are arranged in a binary-tree routing topology. We assume that the root node is always situated at the top of the routing tree. For our experiments, we set the root node to be identical to the processing nodes (same c_n and t_n values), and also make the root node faster (lower c_n value for the root). To simplify terminology, we assign c_r to be the root computation time and limit c_n for the other nodes. Also, since our scheduler assigns computation randomly, we define *coverage* to be the proportion of nodes that are assigned reduce computation. For example, a coverage of 0.0 indicates that only the root node has been assigned a reduce, while a coverage of 1.0 indicates all nodes have been assigned a reduce. For our experiments, we simulate up to 1000 nodes and measure the number of timesteps it takes for the root to reduce all the data from the network

as we increase coverage. For all experiments, the reduce computation was set to consume 100 units and output 1 unit (with a minimum of 200 units before executing). Likewise, the packet size was set to a maximum of 100 units. Finally, each experiment is run independently 10 times over a random routing tree and the results are averaged and normalized.

A. Homogeneous

For the first experiment, we evaluate the effects of varying c_n relative to t_n for networks of size 50, 250, and 1000. This models what occurs as the computation time dominates the transmission time. t_n/c_n was set to 0.25 (computation dominates), 0.50, 2, and 3 (transmission dominates). The root computation time (c_r) was set equal to the node computation time (c_n). We find that when the computation time is smaller than transmission time ($c_n < t_n$), it is always beneficial to increase coverage regardless of the actual network size (Figure 1). This is expected since transmitting the data costs more than computing, the nodes should always choose to compute since this reduces both latency and the amount of data transmitted.

For the scenarios in which $c_n > t_n$, we find that increasing coverage still lowers the latency. However, continually increasing coverage does not yield continued benefit. For larger networks (250 and 1000 nodes), the latency begins to increase again after the network has been covered approximately 50–70%. This U-shaped effect becomes more noticeable with the larger networks. Although one would initially expect that a coverage of 1.0 would yield the greatest benefit, performing reduce operations in our model has a small, but visible cost. As the network becomes larger and the tree structure increases in depth, there will be more partial aggregate data. If enough partial aggregate data is collected, this will induce additional reduce operations.

Intuitively, we also find that the larger the network, the greater the relative benefit is for moderate coverages (< 0.50) (Figure 2). For example, increasing coverage to 0.10 lowers relative latency more for the 1000 node network compared to 250 nodes (Figure 1). This is because the same coverage value covers a larger number of nodes while each reduce operation computes over a larger dataset.

We also performed the same experiment with the node computation time being set to 5 times higher than the root computation time (right-hand Figure 1). This models scenarios in which the root (a.k.a. basestation) is a much faster machine than the other nodes. Unlike the scenario in which c_r is equal to c_n , for 50 nodes, increasing coverage (between 0.10 and 0.30) can *increase* the latency. This is true for all values of c_n , where $c_n < t_n$. This is because with very little coverage and for relatively small networks, the benefit of performing in-network reduce operations does not exceed the benefit of transmitting less data. In addition, if these reduce operations occur near the root, the system is

not able to take advantage of the low root computation time. As the network size increases (250 and 1000 nodes), these effects become less pronounced. Finally, we also find that as computation becomes slower relative to transmission, the more benefit there is to increasing coverage. Intuitively, more expensive computation benefits from more parallelization.

B. Heterogeneous

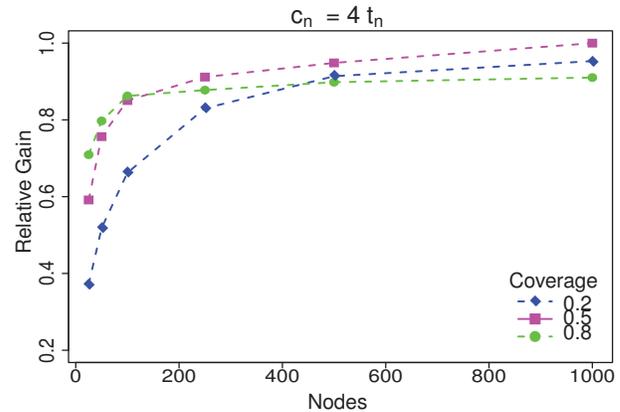


Figure 2. The relative benefit (as measured by latency) as the number of nodes are increased given three coverage values (0.2, 0.5, and 0.8).

For these set of experiments, we compare the effects of making the computation time higher relative to transmission ($c_n > t_n$) across network sizes (25, 50, and 250 nodes). Again, we measure the latency of executing a particular scheduling decision with respect to coverage. For each network size, we set the root computation time to be equal to the node time ($c_r = c_n$), set the root time to be one-fifth smaller ($5c_r = c_n$), and set the root to have zero time (infinitely fast).

For 25 nodes, we observe that latency increases with coverage before dropping back down when c_r is lower than c_n (top-most Figure 3). This is because with such a small network, the cost of transmitting the data is small (few hops), and the benefit of letting the root reduce is great. Unless we can parallelize the task sufficiently, the slow speed of the nodes simply increases the overall latency. This effect becomes more pronounced as c_n becomes larger relative to t_n . When c_n is 20 times greater than t_n , a coverage of 0.0 (i.e. only the root reduces) yields the lowest latency (except when c_r is equal to c_n).

For 50 nodes, we observe approximately the same effects (middle Figure 3). When c_n is only twice greater than t_n , parallelizing the reduce operations yields more benefit compared to the 25 node network. As c_n increases relative to t_n , latency increases with low coverage, although the effect is less pronounced than for the 25 node network. For 100

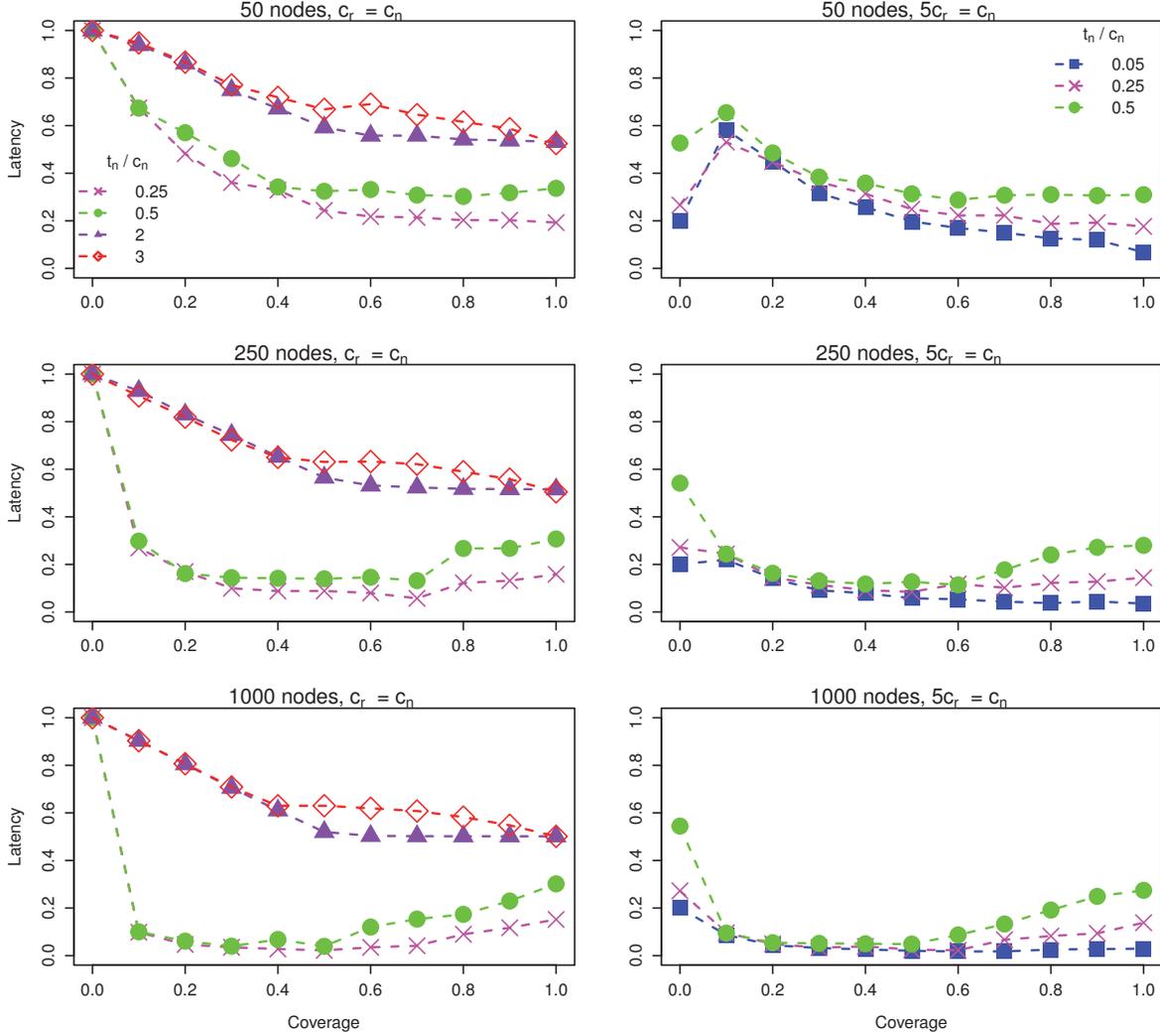


Figure 1. Comparison of latency versus node coverage with different relative values of node computation speed (c_n) and transmission speed (t_n). This experiment was done for 50, 250, and 1000 nodes, with the root computation time (c_r) set equal to the node computation time, and set to $\frac{1}{5}$ the node computation time.

nodes, we find that increasing coverage will immediately lower the latency regardless of c_r (with the one exception of when c_n is 20 times greater than t_n). Although this does not seem intuitive (processing the data in an infinitely fast root should yield a large benefit), the limiting factor becomes the transmission from the nodes to the root.

Because the transmission speed between the root and the nodes is limited, the root will often finish computation before receiving all the data from the network. Consequently, it becomes advantageous to increase coverage (this reduces the amount of data produced by the network). Since the system becomes network-limited, we observe that the latency always converges with increased coverage. This is because as coverage increases, it becomes more likely for a node near the root to perform a reduce operation. This deprives

the root of work (and therefore limits the effects of a lower c_r value). At the extreme case (coverage is 1.0), the root may not perform any significant computation. These values converge sooner in larger networks because a random node in a larger network will likely be the subroot for a larger subset of the network.

As the number of nodes is increased, the relative benefit for a particular coverage increases quickly and then tapers off (Figure 3). For smaller networks, we need proportionally more nodes to perform reduce operations to reduce the latency by the same relative amount. For medium-sized networks (500 nodes), there is very little latency change with respect to coverage. However, for larger networks (1000 nodes), having too much coverage is detrimental since this can cause additional reduce operations.

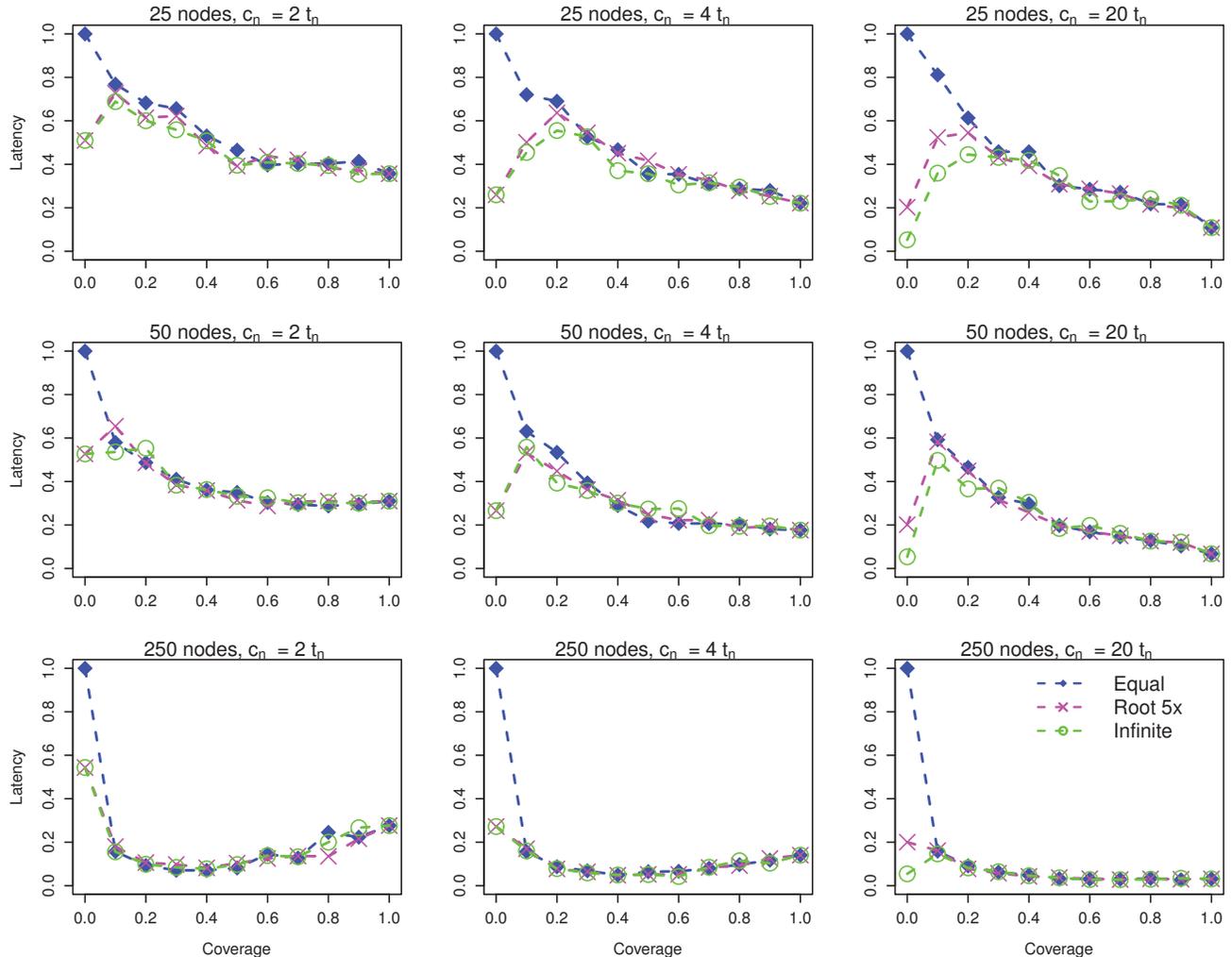


Figure 3. Overall latency versus node coverage when the root computation time (c_r) is set to be equal to c_n , $\frac{1}{5}$ the time, and when c_r is set to 0.0 (infinitely fast). This experiment is evaluated for 25, 50, and 250 nodes as the relative computation time is changed with respect to the transmission time.

IV. DISCUSSION

Although our model and simulation are relatively simple, the results point to some promising avenues for future research. First we find that in the most simple case where computation time is lower than transmission time (many sensor network scenarios fit this description), increasing coverage reliably reduces latency. However, as computation takes longer relative to transmission, 100% coverage is not always ideal. Increasing coverage can force the network to perform additional work with relatively little or no gain. This effect is more pronounced for larger networks and has implications for large-scale, sensor networks. For sensor networks where computation costs may dominate (fourier transforms, poor network connectivity, etc.), a naive assignment of reduce operations, may inadvertently increase network latency.

For the scenario in which the root computation time is lower than the other nodes in the network, we find that the best solution is to hand off the processing to the root node when network sizes are small. As the network increases in size, however, performance becomes limited by the transmission capacity between the network the root node. Even in the case of an infinitely-fast root node, the network is forced to distribute reduce operations to limit data transmission.

These observations, taken together, indicate that a few heuristics may improve upon our random scheduling model. First, a node should only consider scheduling a reduce operation if there are a sufficient number of siblings to parallelize the computation. For example, if the reduce operation takes 3 times longer than a transmission, then there is benefit if there are 3 or more siblings in the same level of the tree that parallelizes that task. A node

should also consider its depth in the tree and the relative speed of the root node. If the root is much faster, then nodes near the root should avoid performing aggregations. Our expectations are to incorporate these heuristics into a distributed scheduling algorithm. This algorithm, besides benefiting multi-hop networks, also has the potential to benefit a wide range of distributed information processing systems. For example, in order to maximize node usage, a MapReduce cluster may organize the compute nodes along a tree-structure. Our algorithm could then be used to prune and restructure the tree.

V. CONCLUSION

We've presented a simple model of an information processing system that can be applied to a variety of real-world systems, including mobile MapReduce networks and sensor networks. Our model is high-level and can be used to study the latency effects of scheduling reduce/aggregation operations in the network. Our model can also accommodate nodes of varying processing capability, making it useful for heterogeneous networks. We used this model in a simulation investigating the results of randomly scheduling reduce operations in a tree-structured network. We do not intend for our random scheduler to be a realistic scheduling algorithm. Instead, we expect the data produced by our simulation to be used as a null-model for future research. Even so, there were still interesting results that we believe should be incorporated into future scheduling algorithms.

VI. ACKNOWLEDGMENT

The author would like to thank Brent Lagesse and Stephen Kelley for their suggestions and ideas. This work was partially funded by an ORNL Lab Directed Research and Development grant (LDRD 05665).

REFERENCES

- [1] J. Dean and S. Ghemawat, "Mapreduce: Simplified data processing on large clusters," 2004.
- [2] E. Meijer, B. Beckman, and G. Bierman, "Linq: reconciling object, relations and xml in the .net framework," in *ACM SIGMOD International Conference on Management of Data*, 2006.
- [3] D. P. Anderson, "Boinc: A system for public-resource computing and storage," in *IEEE/ACM International Workshop on Grid Computing (GRID)*. Washington DC, USA: IEEE Computer Society, 2004.
- [4] D. P. Anderson, J. Cobb, E. Korpela, M. Lebofsky, and D. Werthimer, "Seti@home: an experiment in public-resource computing," *Communications of the ACM*, vol. 45, no. 11, pp. 56–61, 2002.
- [5] B. Abbott, "Einstein@home search for periodic gravitational waves in ligo s4 data," *Physics Review D*, vol. 79, no. 2, p. 022001, Jan 2009.
- [6] S. M. Larson, C. D. Snow, M. Shirts, and V. S. Pande, "Folding@home and genome@home: Using distributed computing to tackle previously intractable problems in computational biology," *eprint arXiv:0901.0866*.
- [7] Y. Yu, P. K. Gunda, and M. Isard, "Distributed aggregation for data-parallel computing: interfaces and implementations," in *ACM Symposium on Operating Systems Principles (SOSP)*. New York, NY, USA: ACM, 2009, pp. 247–260.
- [8] C. Intanagonwiwat, R. Govindan, D. Estrin, J. Heidemann, and F. Silva, "Directed diffusion for wireless sensor networking," *IEEE/ACM Transactions Networking*, vol. 11, no. 1, pp. 2–16, 2003.
- [9] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong, "Tag: a tiny aggregation service for ad-hoc sensor networks," *SIGOPS Operating Systems Review*, vol. 36, no. SI, pp. 131–146, 2002.
- [10] X. Chen, X. Hu, and J. Zhu, "Minimum data aggregation time problem in wireless sensor networks," in *International Conference on Mobile Ad-Hoc and Sensor Networks (MSN)*, Wuhan, China, 2005.
- [11] B. Yu, J. Li, and Y. Li, "Distributed data aggregation scheduling in wireless sensor networks," in *IEEE Conference on Computer Communications*, Rio de Janeiro, Brazil, 2009.
- [12] T. Abdelzaher, T. He, and J. Stankovic, "Feedback control of data aggregation in sensor networks," in *IEEE Conference on Decision and Control*, Atlantis, Paradise Island, Bahamas, 2004.
- [13] T. He, J. A. Stankovic, C. Lu, and T. Abdelzaher, "Speed: A stateless protocol for real-time communication in sensor networks." Los Alamitos, CA, USA: IEEE Computer Society, 2003.
- [14] X. Zeng, R. Bagrodia, and M. Gerla, "Glomosim: a library for parallel simulation of large-scale wireless networks," *SIGSIM Simul. Dig.*, vol. 28, no. 1, pp. 154–161, 1998.