An Approach to Massively Distributed Aggregate Computing on Peer-to-Peer Networks*

Márk Jelasity
University of Bologna, Italy,
and RGAI, University of Szeged, Hungary
jelasity@cs.vu.nl

Wojtek Kowalczyk
Free University Amsterdam
The Netherlands
wojtek@cs.vu.nl

Maarten van Steen
Free University Amsterdam
The Netherlands
steen@cs.vu.nl

Abstract

The emergence of the Internet as a computing platform increases the demand for new classes of algorithms that combine massive distributed processing and complete decentralization. Moreover, these algorithms should be able to execute in an environment that is heterogeneous, changes almost continuously, and consists of millions of nodes. An important class of algorithms that can play an important role in such environments is aggregate computing: computing the aggregation of attributes such as extremal values, mean, and variance. These algorithms typically find their application in distributed data mining and systems management. We present novel, massively scalable and fully decentralized algorithms for computing aggregates, and substantiate our scalability claims through simulations and theoretical analysis.

1. Introduction

With the emergence of the Internet as a computing platform, we are seeing the need for a new class of algorithms that combine massive parallelism and inherent decentralization. This need is exemplified by the recently proposed integration of Grid technology and peer-to-peer networks [3]. This proposal recognizes the fact that massive parallel computing on the Internet will not only require a highly decentralized approach, but also demands a degree of self-organization.

*This work was partially supported by the Future & Emerging Technologies unit of the European Commission through Project BISON (IST-2001-38923).

In our own research, we are seeking solutions to massively distributed processing in which centralized coordination is not possible. Such situations typically occur in modern peer-to-peer networks [5] in which coordination is, if possible at all, handled through dynamically selected special nodes known as superpeers [8]. Given this situation, the research question we address is: What useful *massively parallel computations* can actually be executed on large-scale peer-to-peer overlay networks?

One promising research area for these goals is to find efficient implementations of computing aggregates over attribute values or data items which can be found distributed over a peer-to-peer network [6, 1]. Examples of aggregate computing include finding extremal values, mean, variance, etc. Aggregate computing as a primitive functional building block is interesting because efficient and robust implementations can be given on fully distributed large peer-to-peer overlay networks while the applications of these aggregates include important areas such as distributed data mining and systems management (maintenance, control, monitoring).

Our main contribution consists of algorithms for extremal value and average calculations. These algorithms are fully decentralized and can be executed on extremely large networks consisting of millions of nodes that communicate only through message passing. The algorithms are based on a relatively simple protocol for information dissemination and group membership management, called the *newscast protocol* [4]. We provide experimental analyses and suggest a theoretical framework for the analysis of our averaging algorithms.

In the following, we first briefly explain the newscasting protocol in Section 2.. In Section 3. we describe our algorithms, followed by theoretical and empirical results about

our averaging approach in Section 4.. We finish with a short cost analysis, discussion of related work, and conclusions.

2. The Newscast Protocol

We shall call the basic entities which form the distributed system *agents* to emphasize their possible autonomy and complexity (in other works the terms *process* and *node* are also used in similar contexts).

The newscast protocol is responsible for two functions at the same time: *membership management*, i.e., taking care of subscriptions and failures of a possibly very large group of agents, and *information dissemination* among the group members, which we call *newscasting*. The protocol is extremely simple: each agent knows only a (continuously changing) small set of peers of which one is randomly chosen to exchange information. There are no special agents, the newscast protocol is fully distributed and symmetric: each agent performs the same operations. In this section, we explain how the protocol works and summarize its key properties which are relevant for our present goal of applying it to implement aggregation.

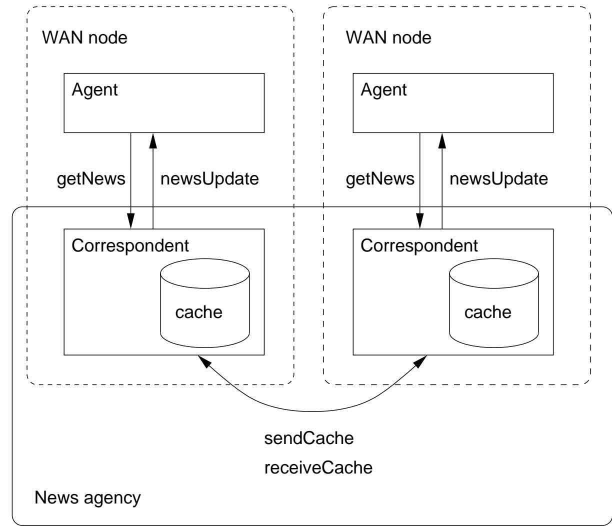


Figure 1. The conceptual organization of a newscast application.

2.1. Principal Operation

To understand how the protocol works it is useful to introduce a virtual concept, which has no corresponding implementation or location in a working newscast group, but helps grasping the underlying design philosophy. This concept is the *news agency*. The basic idea is that the news agency asks all agents regularly for *news* by means of a callback function getNews(). In addition, the news agency provides each agent with news about the other agents in the collective, again through a callback function newsUpdate(news[]). The architecture is illustrated in Figure 1.

The definition of what counts as news is application dependent. The agents simply live their lives (perform computations, listen to sensors and the news, etc.) and based on the computations they have completed and the information they have collected they must provide the news agency with news when asked.

Each agent has an associated *correspondent* that runs on the same machine that hosts the agent. The newscast protocol is implemented by the correspondent. The correspondents jointly form the distributed implementation of the news agency. Each correspondent maintains a cache of c news items, where an integer $c > 0$ is a global protocol's parameter. Whenever an agent passes a news item to its correspondent, the latter timestamps the item, adds its own network address, and subsequently caches the item. A news item itself consists of an agent identifier and the actual news as provided by the agent, as shown in Figure 2.

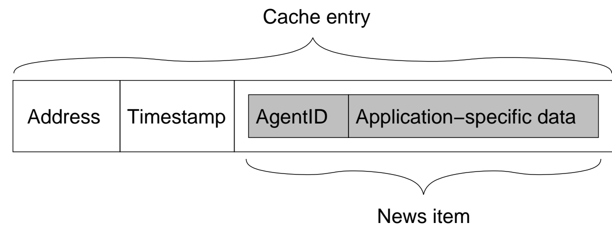


Figure 2. The format of news items and cache entries.

Correspondents regularly exchange caches as follows. Omitting some trivial technical details (which can be found in [4]), each correspondent executes the following five steps once every ΔT time units:

1. Request a fresh news item from the local agent by calling getNews(). Merge the item to the cache.
2. Randomly select a peer correspondent by considering the network address of other (and available) correspondents as found in the cache.
3. Send all cache entries to the selected peer, and, in turn, receive all the peer's cache entries (c items). Merge the received entries into the local cache.
4. Pass the received cache entries from the peer agent to the local agent by calling newsUpdate().
5. The correspondent now has at most $2c + 1$ cache entries; it subsequently throws away the oldest ones to keep the c freshest ones.

The merge operation of the cache ensures that from one agent there is at most one item in the cache. The selected peer correspondent executes the last three steps as well, so after the exchange caches of both correspondents are identical. Note, however, that as soon as any of these two correspondents executes the protocol again, its cache will most likely be different again.

We call ΔT the *cycle length*. Even though the system is not synchronized, it is often convenient to talk about *cycles* of the protocol, which are simply consecutive wall clock time intervals of length ΔT counted from some convenient starting point.

The protocol does not require that the clocks of correspondents are synchronized, but only that the timestamps of news items in a single cache are mutually consistent. This can be achieved as follows. When a correspondent A passes its cache to B, it also sends along its current local time, T_A. When B receives the cache entries, it subsequently adjusts the timestamp of each entry with a value $T_A - T_B$, effectively normalizing the time of each new entry to those already cached. We assume that the communication time between two correspondents is smaller than ΔT which must be true anyway since in each cycle at least one communication has to be completed according to the protocol.

2.2. Membership Management

The newscasting protocol disseminates correspondent addresses together with news items submitted by the agents. This automatically provides us with a simple membership management functionality.

Subscriptions do not need any special sequence of communications, the new correspondent simply has to initialize its cache with at least one known correspondent which is already a member of the group, and start to execute the protocol. Our experiments showed that the system is not sensitive to subscription patterns and tolerates the worst case when each new member subscribes through the same fixed correspondent [4].

Unsubscriptions are treated as failures. An unsubscribing correspondent simply has to stop communicating. Outdated information is quickly removed from the system so if a correspondent does not keep communicating, it will be forgotten.

2.3. Newscasting is not Broadcasting

It is important to note that newscasting is different from broadcasting, a difference that is easily overlooked and which may lead to confusion. A first observation is that newscasting is proactive, that is, it is not initiated by a single node, nor will it ever end. Second, unlike broadcasting, newscasting disseminates a given news item only to a random, relatively small group of peers. This limited dissemination is caused by the fact that, eventually, a news item is removed from a cache in favor of a fresher item.

These observations do not imply that newscasting cannot be deployed for broadcasting purposes. For example, a naive broadcasting scheme is to have an agent repeatedly return the same news when it is called back through getNews(). This scheme, however, will only slowly propagate news to all agents. A much better scheme is to let other agents store, and subsequently forward an incoming news item when requested for fresh news. This store-and-forward scheme effectively mimics a flooding algorithm. A more sophisticated solution is to deploy constrained forwarding in order to prevent nodes to receive too many duplicates.

These examples illustrate the flexibility of newscasting which allows the implementation of a wide range of communication mechanisms and computations; a flexibility we use for computing aggregates as we explain next.

2.4. Some Properties of the Protocol

Without proof, we list some important properties of the newscast protocol which are relevant for the present paper. For more details, please consult [4].

The correspondents and their cache entries at any time define a *communication graph*: the correspondents are the nodes and the cache entries define directed edges. This graph is constantly changing with time. We have shown that this graph has a very low diameter and is very close to a random graph with out-degree c. According to our results, choosing $c = 20$ is already sufficient for very stable connectivity through time, i.e., the newscast group stays constantly connected.

We have also shown that, within a single cycle, the number of cache exchanges for each correspondent can be modelled by a random variable $1 + \text{Poisson}(1)$. In other words, on average there are two exchanges per cycle (one is initiated by the local correspondent and the other one is coming in from a peer) and the variance of this estimate is 1.

3. Calculating Basic Statistics

Let us consider a system of n agents that form a newscast group, and let each agent i store one number a_i—its own value. Our objective is to program these agents in such a way, that they will collectively find, within very few cycles of the protocol, some aggregation of all values (or a good approximation of it). In this section we will illustrate the design philosophy through an algorithm for finding extremal values as a special case of aggregation and then we will focus on approximating the mean of the values, which poses a greater challenge. Using the idea presented in the mean approximation algorithm it is relatively straightforward to develop algorithms for other aggregates like variance.

3.1. Maximum

To shed some more light on how to develop applications for the model, we present a relatively simple example. The task is to find the maximum $a^* = \max_{i=1}^n a_i$. The following algorithm, which is common to all agents, will solve this problem.

```
void newsUpdate(news[]) {
    myMax =
    max(myMax,a,news[1],...,news[c])
}

NewsItem getNews() { return myMax; }
```

where $a = a_i$ for agent i.

Although there is no signal that informs the agents that the value is found, using the theory of epidemic algorithms [2] it can be proven that all agents will hear about the final solution very quickly. From the point of view of a true maximum value the algorithm is in fact an effective broadcasting mechanism, since all agents will keep returning it after they have seen it at least once. So the maximum value spreads exactly like an epidemic, "infecting" a quickly growing number of agents. Let us assume that p_i is the probability that a given agent is not infected in cycle i. Let us assume further that we have only a pull protocol, not push-pull like newscast, that is, let us assume that information goes only one way from the passive peer to the active peer that initiates the communication. Clearly, this way we have a lower bound on speed. The probability that a given agent is not infected in cycle $i + 1$ is given by

$$p_{i+1} = p_i p_i$$

since it had to be uninfected in cycle i and the node it contacted has to be uninfected as well. The initial value $p_0 = (1 - 1/n)$. It is clear that p_i decreases extremely fast.

3.2. Mean

Due to space limitations—to keep our discussion clear—we chose for presenting and analyzing the simplest possible mean approximation algorithm which needs to be started synchronously (but runs asynchronously afterwards). We have also developed, implemented and tested a restarting mechanism for making this specific algorithm adaptive. We will briefly sketch this solution at the end of this section but will not discuss it here any further.

The algorithm we propose is the following:

```
void newsUpdate(news[]) {
    if ( counter < K ) {
        myNews = a
        counter <- counter+1
    }
    else {
        myNews =
        average of items in news[]
    }
}

NewsItem getNews() { return myNews; }
```

where $a = a_i$ for agent i, K is a common parameter which is at least 1 and the variable counter is initialized to 0. During the first K activations each agent publishes its own value, thus, if K is big enough, after this initial phase there are about c copies of each value and they are evenly spread over caches of other agents. In the second phase these values are collectively averaged. Convergence is guaranteed since in each step the overall variance of the numbers in the caches of all the correspondents is decreased with a positive probability (provided the variance is non-zero) and increased with zero probability. Let us note that due to the random nature of the newscast protocol it is essential to chose K to be relatively big (say, $K > c$). Otherwise,(e.g., when $K = 1$) some values could be lost or be over-represented and that could lead to relatively big errors.

We can also expect fast convergence. Intuitively, in the system the *influence* of a single value is broadcasted at the same speed that we have seen in the maximum finding example. And this property is true for all values, because a single value can carry the influence of arbitrarily many individual original values. This is of course only an intuitive

explanation, but in the next section we will validate these claims experimentally.

Finally, over many runs, we can expect (in a probabilistic sense) convergence to the true mean because of symmetry considerations. That is, there is no systematic bias in the system towards any agent or any value. We will take a closer look at the issue of error estimation and its relation to the distribution of the values $a_1, \dots, a_n$ in the next section.

Automatic Restarting

For the sake of completeness but without rigorous analysis or validation we mention that it is possible to extend this algorithm in a way that after the approximation process is converged a new cycle is always started automatically, with a new dissemination phase and convergence phase.

The idea is that each agent keeps track of the cycle counter. When the approximation of an agent has converged according to some criterion, it makes a local decision to switch to the next cycle, and publishes this by attaching the new cycle number to its news item. Each agent also switches to a new cycle if it sees that someone else has already done so.

This mechanism, taking also into account that each agent will sense the same convergence rate throughout the system so they will not necessarily wait for someone else to start the cycle change, will change the cycle counter throughout the whole system within a couple of cycles. This way one can apply this algorithm to continuously monitor the average.

4. Performance on the Peak Distribution

We define the peak distribution as $a_1 = n$ and $a_i = 0, i = 2, \dots, n$. The peak distribution is of special interest for experimentation and theoretical analysis, because of two reasons. First, the peak distribution is the worst case when the system is maximally sensitive to initial fluctuations. Second, the performance on the peak distribution includes all information necessary to predict performance on any other distribution. This statement is rather important, we devote the following paragraphs to proving it.

First of all, observe that the algorithm uses only linear operations on the data so any approximation at any time at any agent is a linear combination of the original values. Furthermore, the outcome is independent of the way a given value set is assigned to the agents, that is, any agent can be assigned any of the values. This latter property follows from the complete symmetry of the protocol. Therefore, the output of the algorithm, that is, the estimation m of the mean, can be described as

$$m = \sum_{i=1}^{n} w_i a_i, \quad \sum_{i=1}^{n} w_i = 1 \tag{1}$$

being the value of an arbitrarily selected agent at termination (i.e. when the value has converged). The weights w_i ($i \in \{1, \dots, n\}$) stand for the relative impact of agent i and they are clearly independent from the value set a_i. In case of perfect averaging the weights should be equal. Because of the stochastic nature of our algorithms they are random variables but with identical distribution, again, due to symmetry.

Therefore, it is sufficient to consider one of these weights to gain information on how well the algorithm calculates averages. To obtain samples on one single weight consider the peak distribution. In this case we get

$$m = \sum_{i=1}^{n} w_i a_i = w_1 a_1 \tag{2}$$

This observation proves the special role of the peak distribution, and the fact that the statistical properties of the output of the averaging protocol on the peak distribution can be used to predict the outcome on any other distribution.

In other words, this allows us to draw general conclusions on the "averaging power" of our algorithms solely based on experiments with the peak data set. Furthermore, it is possible to determine the statistical behavior on all other value sets by Equation (1). For example, for an even n the values $a_1 = 0, \dots, a_{n/2} = 0, a_{n/2+1} = 1, \dots, a_n = 1$ we have the following formula.

$$m = \sum_{i=1}^{n} w_i a_i = \sum_{i=n/2+1}^{n} w_1 \tag{3}$$

Here the mean $m = 1/2$ and standard deviation $\sigma/\sqrt{n/2}$ where σ is the standard deviation of the common weight distribution.

4.1. Experimental Setup

To gain experimental data on the behavior of our system we performed runs with various number of agents (10000, 20000, and 50000) using a simulator of the newscast protocol. For each case we executed 100 independent runs with cache size 20 and terminated after 50 cycles. All runs in this section were run on the peak distribution, in accordance with the observations discussed above.

The only parameter of the algorithm, K, the length of the dissemination phase was set to $K = 20$. Using the result which says that in each newscast cycle there are $1 + \text{Poisson}(1)$ calls to getNews() (see Section 2.4.) this means that the dissemination phase is approximately 10 cycles long.

4.2. Experimental Results

Figure 3 shows the distribution of the approximation of the mean in the 50th cycle, over many runs. The correct mean

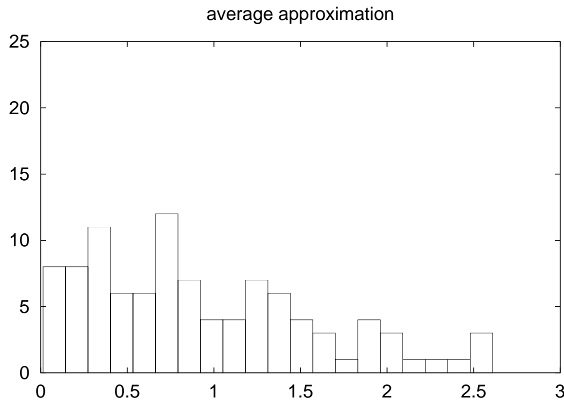


Figure 3. Histogram of the converged output in the 50th cycle collected from 100 independent runs with 10000 agents. The empirical mean and the standard deviation are 0.935 and 0.656, respectively.

is 1. The distribution in the figure shows the approximation in a fixed agent. This is acceptable because in a single run, the approximation in the 50th cycle is practically identical in each agent (see also Figure 5).

Because we use the peak distribution, as was shown in equation (2), the shape of the distribution in Figure 3 is identical to that of the common weight distribution, only in the case of the weights the mean is $1/n$ so the x axis must be scaled accordingly to get the weight distribution. Recall also, that the peak distribution is the worst case, so the variance of the mean approximations over many runs is the largest. For smoother distributions the variance is much smaller as exemplified by equation (3).

Figure 4 shows the mean and the standard deviation of the approximation of the average as a function of time (that is, newscast cycles). Deviation is also shown as a function of n, the number of agents. As previously, these statistics are collected at a single agent, over many runs. We can see that in cycle 30 the algorithm stabilizes at the final approximation and before stabilization the deviation is decreasing very rapidly. We can also see that the speed of convergence is highly insensitive to system size which indicates good scalability.

All the results mentioned so far describe statistics over multiple runs from the point of view of a fixed agent which holds $a_1 = n$. Figure 5 shows statistics of the approximations of the agents during a single run, as a function of time. It can be seen that the standard deviation over the approximations of the agents within the same system decreases exponentially in the convergence phase of the averaging algorithm (i.e., after the first 10 cycles which form the dissemination phase).

5. Cost Analysis

Figure 5 gives an indication of the convergence speed of the algorithm, in terms of the number of newscast cycles. For a more detailed analysis significantly more empirical evidence would be necessary which is outside of the scope of this paper, but the observed exponentially decreasing variance is promising and indicates good scaling properties. For the above reason, and also because the algorithm is intended to be running continuously, from this point we focus on the cost of one newscast cycle.

The cycle length, ΔT, defines the wall clock time of one newscast cycle. The communication cost of one cycle for the overall system depends on the cache parameter c (see Section 2.). In each ΔT time units each correspondent initiates exactly one information exchange session which involves the transfer of $2c$ cache entries. The size of a cache entry can be seen from Figure 2. It has a fixed-sized component and a news item, which is application dependent. In our case, a news item is a single floating point number.

Another constraint on communication cost is that c has to be at least 20 as we mentioned previously to allow a connected newscast group, but, since the newscast protocol requires only weak connectivity, c does not have to be increased with the system size (see Section 2.4.) which is an advantage for good scaling.

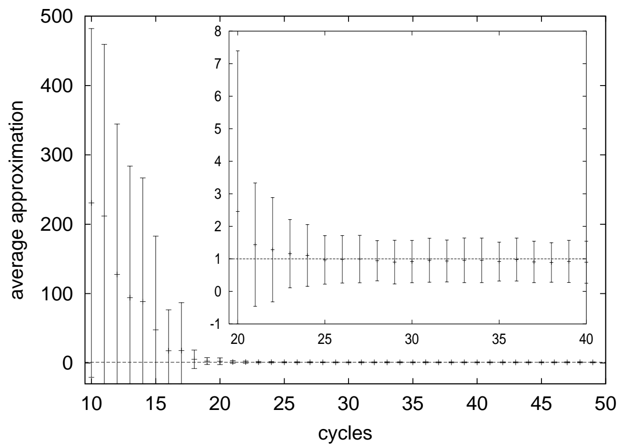
The communication costs of one cycle grow linearly if the whole system is concerned, but stay constant from the point of view of one node. This latter property allows good scaling together with the property of $1 + \text{Poisson}(1)$ communications per correspondent per cycle, which guarantees that independently of system size, a single correspondent will experience the same predictable load without peaks.

Finally, to avoid potential misunderstanding, let us recall that there is no cost related to the initial distribution of the values to be averaged, because the protocol is targeted to applications in which the values are inherently distributed, like storage capacity of nodes in a peer-to-peer network, etc.

6. Related Work

The topic of calculating aggregates in large scale fully distributed systems is relatively new. A prominent approach is Astrolabe [7] which is a hierarchical architecture for aggregation in large distributed systems. Our approach is substantially different in that it is extremely simple, lightweight, and targeted to unstructured, highly dynamic environments.

Another recent work is [1]. They discuss many approaches, based on spanning tree induction and using other,



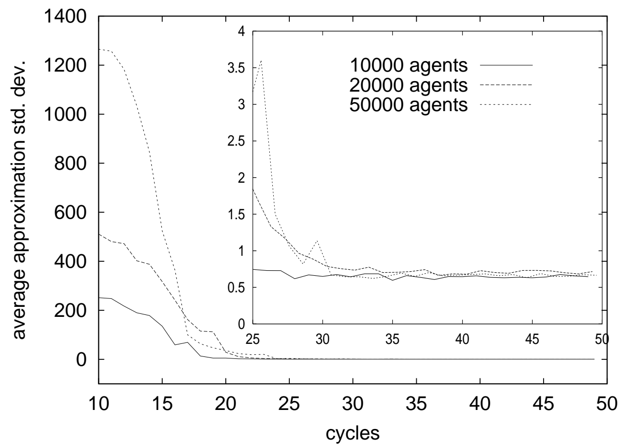
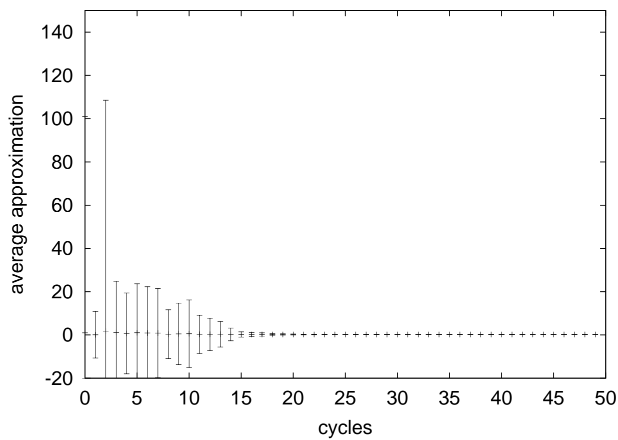


Figure 4. Statistics of the approximation of average of the agent holding $a_1 = n$ as a function of time (cycles) collected from 100 independent runs. Insets show the tails magnified. The left plot corresponds to runs with 10000 agents.



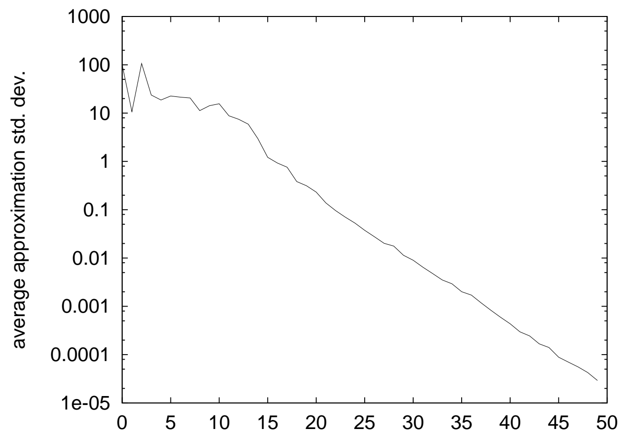


Figure 5. Statistics collected from a single run with 10000 agents over the approximations of the individual agents.

more redundant topologies. The main difference from our approach is that the protocols described there are *reactive*: aggregation is initialized from a certain point and the result is known by only that node, whereas in our case the aggregate is *proactive*: available at all nodes continuously and (in the restarted case) also follows dynamic changes in the environment. Both approaches can have their advantages in different applications.

7. Conclusions

In this paper we presented a method for finding the maximum and the mean of values that are distributed over a large fully distributed network, based on the newscast protocol. Our experimental results suggest that the convergence of the method is fast and theoretical considerations based on the observed performance on the peak distribution reveal that for realistic distributions the expected error of the estimations is very small. Due to a proactive nature of our approach an estimate of the mean is available continuously at all the nodes, and the cost of finding it is constant (does not depend on the network size).

Work on other algorithms and possible applications of the aggregates is under way, mainly in the data mining field. Our present and future work is focused on improving the quality of the estimate and extending the approach to other aggregates like quantiles and network size.

References

[1] M. Bawa, H. Garcia-Molina, A. Gionis, and R. Motwani. Estimating aggregates on a peer-to-peer network, 2003. submitted for publication.

[2] A. Demers, D. Greene, C. Hauser, W. Irish, J. Larson, S. Shenker, H. Sturgis, D. Swinehart, and D. Terry. Epidemic algorithms for replicated database management. In *Proceedings of the 6th Annual ACM Symposium on Principles of Distributed Computing (PODC'87)*, pages 1–12, Vancouver, Aug. 1987. ACM.

[3] I. Foster and A. Iamnitchi. On death, taxes, and the convergence of peer-to-peer and grid computing. In *Proceedings of the 2nd International Workshop on Peer-to-Peer Systems*, Feb. 2003.

[4] M. Jelasity and M. van Steen. Large-scale newscast computing on the Internet. Technical Report IR-503, Vrije Universiteit Amsterdam, Department of Computer Science, Amsterdam, The Netherlands, Oct. 2002.

[5] D. S. Milojicic, V. Kalogeraki, R. Lukose, K. Nagaraja, J. Pruyne, B. Richard, S. Rollins, and Z. Xu. Peer-to-peer computing. Technical Report HPL-2002-57, HP Labs, Palo Alto, 2002.

[6] R. van Renesse. The importance of aggregation. In A. Schiper, A. A. Shvartsman, H. Weatherspoon, and B. Y. Zhao, editors, *Future Directions in Distributed Computing*, number 2584 in Lecture Notes in Computer Science, pages 87–92. Springer, 2003.

[7] R. Van Renesse, K. P. Birman, and W. Vogels. Astrolabe: A robust and scalable technology for distributed system monitoring, management, and data mining, May 2003.

[8] B. Yang and H. Garcia-Molina. Designing a super-peer network. In *Proceedings of the 19th International Conference on Data Engineering (ICDE 2003)*, Los Alamitos, CA, Mar. 2003. IEEE Computer Society Press.