Adam2: Reliable Distribution Estimation in Decentralised Environments

Jan Sacha Jeff Napper Corina Stratan Guillaume Pierre

Department of Computer Science
VU University Amsterdam, The Netherlands
E-Mail: {jsacha,jnapper,cstratan,gpierre}@cs.vu.nl

***Abstract*—To enable decentralised actions in very large distributed systems, it is often important to provide the nodes with global knowledge about the values of attributes across all nodes. This paper shows how, given an attribute whose values are distributed across a large decentralised system, each node can efficiently estimate the statistical distribution of these values. Simulations using heavily skewed real-world node attribute distributions show that our estimation methods outperform the state-of-the-art heuristics by an order of magnitude with an average error of 0.05% and a maximum error of 2%. To obtain this accuracy, each node sends on average just 120 kB of data independent of the system size. Our algorithms also achieve this accuracy in the presence of heavy churn of system membership. Furthermore, our algorithm enables self-tuning by continuously estimating the accuracy of its own distribution approximation.**

I. Introduction

Large-scale distributed systems such as computing clouds, Grids, and peer-to-peer overlays are increasingly relying on decentralisation, by scattering application-level and system-level information across a very large collection of loosely-coupled nodes. While this design principle has demonstrated its effectiveness in terms of scalability and robustness, it introduces serious challenges for global-level tasks like monitoring and optimisation. One important challenge is to estimate global system properties, for example the total available storage space or the average load on the machines. Such global properties can be computed in a decentralised fashion by using aggregation protocols [1].

Aggregation protocols allow for the computation of a global scalar value – like a mean, a total count, or the rank of a node compared to the others according to a given metric. However, this is often not sufficient for effectively monitoring and optimising a distributed system. In many cases it is necessary to have a view of how properties vary across the system's nodes, by estimating their statistical distribution. For example, in a fully decentralised load balancing mechanism, any node can detect a global load imbalance by monitoring the statistical distribution of the load at all other nodes. Estimating the statistical distribution of attribute values also allows identifying outliers and clusters, which can be used to detect hardware and software defects or intrusion attempts [2].

The main challenge for a decentralised protocol that estimates statistical distributions of attribute values is to obtain good accuracy while maintaining low communication overhead. Achieving good accuracy is particularly difficult in real-world systems where sets of values are skewed and hard to approximate by synthetic distributions. Although some solutions for decentralised estimation of statistical distributions have been recently proposed, they suffer from poor accuracy for skewed distributions or high communication overhead [3], [4].

This paper presents Adam2, a new decentralised protocol to accurately estimate the distribution of an attribute with a low communication cost[1]. Our algorithm continuously monitors the accuracy of its own distribution estimation. This feedback mechanism allows an application to dynamically tradeoff estimation accuracy to further reduce overhead.

Adam2 is based on a gossip communication model. Gossip protocols provide scalable and robust information dissemination, while achieving a low overhead [1]. We use a gossip-based method to efficiently and autonomously estimate the fraction of nodes in the system whose attribute value is lower than a certain threshold. By applying this method to a set of thresholds, each node quickly estimates the distribution of the attribute across the whole system. The set of thresholds determines the accuracy of the distribution approximation. To improve accuracy, our algorithm refines the set of thresholds across multiple consecutive *instances*. The thresholds of an instance are based on the results from the previous instance. Nodes execute the entire algorithm autonomously without any designated coordinator.

We evaluate our algorithm through simulations using real-world attribute distributions with different characteristics [5]. We show that even for heavily skewed distributions it reduces the maximum vertical distance between the real distribution curve and the estimated one to about 2%, and the average distance between the two curves to about 0.05%, while each node sends on average 120 kB of data independent of system size. This level of accuracy is one order of magnitude better than current state-of-the-art systems.

[1]Adam2 was named after Adam1, who increased world population by one on October 28th, 2009. For obscure reasons, Adam Sacha v1.0 was registered at the municipality with no release number.

This paper is organised as follows. Section II discusses related work. Section III presents our system model. Sections IV, V, and VI describe the algorithms that estimate an attribute distribution using one aggregation instance, refine the set of interpolation points between multiple instances, and estimate the approximation accuracy. Section VII discusses our evaluations, and Section VIII concludes.

II. Related Work

The task of data aggregation, or synopsis construction, has been well-studied in the past in the areas of sensor networks [6] and distributed databases [7]. However, most of the proposed algorithms are reactive. Each time a node requests aggregation, a dissemination tree (or weighted graph) is constructed between nodes in order to collect the required data from the system. Such graphs are neither robust to failures of nodes near the sink nor do they efficiently disseminate the result to all nodes.

Adam2 is based on gossip protocols, which are renowned for their scalability, robustness, and low cost [1]. These protocols have been used to approximate simple system properties such as minimum, maximum, and mean values of an attribute. We extend them by allowing nodes to approximate system-wide distributions and to assess and improve the accuracy of these approximations.

Several algorithms allow nodes to estimate their own ranks and slices [8]–[10]. While these solutions incur less overhead, they provide more limited information than a distribution estimation. For example, they do not enable nodes to estimate whether an attribute distribution is skewed, imbalanced, or contains outliers: node ranks by definition are always assigned between 1 and N (system size), regardless of the actual attribute distribution.

The problem of outlier detection is addressed using gossip by [2]. Nodes gossip synopses of clusters and outliers to enable both the removal of outliers and the discovery of cluster formation. However, the cluster synopses do not estimate the full distribution of node parameters. Adam2 is also well-suited to other distributions without clusters.

A simple way to estimate an attribute distribution is to draw a random sample of attribute values [4], [11]. However, as we show section VII, such an approach is extremely inefficient compared to Adam2.

Haridasan et al. estimate an attribute distribution by gossipping synopses of equi-depth histograms [3]. Using equi-depth bins, the system converges towards an accuracy around 7% in the absence of churn. Adam2 obtains a much better accuracy under the same conditions. Furthermore, it also estimates its own accuracy to enable a tradeoff of accuracy for communication overhead.

III. System Model

We consider a distributed system consisting of a large number of autonomous nodes. The goal of Adam2 is to estimate, in a scalable fashion, the cumulative distribution of some discrete attribute A at every node in the system. Nodes are called *peers* because we assume no central point of control and all nodes participate equally in the algorithm. The peers are organised in a P2P overlay where each peer maintains links to a small number of randomly selected nodes called its *neighbours*. The set of neighbours of a peer changes over time, as peers exchange neighbour lists to obtain robust connectivity [11].

The cumulative distribution function (CDF) for an attribute A is defined as a function $F : \mathbb{R} \rightarrow \mathbb{R}$ such that $F(x)$ is equal to the fraction of nodes that have a value for A at or below x:

$$F(x) = \frac{1}{N}\Big|\{p : A(p) \leq x\}\Big|$$

where N denotes the system size.

To approximate F, we estimate the function at a subset of the discrete points, keeping the results in a data structure similar to a cumulative histogram. Specifically, we define a sequence of λ elements, called H, where the i'th element, $H(i)$, contains a pair (t_i, f_i) representing the fraction of peers f_i that have a value for A at or below the *threshold* t_i :

$$f_i = \frac{1}{N}\Big|\{p : A(p) \leq t_i\}\Big|$$

The thresholds can be chosen arbitrarily within the attribute domain. Each element corresponds to a single CDF value, since $F(t_i) = f_i$ for $1 \leq i \leq \lambda$. Hence, the CDF function can be approximated by interpolating the points of H. We discuss in the next section how to efficiently and accurately obtain the pairs (t_i, f_i).

We measure the CDF approximation accuracy using two classical metrics. The Kolmogorov-Smirnoff (or maximum error) metric defines the distance between function F and its approximation F_p at node p as: $\sup_x |F(x) - F_p(x)|$. Given that the attribute space in our system is discrete, we define the maximum error of F_p as:

$$\mathrm{Err}_m(p) = \max_x |F(x) - F_p(x)|$$

Since different peers in the overlay can generate slightly different distribution estimations, we calculate the corresponding aggregate of these metrics over all peers:

$$Err_m = \max_p \mathrm{Err}_m(p)$$

This error metric provides an upper bound on the approximation error of any peer in the system.

While Err_m is useful to bound the error that any peer observes, this bound is determined by a single point discrepancy between F and F_p. Hence, it is quite sensitive to noise. A common approach to summarise the error contributed by all points calculates the area between the two curves: $\int_x |F(x) - F_p(x)| \ dx$. In the discrete case, this metric corresponds to a sum of $|F(x) - F_p(x)|$ over all attribute

values. We use the average vertical distance between F and F_p to allow comparisons of error across different attributes:

$$\mathrm{Err}_a(p) = \sum_{x=min}^{max} \frac{|F(x) - F_p(x)|}{max - min}$$

Again, we calculate an aggregate across all peers:

$$Err_a = \operatorname*{avg}_p \mathrm{Err}_a(p)$$

IV. CDF Approximation Algorithm

Our CDF approximation algorithm is based on periodic gossip rounds, executed at roughly the same rate by all nodes, where neighbouring nodes exchange information. A sequence of several gossip rounds, called an *aggregation instance,* generates a new CDF approximation at all nodes in the system. Nodes occasionally initiate new instances to improve the estimation accuracy and adapt to system churn.

An aggregation instance is started probabilistically by any node, which selects a set of t_i thresholds and epidemically spreads the information about the new instance and the thresholds to other nodes using gossip (see Figure 1). The nodes run an averaging protocol wherein nodes exchange their current values during each round of gossip and calculate new values by averaging the current and received values [1]. In order to calculate the fraction f_i of nodes that have attribute values below (or at) t_i, a peer p enters the averaging protocol with a value of 1 if $A(p) \leq t_i$ and 0 otherwise. Through a sequence of gossip exchanges and the corresponding averaging, the nodes estimate the mean of all the introduced values, which is equal to f_i.

Similarly, nodes estimate the system size N using the averaging protocol: Each peer p enters the protocol with a *weight* variable $w_p = 0$, except the unique initiator q of the instance which sets $w_q = 1$. Over successive exchanges, the mean approaches $1/N$.

The accuracy of the averaging protocol increases exponentially with time. After a fixed number of rounds, all nodes update their CDF and system size estimations, and terminate the aggregation instance.

We associate each aggregation instance with a unique instance identifier *id*. The instances may overlap in time, and thus a peer may participate in multiple independent instances simultaneously. Since the instances are executed in isolation from each other, we simplify the algorithm description and assume only one running aggregation instance.

Starting an Aggregation Instance: Any peer in the system may start a new aggregation instance. Other peers learn of new instances through the regular gossip exchanges. To prevent the system from being overwhelmed by new instances, a peer starts a new instance with probability P_s per round calculated as $\frac{1}{N_p R}$. N_p is the current estimation of N at peer p generated in a previous aggregation instance (nodes joining the system are bootstrapped by their initial neighbours), and R is the system constant that regulates the frequency of new aggregation instances. In a stable state, with a steady number of peers in the system, a new aggregation instance is created on average every R rounds.

```
1: // Executed by a probabilistically self-selected
2: // node at the beginning of an instance
3: StartInstance(p):
4: {t_i} ← select λ interpolation points
5: H_p ← {(t_i, f_i) | f_i = 1 iff A(p) ≤ t_i; 0 otherwise}

6: // Run by each node in each round
7: Round(p):
8: q ← select random neighbour
9: sendRequest (H_p) to q
10: receiveResponse (H_q) from q
11: Merge(H_q)
12: while round has not finished do
13:     receiveRequest (H_n) from n
14:     sendResponse (H_p) to n
15:     Merge(H_n)
16: end while

17: Merge(H_q):
18: if H_q ≠ ∅ then
19:     let H_q = {(t_i, f_i)}
20:     if H_p = ∅ then
21:         H_p ← {(t_i, f'_i) | f'_i = 1 iff A(p) ≤ t_i; 0 otherwise}
22:     end if
23:     H_p ← {(t_i, (f_i + f'_i)/2)}
24: end if
```

Figure 1. Aggregation algorithm at peer p. For simplicity, the system size estimation and instance termination mechanisms are not shown. The H_p variable is initialised with $\emptyset$ at all peers.

For each aggregation instance, peer p stores the set of interpolation points H_p and the weight w_p that it uses to estimate the system size. To start a new instance, peer p first sets weight $w_p = 1$, then selects a set of threshold values t_i using the SELECTPOINTS procedure described later, and finally generates an initial set of interpolation points $H_p = \{(t_i, f_i) \mid 1 \leq i \leq \lambda,\ f_i = 1 \text{ if } A(p) \leq t_i;\ 0 \text{ otherwise}\}$.

Executing an Aggregation Instance: An aggregation instance comprises a sequence of gossip rounds. A gossip exchange between peers p and q is entirely symmetric: Peer p sends H_p and w_p to q, and peer q replies with H_q and w_q. Both peers then merge the received values in the MERGE procedure. If either peer has yet not seen the aggregation instance *id*, it *joins* the instance by sending an empty set, initialising its data structures, and merging the received values. The other peer then ignores the exchange on receiving an empty set. When a peer joins an instance, it initialises its data structures by setting weight $w_p = 0$ and creating an initial set of interpolation points $H_p = \{(t_i, f_i) \mid 1 \leq i \leq \lambda,\ f_i = 1 \text{ if } A(p) \leq t_i;\ 0 \text{ otherwise}\}$. Note that all peers use the same thresholds t_i to initialise H_p as assigned by the peer that started the instance. Finally, both peers average the w_p and w_q weights and merge H_p and H_q by averaging the corresponding f_i values.

Terminating an Aggregation Instance: Every instance is associated with a time-to-live counter, which is reduced by one per round at each peer. For simplicity, this mechanism is not shown in Figure 1. When an instance ends, each peer p updates its estimation of the number of nodes in the system $N_p = \frac{1}{w_p}$ and approximates the whole attribute CDF by interpolating the points of H_p. We use simple linear regression between each consecutive pair of points to obtain F_p, but more complex approaches are possible. Finally, each peer deletes its H_p set and stops participating in the aggregation instance.

Extreme CDF Values: So far, for simplicity we have ignored two special points in any approximation: the first and last. Adam2 finds the minimum and maximum attribute values to use in later aggregation instances. In order to discover these values, the local minimum and maximum at each node are added to H and treated specially. When merging tuples, their corresponding minimum and maximum values are chosen. All nodes then quickly converge on the global minimum and maximum for all attribute values.

Multiple Attribute Values per Node: The aggregation algorithm can be easily extended to handle cases where individual nodes are allowed to have multiple attribute values. For example, to estimate the distribution of file sizes at all nodes in the system each node contributes its set of file sizes. In this case, we define $A(p) \subset A$ as the set of values for attribute A at peer p and A as the set of all attribute values at all nodes in the system. The CDF for attribute A is defined as function $F : \mathbb{R} \rightarrow \mathbb{R}$ such that

$$F(x) = \frac{\left|\{a \in A : a \leq x\}\right|}{|A|}$$

As previously, the CDF is approximated by calculating the value for F in a set of discrete points (t_i, f_i) where $F(t_i) = f_i$. To calculate f_i, nodes generate two values using the averaging algorithm. First, each node p calculates avg_i – the average number of attribute values below t_i per node – by contributing $|\{a \in A(p) : a \leq t_i\}|$ to the averaging algorithm. Second, each node p calculates avg – the average number of attributes per node – by contributing $|A(p)|$ to the averaging algorithm. Note that avg is independent of i and can be calculated once for all the CDF points. The f_i value is then given as $f_i = \frac{avg_i}{avg}$.

V. Interpolation Point Selection

When starting a new aggregation instance, each peer needs to decide on the placement of the interpolations points in H. Initially a node may have no prior knowledge about the attribute distribution. The simplest approach in this case is to spread the interpolation points at uniform intervals within the attribute domain. However, the distributions of node characteristics in large-scale distributed systems are often highly skewed [5], resulting in a poor approximation using uniform intervals. We show in Section VII-B better performance using attribute values found in a subset of neighbours of the initiating node.

Once the system has a rough estimate of the attribute distribution, it can further refine the selection of interpolation points in future instances, and reduce the CDF approximation error. Different selection algorithms may be used depending on the metric that the application tries to optimise.

A. Minimising the Maximum Error

One of the simplest threshold selection heuristics to reduce $\mathrm{Err}_m(p)$, which we call *HCut*, chooses the interpolation points for a new aggregation instance such that they divide the image of H_p into $(\lambda + 1)$ equal size quantiles. Since $\mathrm{Err}_m(p)$ is determined by the maximum vertical distance between interpolation points, this heuristic attempts to bound the maximum error to $\frac{1}{\lambda+1}$, assuming the CDF does not change between aggregation instances. Figure 2(a) illustrates the HCut algorithm: the interpolation points for the next aggregation instance (t_1, t_2, t_3) correspond to 25%, 50%, and 75% quantiles.

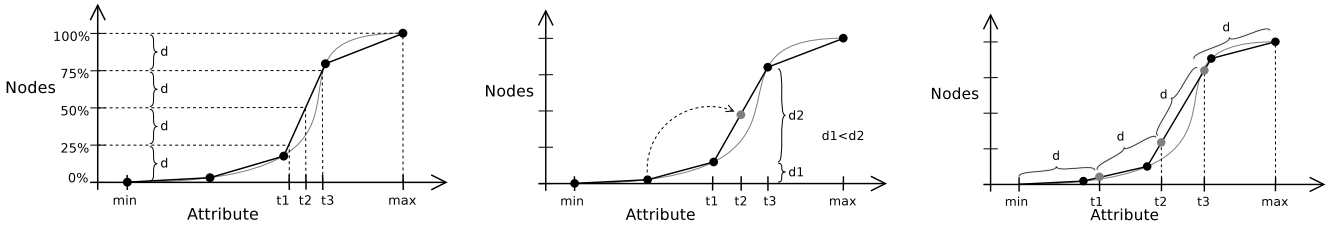
The HCut algorithm is efficient for approximating smooth CDFs. However, in many systems the number of possible attribute values is small. For example, many PCs have 512 MB, 1 GB, or 2 GB of RAM, but relatively few current machines have an amount of RAM that is between these values. The CDFs of such real-world attributes are step functions that are poorly approximated by HCut.

To approximate these CDFs, we propose *MinMax* – a heuristic that attempts to identify and approximate steps in a CDF curve. Figure 3 shows the pseudocode for MinMax. MinMax iteratively finds the farthest two consecutive interpolation points by vertical distance, denoted n and $n-1$, in the previous set of interpolation points H_{old}, and the closest three interpolation points by vertical distance in H denoted $m-1$, m, and $m+1$. If the two farthest points are farther apart than the closest three, the midpoint m of the closest three is removed from both H and H_{old}, and a new point is added to H at the new, interpolated midpoint between n and $n-1$. When no points satisfy the condition, the thresholds in H are returned as the output of the algorithm.

A sample MinMax step is graphically illustrated in Figure 2(b). MinMax changes the interpolation points only if it is expecting to reduce the interpolation error. By iteratively splitting the steepest fragments in the interpolated curve over multiple aggregation instances, MinMax efficiently identifies steps in the CDF.

B. Minimising the Average Error

The HCut and MinMax heuristics attempt to minimise the maximum vertical distance measured by $\mathrm{Err}_m(p)$. However, $\mathrm{Err}_a(p)$ depends upon the area between the CDF and the interpolation. To reduce the area, we consider the *LCut* heuristic that selects the interpolation points based on their Euclidean distance instead of vertical distance.



(a) Interpolation point selection using HCut. The grey curve represents F– the true CDF. The black line represents H_p – the previous CDF interpolation at p. The selected points are $t1$, $t2$, and $t3$.

(b) Interpolation point selection using MinMax. The vertical distance d_1 is less than d_2. Moving the midpoint of the first segment to the second at point t_2 is thus likely to reduce $Err_a(p)$.

(c) Interpolation point selection using LCut. Points $\{t_1, t_2, t_3\}$ are chosen to divide equally the Euclidean distance along the previous approximation H.

Figure 2. The HCut, MinMax and LCut heuristics

SelectPoints(H):
$H_{old} \leftarrow H$
loop
  find n that maximises $|f_n - f_{n-1}|$ in H
  find m that minimises $|f_{m+1} - f_{m-1}|$ in H_{old}
  if $|f_n - f_{n-1}| > |f_{m+1} - f_{m-1}|$ **then**
    remove point (t_m, f_m) from H and H_{old}
    add point $(\frac{t_n+t_{n-1}}{2}, \frac{f_n+f_{n-1}}{2})$ to H
  else
    return H
  end if
end loop

Figure 3. MinMax interpolation point selection algorithm. The algorithm iteratively attempts to split the widest vertical gap while removing the midpoint from the narrowest cluster of three points.

Figure 2(c) illustrates the LCut heuristic. It first calculates the length of the H linear interpolation curve for the previous aggregation instance. Then, it divides the H curve into λ equal length (by Euclidean distance) segments to determine the new point placement. The horizontal axis is scaled by $max - min$ in order to equalise the horizontal and vertical coordinate ranges. As shown later in Section VII-C, LCut achieves lower average interpolation error Err_a than HCut and MinMax, but higher maximum error Err_m.

VI. Dynamic Confidence Estimation

Adam2 also allows each node to estimate its own CDF approximation accuracy. This can be used to dynamically tune the algorithm parameters – such as the number of interpolation points and the number of executed instances – according to application-specific accuracy requirements.

The accuracy estimation is based on the fact that nodes estimate CDF values very accurately at the points of H. To estimate the approximation accuracy, the node that starts the instance generates an additional set of *verification points* V similar to the interpolation points H where each element in V is a pair (t'_i, f'_i) such that $F(t'_i) = f'_i$. The extra V points are added to aggregation algorithm to be gossipped and merged along with the original H points.

The t'_i thresholds for the verification points are chosen by the node that initiates a new aggregation instance according to the selected error metric. In order to estimate the average CDF approximation accuracy $Err_a(p)$, the t'_i thresholds are selected uniformly between the attribute minimum and maximum. At the end of an instance, each peer p estimates the accuracy of its CDF approximation F_p as

$$EstErr_a(p) = \underset{(t'_i, f'_i) \in V_p}{\text{avg}} |F_p(t'_i) - f'_i|$$

The maximum approximation error $Err_m(p)$ is generally more difficult to estimate compared to $Err_a(p)$ since it is determined by a single point in the CDF. In order to estimate $Err_m(p)$, the peer q that starts a new aggregation instance selects the verification points V_q based on its current CDF interpolation. Specifically, the V_q points are inserted between the H_q points by iteratively dividing the farthest two points in H_q by vertical distance. This way, peer q attempts to find the attribute values at which the true CDF and the interpolated curve most differ. When an instance ends, each peer p estimates its approximation accuracy as

$$EstErr_m(p) = \max_{(t'_i, f'_i) \in V_p} |F_p(t'_i) - f'_i|$$

VII. Performance Evaluation

We evaluate Adam2 in PeerSim, a simulator for peer-to-peer systems [12]. This allows us to evaluate systems with 100,000 nodes, which would be infeasible using a real-world deployment. Unless specified otherwise, all evaluations are based on 100,000 nodes and $\lambda = 50$ interpolation points.

We did not use a synthetic distribution of attribute values. Synthetic distributions are typically smooth and therefore easier to approximate. We instead use real-world data from the BOINC volunteer computing project where skew and discontinuities occur [5]. For each machine that participated in BOINC in 2008, we extracted several attributes, including: measured CPU performance in FLOPS, measured downstream bandwidth, amount of installed memory, and amount of installed disk space. We filtered out samples

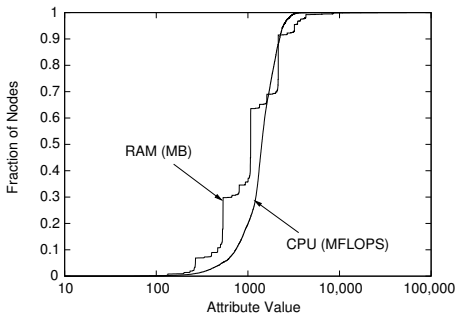


Figure 4. Actual attribute distributions F from the BOINC project [5].

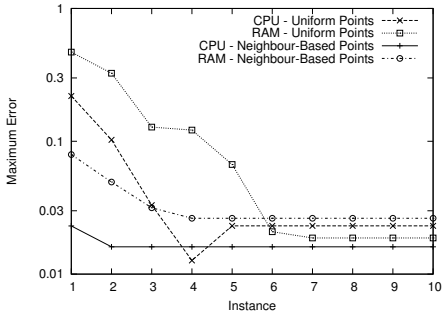


Figure 5. Accuracy of MinMax using different bootstrap approaches.

from the trace that result from obviously faulty readings (for example, a machine with a bandwidth capacity above 10^{31} bps or one with a negative amount of memory). For ease of presentation, we present in this paper only the experiments with the CPU and RAM attributes. The other attributes generated similar results. Figure 4 shows the actual CDFs of the CPU and RAM attributes. The CPU attribute has a smooth distribution, while the CDF for RAM contains visible steps. Step functions are harder to estimate accurately because their curves are not suited to simple interpolation algorithms.

We compare Adam2 with two other CDF estimation approaches: the histogram-based EquiDepth heuristic [3] and random sampling [4]. In the latter approach we construct an attribute CDF based on a random subset of attribute values drawn from the system. For each algorithm, we measure the maximum approximation error Err_m and the average approximation error Err_a. We show results from single runs of the algorithms; multiple runs produce similar results.

A. Single-instance CDF Estimation Accuracy

Figure 6(a) plots the Err_a and Err_m metrics at each protocol round within a single aggregation instance. We compare the accuracy obtained at the interpolation points with those of the entire CDF domain. For clarity, only the RAM attribute is displayed – the algorithm generates similar results for the CPU attribute. During the first few rounds, not all nodes have joined the aggregation instance and the error is equal to the maximum value of one. However, starting from round 10, the error at the interpolation points decreases at an almost perfectly exponential rate, and quickly becomes negligible. After 70 rounds it reaches the level of hardware rounding errors. We consider 25 rounds sufficient to accurately calculate the CDF at the interpolation points. The standard deviation of our error metrics across all system nodes remains below 10^{-5}, and hence, in a single aggregation instance all peers generate nearly identical CDF approximations.

At the same time, the Err_m and Err_a error over the entire CDF domain does not decrease below a few percent due to interpolation errors. To further reduce the interpolation error, nodes need to either add new points to H or select a new set of interpolation points that better fits the CDF curve.

Figure 6(b) plots the performance of EquiDepth in identical settings. Both algorithms observe similar errors over the entire CDF (8% max for Adam2 and 10% max for EquiDepth). However, the approximation error over time at the selected bins does not improve in the EquiDepth approach. EquiDepth incurs a significant approximation error due to sample duplication. The very high accuracy of Adam2 at the selected points is essential both to dynamically gauge the accuracy of its own CDF estimation and to refine the selection of points in later aggregation instances.

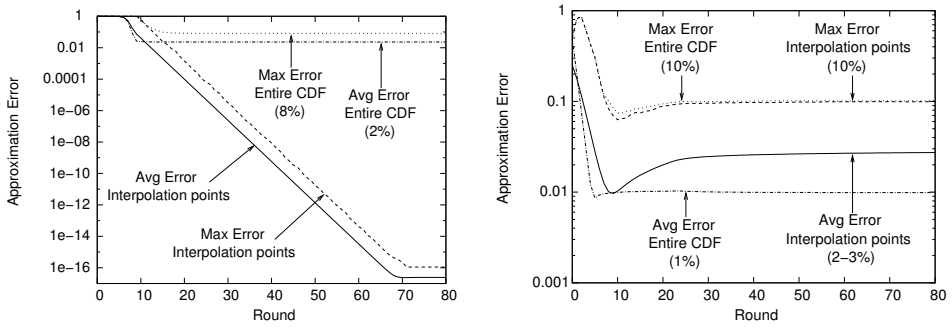
B. Initial Interpolation Point Selection

Section V discusses the selection of the initial set of interpolation points in the absence of a previous CDF estimation. Figure 5 compares two simple approaches: 1) assigning interpolation points uniformly between the minimum and maximum attribute value (labelled "Uniform Points"); and 2) using a random subset of the attribute values of the peer's neighbours in the P2P overlay ("Neighbour-Based Points").

The results clearly demonstrate that the Neighbour-based approach significantly improves the algorithm's convergence. We believe that since MinMax spreads the interpolation points according to the attribute distribution, taking the initial interpolation points from neighbours bootstraps the algorithm with points already from the desired distribution. Further, we also see that MinMax converges much faster for the smooth CPU distribution than for the heavily-skewed RAM distribution where a precise selection of interpolation points is crucial for accuracy. Similar results hold for the other refinement algorithms. In the next sections we always use the neighbour-based approach to bootstrap aggregation instances.

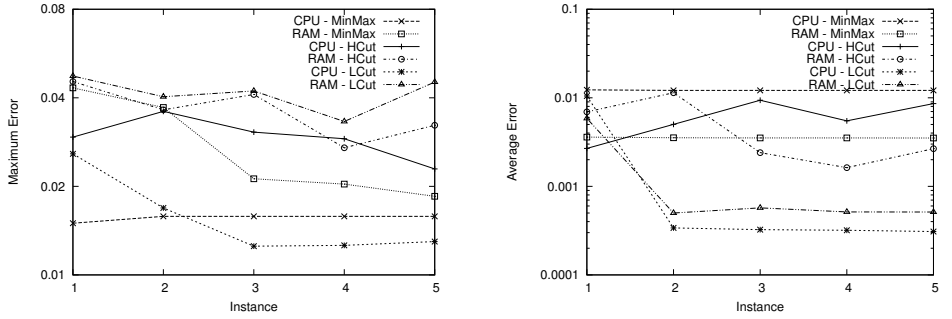
C. Convergence over Multiple Instances

We now compare Adam2 with EquiDepth and random sampling over multiple aggregation instances. Figure 7 shows that multiple instances in Adam2 effectively improve accuracy. All algorithms achieve good Err_m results for smooth distributions (CPU). For a heavily-skewed attributes



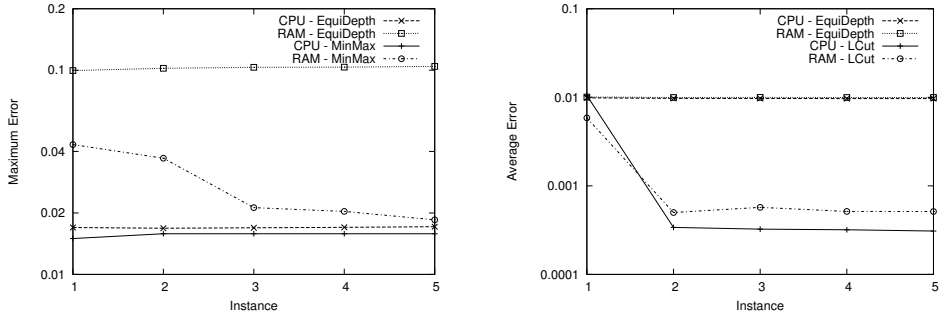
(a) Adam2 (b) EquiDepth

Figure 6. Approximation accuracy over one aggregation instance (RAM).



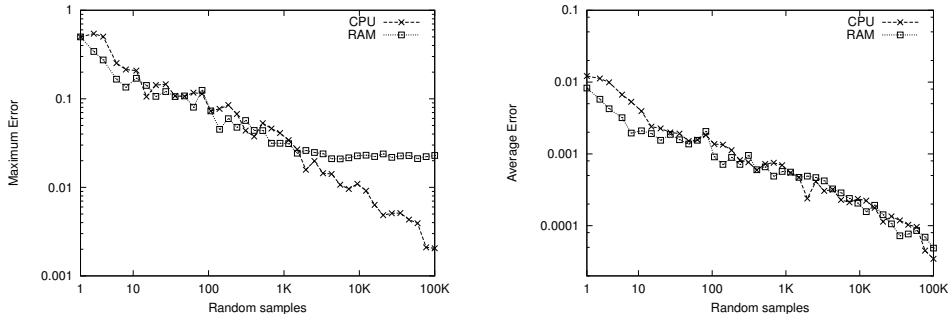
(a) Maximum distance (Err_m) (b) Average distance (Err_a)

Figure 7. Comparison between HCut, MinMax, and LCut over multiple instances.



(a) Maximum distance (Err_m) (b) Average distance (Err_a)

Figure 8. Approximation error in EquiDepth over multiple phases.



(a) Maximum distance (Err_m) (b) Average distance (Err_a)

Figure 9. Approximation error for random sampling.

such as RAM, MinMax significantly outperforms the others because it efficiently identifies the steps in the distribution. We thus focus on the MinMax algorithm when minimising Err_m in the remaining experiments. For Err_a, after 3 instances LCut achieves an order of magnitude improvement compared to other algorithms. We similarly focus on LCut when minimising Err_a in later experiments.

The performance differences of MinMax and LCut for each error metric demonstrates the difficulty of optimising for both metrics simultaneously. Although LCut performs best for Err_a and also for Err_m for smooth CDFs, it has the worst performance for Err_m for skewed CDFs where precise point selection is crucial. Defining a single heuristic that works well for both metrics and diverse CDF shapes remains as future work.

We compare these results with those of EquiDepth and random sampling, respectively in Figures 8 and 9. We execute the EquiDepth phases with the same frequency, duration, and number of bins as aggregation instances to make comparison as fair as possible. Since EquiDepth does not refine its histogram bins based on previous CDF estimations, it generates the same error in every phase. Consequently, EquiDepth's Err_m is a few times higher than MinMax, particularly for step-like CDFs. For Err_a it performs an order of magnitude worse than Adam2 using LCut.

The accuracy of random sampling depends on the sample size. In our 100,000-node system, about 1,000 to 10,000 random samples are necessary to achieve an accuracy similar to that of MinMax or LCut. Drawing these random samples using [4] would however generate several network messages per requested sample – a prohibitive cost compared to our approach. We finally note that the error measurements for random sampling are higher for heavily-skewed CDFs compared to smooth CDFs.

D. Influence of the Number of Interpolation Points

One way to improve accuracy is to increase the number of interpolation points. This section explores the tradeoff between the number of interpolation points (and hence the communication costs) and the obtained accuracy.

Figure 10 plots Err_m and Err_a after 4 instances (phases) in Adam2 and EquiDepth when using between 10 and 100 interpolation points (bins). Clearly, more interpolation points bring better accuracy. The slight variations in the graph can be explained by the random component of our algorithms. As previously, EquiDepth is outperformed by MinMax with the Err_m metric and LCut with the Err_a metric.

50 points provide acceptable accuracy for many possible applications: $Err_m \sim 2\%$ using MinMax, or $Err_a \sim 0.1\%$ using LCut. However, for the applications that need higher accuracy, increasing the number of points incurs modest performance penalty: with 10 extra points, the size of the messages increases by about 160 bytes; for current networks this is almost negligible. Furthermore, if the CDF does not change significantly over time, nodes can combine interpolation points obtained over multiple aggregation instances to further reduce the overall estimation errors.

E. Scalability

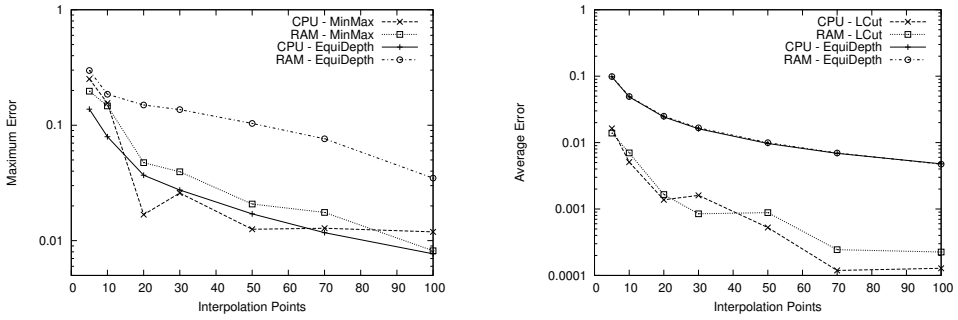
Figure 11 shows the relationship between the number of nodes in the system and the CDF approximation accuracy using Adam2. Due to randomisation, the Err_m error varies between the measurements, but the error remains in the same order of magnitude. The Err_a error decreases for larger systems due to the longer tail of the distribution. Larger populations of nodes in our experiments have probabilistically higher maximum attribute values, and the longer tails of the attribute CDFs are easily approximated using linear interpolation. Since the Err_a error is calculated over all attribute values, CDFs with longer tails produce lower approximation error.

Adam2 has only one configuration parameter that depends on the system size: the instance time-to-live. During one instance, the information about attribute thresholds t_i needs to be propagated to all nodes in the system, and all nodes need to estimate the corresponding f_i fractions using the averaging protocol. However, since the propagation speed for a push-pull epidemic is exponential and the mean estimation algorithm converges at an exponential rate, we argue that an instance duration of a few dozen rounds is sufficient to obtain high CDF approximation accuracy even in extremely large systems.

F. Dynamic Attribute Distributions

We have focused so far only on systems in which node attribute values and attribute CDFs do not change over time. In systems where attribute CDFs are dynamic, the approximation accuracy depends not only on the aggregation and interpolation errors but also on the rate at which the actual CDF changes. In Adam2, a node evaluates its attribute value only when it creates or joins a new aggregation instance. The node then runs the protocol until the end of the instance irrespective of any changes in the current attribute value. The CDF estimation error at the end of an instance, defined by the Err_m and Err_a metrics, thus depends both on the accuracy of aggregation, which we have evaluated in the previous sections, and the difference between the attribute CDFs at the beginning and end of the instance, which is entirely application specific.

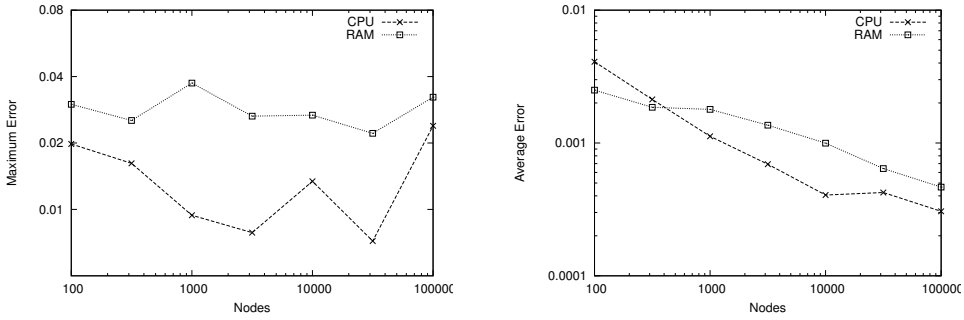
The accuracy of dynamic CDF estimation can be improved by decreasing the instance duration, for example by reducing the gossip period. When nodes gossip more frequently, they exchange data at a higher rate, but the total communication cost per instance (and hence the cost of CDF approximation) remains unchanged since nodes generate exactly the same number of messages. The minimum gossip period is determined by the message round-trip time, since



(a) Maximum distance (Err_m)

(b) Average distance (Err_a)

Figure 10. Influence of the number of interpolation points on aggregation accuracy.



(a) Maximum error (Err_m)

(b) Average error (Err_a)

Figure 11. Influence of the system size on approximation accuracy.

nodes need to exchange a pair of messages during each gossip round.

G. Impact of Churn

Many large distributed systems exhibit a slow, continuous change in membership called churn. Any monitoring system must therefore adapt to such changes. We model churn by randomly removing a fixed fraction of nodes in the overlay with new nodes at each simulation round. We set a churn rate typical of P2P systems [13]: Assuming a gossip periodicity of one second and a mean session duration of 15 minutes, approximately 0.1% of nodes leave the system per round and rejoin with a different attribute value drawn from the same distribution. We do not model changes in the attribute distribution, as such changes are entirely application specific.

Figure 12(a) shows Err_m and Err_a in one aggregation instance under churn. The evaluation metrics do not include nodes that join the system during the instance execution, since their CDF approximations are undefined. After an initial phase, when the instance is propagated to all nodes, the approximation errors gradually decrease. Since some nodes leave the system before their f_i values are disseminated and averaged, the approximation error at interpolation points does not converge to zero. However, the obtained accuracy is in the order of 0.01%, and is clearly sufficient to approximate the CDF through interpolation.

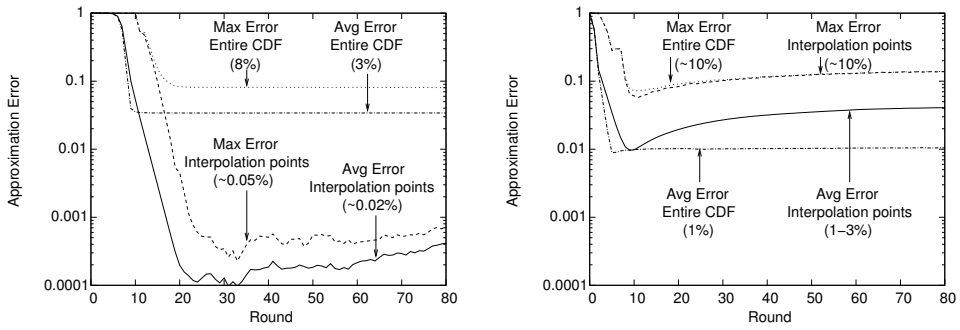
Figure 12(b) shows the approximation error produced by an EquiDepth phase in the same system setup. EquiDepth is not significantly affected by churn, but as previously, it only reduces Err_m to 10% and Err_a to 1%, even at the selected histogram bins.

Figure 13 shows the Err_m and Err_a errors incurred by Adam2 and EquiDepth after 8 protocol instances (phases). In this experiment, joining nodes are included in the evaluation metrics since they receive initial CDF approximations – generated in the previous aggregation instances – from their neighbours. Joining nodes ignore aggregation instances (phases in EquiDepth) that have started before they entered the system in order not to distort the results from already running aggregation instances.

Both systems show a very high resilience to churn, where accuracy starts to significantly decrease only at rates of 1% nodes per gossip round (i.e., 1% per second). This rate is 10 times higher than the rates observed in [13].

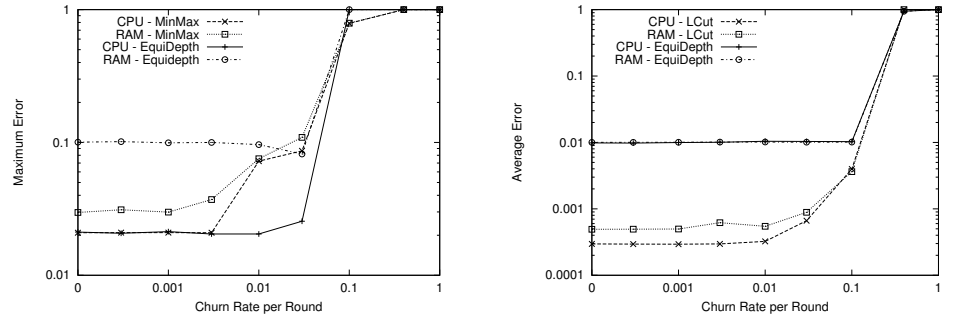
H. Confidence Estimation

As described in section VI, Adam2 allows nodes to assess the accuracy of their own CDF approximations. We evaluate the accuracy estimation algorithms by computing the average difference between the nodes' assessment of an error metric and the actual value for that error metric. Given the true CDF approximation accuracy $Err_a(p)$ at node p, and p's



(a) Adam2 (b) EquiDepth

Figure 12. Approximation accuracy in the presence of churn, for a single instance (RAM).



(a) Maximum error (Err_m) (b) Average error (Err_a)

Figure 13. Impact of churn rate on approximation accuracy.

own estimation of its accuracy $EstErr_a(p)$, we define the error in accuracy estimation at node p as:

$$\frac{|Err_a(p) - EstErr_a(p)|}{Err_a(p)}$$

Similarly, the error in $Err_m(p)$ at node p is defined as:

$$\frac{|Err_m(p) - EstErr_m(p)|}{Err_m(p)}$$

Figure 14 shows the accuracy estimation results for these two metrics. Using 20 verification points, nodes can estimate their own average approximation accuracy with a 10% error. This adds 40% traffic overhead to our CDF approximation algorithm. As expected, more verification points are needed to obtain an accurate estimation of Err_m. However, the experiment shows that even this difficult metric can be roughly estimated using Adam2.

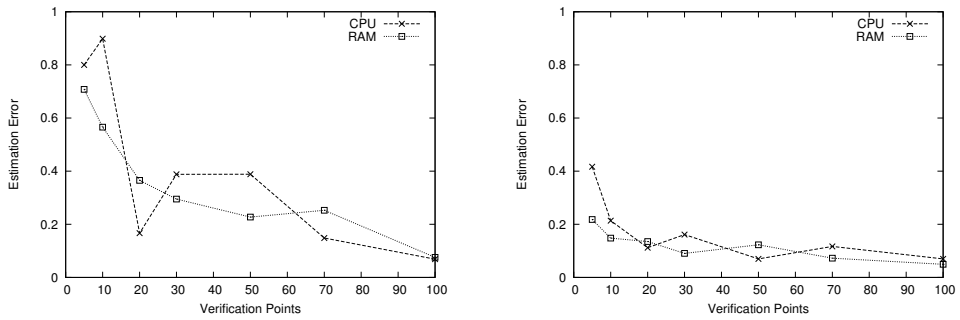
I. Cost Evaluation

An important objective for any monitoring algorithm is to minimise communication costs. The network traffic exchanged by a node in Adam2 is proportional to the number of interpolation points (λ) and the number of gossip rounds. For $\lambda = 50$ the size of a gossip message is approximately 800 bytes. At each round, a node starts exactly one gossip exchange with a randomly chosen neighbour and is, on average, contacted by one other node. Each gossip exchange requires sending and receiving one message, resulting in 2 messages sent and 2 messages received on average every round. Therefore, for one instance with $\lambda = 50$ and 25 rounds, each peer will send, on average, about 40 kB of data (50 messages), and receive another 40 kB. Since three aggregation instances are sufficient for MinMax and LCut to converge, an accurate CDF approximation can be obtained by sending 120 kB of data (150 messages) per node. This cost does not depend on the system size.

The time required to generate a CDF estimation depends on the gossip periodicity. If we consider a reasonable periodicity of 1 second, then an accurate CDF can be obtained in about 75 seconds (3 instances) using an average upstream bandwidth of about 1.6 kB/s, and a downstream bandwidth with a similar value. The CPU, memory, and topology maintenance costs are negligible.

The costs of EquiDepth are very similar to those of Adam2, as both systems send the same number of messages containing similar information. Although EquiDepth converges faster with the same overhead, our refinement algorithms quickly achieve lower error rates.

In random sampling, about 1,000 to 10,000 samples must be obtained by a node in a 100,000-node system in order to achieve a CDF approximation accuracy comparable to that



(a) Maximum error (Err_m)

(b) Average error (Err_a)

Figure 14. Accuracy estimation error for MinMax.

of MinMax or LCut. Using random walks [4], this requires generating between 1,000 and 10,000 messages per node – an order of magnitude more compared to Adam2.

VIII. Conclusions

This paper introduced Adam2, an algorithm for efficiently and accurately estimating the statistical distribution of an attribute belonging to nodes in a large-scale distributed system. Adam2 has a low cost on the order of 1.6 kB/s traffic over 75 seconds, and generates approximations within an average error of 0.05% and a maximum error of 2%. Further, the algorithm can estimate its own accuracy, and thanks to its use of gossip techniques, is quite resilient to churn – obtaining roughly the same average error for very high churn rates up to 1% per second.

These results follow the trend of using gossip-based algorithms to efficiently spread information in very large environments. We are confident that this trend will be pursued with other out-of-the-box algorithms for decentralised information aggregation. Future large-scale applications will thus be able to easily implement monitoring and optimisation functions by picking the needed mechanisms from standard libraries, without the need to reinvent the wheel each time.

References

[1] M. Jelasity, A. Montresor, and O. Babaoglu, "Gossip-based aggregation in large dynamic networks," *ACM Transactions on Computer Systems*, vol. 23, no. 3, pp. 219–252, 2005.

[2] I. Eyal, I. Keidar, and R. Rom, "Distributed clustering for robust aggregation in large networks," in *Proc. 5th Workshop on Hot Topics in System Dependability*, 2009.

[3] M. Haridasan and R. van Renesse, "Gossip-based distribution estimation in peer-to-peer networks," in *Proc. 7th Intl. Workshop on Peer-to-Peer Systems*, 2008.

[4] C. Hall and A. Carzaniga, "Uniform sampling for directed P2P networks," in *Proc. Euro-Par*, 2009.

[5] D. P. Anderson and K. Reed, "Celebrating diversity in volunteer computing," in *Proc. HICSS*. IEEE, 2009, pp. 1–8.

[6] S. Nath, P. B. Gibbons, S. Seshan, and Z. R. Anderson, "Synopsis diffusion for robust aggregation in sensor networks," *ACM Transactions on Sensor Networks*, vol. 4, no. 2, 2008.

[7] R. van Renesse, K. P. Birman, and W. Vogels, "Astrolabe: A robust and scalable technology for distributed system monitoring, management, and data mining," *ACM Transactions on Computer Systems*, vol. 21, no. 2, pp. 164–206, May 2003.

[8] A. Montresor, M. Jelasity, and O. Babaoglu, "Decentralized ranking in large-scale overlay networks," in *Proc. Self-Adaptive and Self-Organizing Systems Workshop*, 2008.

[9] M. Jelasity and A.-M. Kermarrec, "Ordered slicing of very large-scale overlay networks," in *Proc. P2P*, 2006.

[10] A. Fernández, V. Gramoli, E. Jiménez, A.-M. Kermarrec, and M. Raynal, "Distributed slicing in dynamic systems," in *Proc. ICDCS*, 2007.

[11] M. Jelasity, S. Voulgaris, R. Guerraoui, A.-M. Kermarrec, and M. van Steen, "Gossip-based peer sampling," *ACM Transactions on Computer Systems*, vol. 25, no. 3, 2007.

[12] M. Jelasity, A. Montresor, G. P. Jesi, and S. Voulgaris, "The Peersim simulator," `http://peersim.sf.net`.

[13] D. Stutzbach and R. Rejaie, "Understanding churn in peer-to-peer networks," in *Proc. Internet Measurement Conf.*, 2006.