Resilient gossip algorithms for collecting online management information in exascale clusters

A. Barak, Z. Drezner, E. Levy, M. Lieber, A. Shiloh

A. Barak, Z. Drezner, E. Levy, M. Lieber, A. Shiloh,
Resilient gossip algorithms for collecting online management information in exascale clusters,
Concurrency and Computation: Practice and Experience, Vol. 27, Number 17, pp. 4797-4818, 2015,

which has been published in final form at http://dx.doi.org/10.1002/cpe.3465.

Publication History:

- Issue published online: 27 Oct 2015
- Article first published online: 27 Jan 2015
- Manuscript Accepted: 7 Dec 2014
- Manuscript Revised: 3 Dec 2014
- Manuscript Received: 21 Jul 2014 (this pre-peer reviewed version)

Dual-layer gossip algorithms for online management of exascale clusters

Amnon Barak[*], Zvi Drezner[†], Ely Levy[*], Matthias Lieber[‡] and Amnon Shiloh[*]

SUMMARY

Management of forthcoming exascale clusters requires frequent collection and sharing of run-time information about the health of the nodes, their resources and the running applications. This paper presents a new paradigm for online management of scalable clusters, consisting of many nodes and a relatively small number of servers that manage these nodes. The paper presents the details of gossip algorithms for sharing local information within subsets of nodes and for sending selected global information to a master server, which in turn holds up-to-date information on all the nodes. The presented algorithms are decentralized, scalable and can work well even when some nodes are down. The paper gives formal expressions for approximating the average age of the local information at each node and the average age of the collected information at the master. It then shows that the results of these approximations closely match the results of simulations and measurements on a real cluster of the above averages for different-sized subsets, thus pushing ahead towards an exascale cluster with a million compute nodes. The main outcome of this paper is that partitioning of large clusters can reduce the number of messages required for providing local and global information that is no older than a desired average.

KEY WORDS: cluster management; exascale clusters; gossip algorithms; resource management

1 INTRODUCTION

Management of forthcoming extreme-scale clusters with millions of nodes, rely on the collection of run-time information about the health of the nodes, the state of their resources and the demands of their currently-running applications. This may include node availability, load, free memory, temperature and communication-patterns. Due to scalability limitations as well as the volume and diverse use of the collected data for different purposes, we distinguish between two types of collected run-time information: *global* and *local*. Global information is used for system-wide management and monitoring, for example, node-availability for allocation of large jobs or total-usage-per-user for billing. Local information is required for online management of resources within a limited subset of nodes, for example, current-load for load-balancing or temperature for migrations.

[*]Department of Computer Science, The Hebrew University of Jerusalem, Jerusalem 91904, Israel.

[†]College of Business and Economics, California State University, Fullerton, CA 92834, USA.

[‡]Center for Information Services and High Performance Computing, Technische Universität Dresden, 01062 Dresden, Germany.

We assume that while local information requires frequent updates, global information does not need to be collected too frequently - information that is a few seconds old is quite acceptable. To support scalability, passing global information requires efficient communication and volume reduction, so only necessary data is passed. This is often done by tree-based protocols such as MRNet [1], a Tree-Based Overlay Network (TBON) software that supports scalable communication between nodes, but can also be done by gossip-based protocols such as CYCLON [2], which uses an overlay network and randomness to disseminate information across a large set of nodes. In order to allow the management system to quickly respond or initiate corrective measures, local information requires the collection of fresh and detailed information on each resource and the running applications. The outcome is a large volume of data that can create limit scalability. The main goal of this paper is to show that scalability can be achieved by separating the management of global and local information, thus significantly reducing the overall number and size of network-messages.

This paper presents a new paradigm for online management of scalable clusters, consisting of a large number of compute *nodes* and a relatively small number of *servers* that manage global information, thus advancing towards an exascale cluster with a million compute nodes. The compute nodes are divided into disjoint subsets, called *colonies*, based on a common goal or optimization criteria, such as network proximity. The colonies and the servers communicate along a modified TBON topology, where each colony is represented as a leaf by a parent server. A parent server may represent several colonies and the root server is called the *master*.

Within each colony, fresh information is shared in a decentralized manner, using a randomized gossip algorithm similar to the algorithm in [3], which routinely provides each node with information (both local and global) about relevant resources in all the other nodes in its colony. This kind of gossip algorithms is similar to a distributed bulletin board, in which every node has sufficiently accurate view of all the nodes in its colony. The amount of data circulated is reduced by sending *windows* that contain only recent information. The local part of the information is used for internal resource management within the colony, while global data is passed to the colony's parent server.

Transport of global information through the TBON is then accomplished by sending all the relevant data up the tree until all the global information is gathered at the master. A simple example is an Exascale cluster with 1 Mega ($\approx$ million) nodes. The cluster could be divided into 1,024 colonies, each with 1,024 nodes, so that if each colony sends one message to the master then it receives $1,024$ messages each unit of time.

The rationale behind the idea to use colonies is that much of the management can be done at a local level. Since the information required for local management needs to be passed between related nodes anyway, it can also be used, with little or no additional cost, for global management such as monitoring and/or job assignments. The number of messages required for global management is thus a small fraction of the total number of messages.

The paper presents the details of two algorithms that use colonies and shows the tradeoff between the colony size, the number of messages sent to the parent server and the resulting average age of the information, both in the colony nodes and in the parent server. We chose the average age because it better reflects the "freshness" of the information vs. other measures such as the maximal age. To simplify the analysis, it is assumed that the tree has only two layers, i.e., the

parent server and the master are the same. It is also assumed that all the colonies are of the same size and that all colonies and the master use the same time-unit, hence called the gossip-interval.

1.1 Organization of the paper

The paper is organized as follows: the next section presents two gossip-based information dissemination algorithms for updating the master by the colonies. Section 3 presents a comparison between analytical approximations, simulations and measurements on a real cluster of the average window size and the average age of information within colonies of different number of nodes. Section 4 presents analytical expressions for approximating the average age in the master. Section 5 presents a comparison between the results of approximations, simulations and measurements of the master's average age. Section 6 examines how the information-age is effected when some nodes are down. Section 7 describes practical considerations when using the results in large clusters. Related work is described in Section 8 and our conclusions are given in Section 9.

2 PUSH AND PULL GOSSIP ALGORITHMS

Consider a cluster with a large number of *nodes* that are divided into *colonies* and a *master* that performs system-wide services (such as job allocation and monitoring) that require fresh global information on all the nodes in all the colonies. The division into colonies is based on some optimization criteria, usually network proximity. Within colonies, each node collects information about its resources, knows the identity of all the other nodes in its colony and shares this information with them.

This section presents two, *push* and *pull* gossip algorithms for distributing information among all the nodes in each colony and updating the master with the global parts of this information. In both algorithms, the exchange of information among the colony nodes is based on Algorithm 1 in [3], where each colony-node regularly monitors the state of its resources and maintains an information *vector* with *entries* for all the other nodes in its colony. Each such vector-entry includes the last-known state of the resources in the corresponding node and the age of that information. The nodes of each colony exchange (circulate) information among themselves, where every unit of time, each node sends a *window*, containing all the entries whose known age does not exceeds a threshold value T to another, randomly chosen node in its colony. Note that on average, each colony node receives one such message per unit of time. Also, each unit of time, one (or a few) randomly selected node(s) from each colony send the global part of their vectors to the master. Note that neither the colony nodes nor the master keep older information when newer information is available. The colony nodes and the master are not synchronized, i.e., although all nodes use the same gossip-interval, that interval begins at random points in time for different nodes.

In this paper we assume that the colonies are fixed and disjoint, with the same number of nodes n in each colony; that the gossip-intervals for the master and all colonies are the same; and that the local information that needs to be shared within each colony already includes all the information required by the master.

The difference between the two algorithms is that in the push algorithms, colony nodes initiate the master-updates, while in the pull algorithm, the master requests these updates from colony-

nodes.

Below, the term "updates its vector" involves filling the node's own entry with its current state of resources and setting its age to 0; as well as increasing the age of its remaining vector entries to reflect the time passed since the information arrived. Also, "adjusting for network latency" means adding the time taken for the last message to arrive, if known, to all its entries.

2.1 The "push" gossip algorithm

In the following algorithm, colony nodes share information and push (send) the relevant parts of their vectors to the master:

Push algorithm - master receives vectors from each colony:

At a fixed point every unit of time, **each** *colony node:*

- Updates its *vector* and immediately sends a *window* with all its *vector* entries whose current age does not exceed T to another node in its colony, chosen randomly with a uniform distribution.

When *a colony node receives a* window *it:*

- Adjusts the window for network latency.
- Replaces each *vector* entry with the corresponding *window* entry, if the latter is newer.
- Registers the arrival time in the vector entries that were replaced, using its local clock.
- With probability $\frac{k}{n}$ (k is the intended average update rate), updates its vector and sends its global parts to the master.

When *the master receives a* vector *it:*

- Adjusts the vector for network latency.
- Registers the *vector's* arrival time on all the received entries using its local clock.
- Updates all its entries with the latest received vector entries, if the latter is newer.

2.2 The "pull" gossip algorithm

In this algorithm, colony nodes share information while the master regularly pulls (requests) information from one or a few randomly selected nodes in each colony:

Pull algorithm - master requests vectors from each colony:

At a fixed point every unit of time, **each** *colony node:*

- Updates its *vector* and immediately sends a *window* with all its *vector* entries whose current age does not exceed T to another node in its colony, chosen randomly with a uniform distribution.

When *a colony node receives a* window *it:*

- Adjusts the window for network latency.
- Replaces each *vector* entry with the corresponding *window* entry, if the latter is newer.
- Registers the arrival time in the vector entries that were replaced, using its local clock.

At the end of its unit of time, the **master**:

- Increments the age of all entries by one unit of time.
- Requests k *vectors* (k is a constant update rate), from k different and randomly selected nodes in each colony.
- Adjusts each received vector for network latency.
- Updates all its entries with the latest received vectors entries, if newer.

When *a colony node receives a request from the master it:*

- Updates its vector and sends its global parts to the master.

Note that the "pull" algorithm provides the master with a regular and complete information about the colony nodes, but it requires additional messages compared to the "push" algorithm.

3 THE AVERAGE WINDOW SIZE AND VECTOR AGE

This section presents analytical expressions for approximating the colonies average window size sent by the above algorithms and the resulting average vector age. It then shows a comparison between the results obtained by these analytical approximations for different-sized colonies and values of T, with the corresponding results obtained by both simulations of the algorithms, and running these algorithms on a real cluster.

Note that since both the "push" and the "pull" algorithms include an identical first step for circulating windows among the nodes within each colony, the results presented in this section are valid for both algorithms. Also note that both the analytical expressions and the simulations ignore network delays.

3.1 Approximating the average window size

Consider a colony with n nodes that is running either of the above algorithms in a steady state (once reaching an equilibrium). Assume also that there are no network delays and that the clocks on all nodes are evenly and randomly distributed. Let $W(T)$ be the average window size, that is the average number of nodes whose information age does not exceed T units of time.

Suppose that the time progressed from T to $T + \Delta T$ is small (infinitesimal). The probability that a particular node received a message from another node is ΔT because on average, it receives one message every unit of time.

When a window is received, $W(T)$ of the entries in this particular node already have an age that does not exceed T and thus remain in this set (though their ages may improve). The age of the remaining $n - W(T)$ entries may become less than T if they appear in the received window, which has a probability of $\frac{W(T)}{n}$. This proportion of the remaining $n - W(T)$ entries is added to $W(T)$. The addition to $W(T)$, which is now measured for $T + \Delta T$ because all ages increased by ΔT, is therefore:

$$W(T + \Delta T) - W(T) = \Delta W(T) = [n - W(T)] \frac{W(T)}{n} \Delta T \ .$$

Since $W(0) = 1$ is the number of nodes with age 0, $W(T)$ can be found by solving the differential equation:

$$\frac{dW}{dT} = [n - W] \frac{W}{n} \ . \tag{1}$$

In Appendix A it is shown that the solution of Equation (1) is:

$$W(T) = \frac{ne^T}{n - 1 + e^T} \ . \tag{2}$$

Observe that for large values of n and small values of T, $W(T) \approx e^T$.

3.1.1 The approximations

We used Equation (2) to approximate average window sizes for colonies between 128 to 8,192 nodes and T between $2 - 10$ units of time. For reference, we also include approximations for colonies with 1 Mega (M) and 1 Giga (G) nodes, demonstrating the corresponding results for large colonies or clusters without partitions.

The results for each colony size and value T are shown in the "Approximation" rows in the tables below.

3.1.2 Simulations

We developed a hardware-independent simulator for both the "push" and the "pull" algorithms, measuring various properties such as the average window size and the average age of the vector. As the algorithms take time to reach a steady state, the simulator incorporates an initial statistically-based stabilization phase. Since the simulations include a random element, the simulation results shown in the paper were obtained by averaging 5 simulations, each lasting 100 units of time after

reaching a steady state, except for $T = 2$, where larger variations requires more units of time. The results are shown in the "Simulation" rows of the tables below.

3.1.3 Measurements on a real cluster

To include the impact of the network delay in real clusters, we used a modified version of the above simulator on the IBM Blue Gene/Q system JUQUEEN installed at Jülich Supercomputing Centre, Germany [4]. The system has a total of 28,672 nodes, each with a 16-core PowerPC A2 1.6GHz processor, connected by a 5D torus network, with a duplex, peak bandwidth of 2GB/s per link and a worst-case latency of 2.6μs per message [5].

We ran one gossip process per colony node along with a background, network-intensive application on the remaining 15 cores of each node. To include the message transfer time in the aging of the information, we took advantage of the special, highly synchronized clock of the Blue Gene/Q.

To create a worst-case scenario for network contention between gossip and the background applications, we ran MPI-FFT from the HPC Challenge (HPCC) benchmark suite [6, 7]. This application performs a parallel one-dimensional discrete Fourier transform and heavily strains the Blue Gene/Q's interconnect with large messages.

We used a gossip interval of 125 ms, chosen based on a preliminary test so that on the one hand it does not create network delay for small colonies while on the other hand it makes best use of the capability of the Blue Gene/Q network. The results were averaged in the same manner as the simulations (above) and are shown in the "Measurements" rows of the tables below.

3.1.4 Results

The resulting approximations, simulations and measurements of the average window sizes are shown in Table 1, where the leftmost column lists the colony size and the rightmost five column show the results for different values of T.

From the table it can be seen that there is a good match between the results of the approximations and the simulations. The measurements also fit for almost all cases with $T \leq 6$. However, as both the colony size and T increase, there is a reduction in the measured average window sizes (highlighted in boldface), compared with the approximations and the simulations. Simultaneously, there is a reduction in the background application's performance (measured in GFlops/Sec.), which is shown in boldface in Table 2.

The background application can only be effected by network contentions, while the gossip program can be effected both by network contentions and the increased amount of CPU processing that is required for assembling and disassembling larger messages as the colony size and T increase. This explains why the average window sizes with $2K$ or more nodes and $T = 8, 10$, are significantly lower than the approximation/simulation results in Table 1, despite the lack of a similar indication in Table 2.

Table 1: Average window size.

Colony size (nodes)	Method	Circulating windows not exceeding age				
		2	**4**	**6**	**8**	**10**
128	Approximation	7.04	38.48	97.35	122.77	127.27
	Simulation	7.11	38.41	95.47	121.38	126.17
	Measurement	7.09	38.23	95.11	121.27	126.20
256	Approximation	7.21	45.15	156.85	235.83	253.07
	Simulation	7.24	44.97	152.60	232.68	251.76
	Measurement	7.23	44.89	151.77	232.46	251.69
512	Approximation	7.30	49.42	225.88	437.08	500.39
	Simulation	7.30	49.37	219.22	429.33	497.79
	Measurement	7.29	49.09	217.74	427.76	497.45
1K	Approximation	7.34	51.88	289.61	762.37	978.55
	Simulation	7.34	52.01	283.16	744.13	972.35
	Measurement	7.33	51.71	278.60	734.03	969.43
2K	Approximation	7.37	53.21	337.17	1,214.21	1,873.86
	Emulation	7.35	53.18	331.34	1,179.06	1,855.81
	Measurement	7.33	52.90	325.24	**1,143.09**	**1,836.94**
4K	Approximation	7.38	53.89	367.34	1,725.56	3,453.88
	Simulation	7.34	53.80	362.20	1,680.88	3,409.99
	Measurement	7.34	53.48	354.50	**1,574.42**	**3,292.48**
8K	Approximation	7.38	54.24	384.54	2,185.83	5,971.41
	Simulation	7.31	53.64	377.27	2,123.41	5,889.18
	Measurement	7.32	53.57	368.99	**1,927.81**	**5,323.18**
1M	Approximation	7.39	54.60	403.27	2,972.51	21,573.32
1G	Approximation	7.39	54.60	403.43	2,980.95	22,026.01

Table 2: MPI-FFT performance (GFlops/Second).

Colony size (nodes)	Circulating windows not exceeding age				
	2	**4**	**6**	**8**	**10**
128	200.40	200.42	200.32	200.26	200.32
256	368.80	367.84	367.64	367.78	367.58
512	663.00	662.30	662.10	660.40	661.40
1K	1,129.80	1,126.72	1,127.02	1,126.58	1,126.82
2K	1,650.50	1,648.82	1,650.18	1,647.26	1,647.82
4K	2,334.30	2,340.10	2,342.80	**2,321.20**	**2,308.90**
8K	3,330.70	3,327.64	3,328.72	**3,316.52**	**3,286.74**

3.2 Average vector age

To approximate the average age of the vectors when colonies circulate windows with entries not exceeding age T, first, we calculate the age distribution of the colony vector for ages $t > T$.

Let $W(T)$ be the number of colony nodes whose age does not exceed T and $X(t)$ be the number of colony nodes whose age does not exceed t. For $t \leq T$, $X(t) = W(t)$. For $t > T$ the age distribution is computed as followed:

Assume an infinitesimal change in time from t to $t + \Delta t$. The probability that a message arrived during this period is Δt. The probability that a node out of the $n - X(t)$ whose age is greater than t, will obtain a lower age following the message is $(n - X(t))\frac{W(T)}{n}$, because a message contains only nodes whose age does not exceed T in the sending node. Consequently, $X(t + \Delta t) = X(t) + (n - X(t))\frac{W(T)}{n}\Delta t$. Therefore, $\frac{dX(t)}{dt} = (n - X(t))\frac{W(T)}{n}$. The solution of this differential equation is $X(t) = n - Ce^{-\frac{W(T)}{n}t}$ for some constant C. Since $X(T) = W(T)$, it follows that $C = (n - W(T))e^{\frac{TW(T)}{n}}$. Therefore, for $t \geq T$, $X(t) = n - (n - W(T))e^{-\frac{W(T)}{n}(t-T)}$.

To summarize:

$$X(t) = \left\{ \begin{array}{l|l} t \leq T & W(t) \\ t \geq T & n - (n - W(T))e^{-\frac{W(T)}{n}(t-T)} \end{array} \right. . \quad (3)$$

Note that for $t = T$ both expressions are equivalent.

Let $A_w(T)$ denote the average age of the window with all the entries not exceeding T. Then in Appendix B it is shown that:

$$A_w(T) = \frac{n \ln(W(T)) - T(n - W(T))}{W(T) - 1} \quad . \quad (4)$$

Let $A_g(T)$ denote the average of all the vector entries whose age is greater than T and let $A_v(T)$ denote the average age of the whole vector.

By Equation (3):

$$\frac{dX(t)}{dt} = \frac{W(T)}{n}(n - W(T))e^{-\frac{W(T)}{n}(t-T)} .$$

By algebraic manipulations, we get:

$$A_g(T) = \frac{\int_T^\infty t\frac{dX(t)}{dt}dt}{\int_T^\infty \frac{dX(t)}{dt}dt} = \frac{\int_T^\infty te^{-\frac{W(T)}{n}t}dt}{\int_T^\infty e^{-\frac{W(T)}{n}t}dt} = T + \frac{n}{W(T)} .$$

To calculate $A_v(T)$, observe that when information not exceeding T is circulated among the nodes, there are on average $W(T)$ nodes in $A_w(T)$ and therefore $n - W(T)$ nodes in $A_g(T)$.

Thus, the average vector age is:

$$A_v(T) = \frac{W(T)A_w(T) + (n - W(T))A_g(T)}{n} = A_w(T) + \left(1 - \frac{W(T)}{n}\right)(A_g(T) - A_w(T)) , \quad (5)$$

where $W(T)$ is defined in Equation (2) and $A_w(T)$ in Equation (4).

Observe that when circulating the whole vector, that is $W(T) = n$, then $A_v(\infty) = A_w(\infty) = \frac{n}{n-1}\ln n$.

3.2.1 Results

As with the average window size, we used Equation (5) to approximate average vector ages. The corresponding results of the simulations and the measurements are shown in the respective rows of Table 3. The rightmost column shows the average age when circulating the whole vector, which is the absolute minimum.

From Table 3 it can be seen that there is a good match between all the corresponding results. The last 2 rows indicates that without partitioning into colonies, using large clusters with over 1M nodes entail large average ages for all values of T, even when circulation the whole vector.

Table 3: Average vector age.

Colony size (nodes)	Method	Circulating windows not exceeding age					Circulating the whole vector
		2	4	6	8	10	
128	Approximation	19.15	6.00	4.93	4.89	4.89	4.89
	Simulation	18.71	5.96	4.91	4.88	4.89	
	Measurement	18.75	5.99	4.93	4.89	4.90	
256	Approximation	36.49	8.49	5.70	5.57	5.57	5.57
	Simulation	36.07	8.51	5.75	5.61	5.60	
	Measurement	36.15	8.52	5.76	5.61	5.61	
512	Approximation	71.15	13.27	6.70	6.26	6.25	6.25
	Simulation	70.84	13.31	6.78	6.31	6.30	
	Measurement	70.97	13.36	6.80	6.33	6.32	
1K	Approximation	140.44	22.69	8.21	6.99	6.94	6.94
	Simulation	140.20	22.64	8.32	7.04	6.99	
	Measurement	140.32	22.75	8.38	7.09	7.04	
2K	Approximation	279.03	41.47	10.90	7.79	7.63	7.63
	Simulation	278.94	41.27	11.06	7.87	7.69	
	Measurement	279.58	41.45	11.17	7.96	7.77	
4K	Approximation	556.20	78.99	16.06	8.83	8.34	8.32
	Simulation	556.67	78.08	16.28	8.92	8.38	
	Measurement	557.23	78.55	16.53	9.12	8.56	
8K	Approximation	1,110.53	154.02	26.26	10.44	9.07	9.01
	Simulation	1,114.07	151.97	26.68	10.59	9.11	
	Measurement	1,114.06	152.22	27.18	11.02	9.45	
1M	Approximation	141,911	19,209	2,605	360	58	13.86
1G	Approximation	145M	19M	2M	360K	48K	20.79

4 ANALYZING THE MASTER

This section presents analytical expressions for approximating the average age of all the master entries using both the "push" and the "pull" algorithms.

4.1 Analyzing the master using the "push" algorithm

Consider a colony that runs the "push" algorithm, circulating windows not exceeding age T and sending relevant parts of their vectors to the master at an average rate of k per unit of time. By Equation (3), the number of colony-nodes whose age does not exceed t is $X(t)$.

Suppose that the time progressed from t to $t+\Delta t$ is small (infinitesimal). The probability that the master receives a vector from a colony node is $k\Delta t$ because it receives on average k vectors every unit of time.

Let $W_M(t)$ be the number of nodes of a certain colony for which the master has information whose age is not greater than t. When a vector is received, $W_M(t)$ nodes have an age which does not exceed t and thus remain in this set (though their ages may improve). The age of the remaining $n-W_M(t)$ entries may become smaller than t if their age in the received vector is less than t. The vector sent to the master by the colony-node arrived just after that node received a window from another colony-node - therefore its age is on average half a unit of time younger. Thus the average number of nodes not exceeding t is $X(t+0.5)$. The probability that a node in the sent vector has an age lower than t is $\frac{X(t+0.5)}{n}$. This proportion of the remaining $n-W_M(t)$ nodes in the master's information is added to $W_M(t)$. Since Δt is infinitesimal, the probability that a message is received during this time period is $k\Delta t$. The addition to $W_M(t)$, which is now measured for $t+\Delta t$, because all ages increased by Δt, is therefore:

$$W_M(t+\Delta t) - W_M(t) = [n - W_M(t)] \frac{X(t+0.5)}{n} k\Delta t \ .$$

We can find $W_M(t)$ by solving the differential equation:

$$\frac{dW_M(t)}{dt} = k\,[n - W_M(t)] \frac{X(t+0.5)}{n} \ . \tag{6}$$

Appendix C shows that the solution of Equation (6) for $t \leq T - 0.5$ is:

$$W_M(t) = n - (n - W_M(0)) \left[\frac{n - 1 + e^{0.5}}{n - 1 + e^{t+0.5}} \right]^k \ ,$$

and for $t \geq T - 0.5$ is:

$$W_M(t) = n - (n - W_M(0)) \left[\frac{n - 1 + e^{0.5}}{n - 1 + e^{T}} \right]^k e^{-k\left\{t+0.5-T+\frac{n-1}{e^T}\left(e^{-\frac{W(T)}{n}(t+0.5-T)}-1\right)\right\}} \ . \tag{7}$$

4.1.1 The average age of all the master entries

In the "push" algorithm, Av_M, the average age of all the master entries is calculated at an arbitrary point each unit of time. A correction is needed to account for the time passed since the last message was received. If one message is received, the time until the calculation of the average is 0.5 units on average. If two messages are received, then it is 1/3 time units on average. In general, when s messages are received, the time lapse from the last message is $\frac{1}{s+1}$. The n cluster-nodes send messages with a probability of $\frac{k}{n}$ every time unit. The probabilities of receiving $1, 2, 3, \ldots$ messages are distributed by a binomial function. Since $\frac{k}{n}$ is small, the probability distribution can be approximated well by a Poisson distribution with an average of $n\frac{k}{n} = k$. The probability that s messages are received in a unit of time is thus $P_s = \frac{k^s}{s!}e^{-k}$. The expected time lapse is:

$$\sum_{s=1}^{\infty} \frac{1}{s+1} \frac{k^s}{s!} e^{-k} = \sum_{s=1}^{\infty} \frac{k^s}{(s+1)!} e^{-k} = e^{-k} \left\{ \frac{e^k - 1 - k}{k} \right\} = \frac{1}{k} - (1 + \frac{1}{k}) e^{-k} \ .$$

Therefore,

$$Av_M = \frac{\int_0^{\infty} t \frac{dW_M(t)}{dt} dt}{\int_0^{\infty} \frac{dW_M(t)}{dt} dt} - \frac{1}{k} + (1 + \frac{1}{k}) e^{-k} \ . \tag{8}$$

The derivative $\frac{dW_M(t)}{dt}$ is calculated in Appendix D.

Note that we also tried the approach of sending only windows of size $W(T)$ to the master. In that case, the average age of all the entries of the master could be found by applying $X(t) = W(T)$ for $t + 0.5 > T$. However, the results were significantly worse than when sending the whole vector to the master.

4.1.2 Limit of the average age

When sending the whole vector to the master, that is when $T \to \infty$, only the case $t + 0.5 \leq T$ applies, because $T = \infty$. Then by using Equation (8), it is shown in Appendix E that:

$$Av_M = \left[\frac{n-1+e^{0.5}}{n-1}\right]^k \ln\frac{n-1+e^{0.5}}{e^{0.5}} - \sum_{j=1}^{k}\frac{\left[\frac{n-1+e^{0.5}}{n-1}\right]^{k-j}}{j} + (1+\frac{1}{k})e^{-k} \ . \tag{9}$$

4.2 Analyzing the master using the "pull" algorithm

In the "pull" algorithm the master requests information from k different and randomly selected nodes in each colony every unit of time. Since the local clocks of the colony nodes are randomized, the outcome does not depend on the master's clock.

In the "push" algorithm the vector has a lower average age because the colony vector is updated, on average, half a time unit earlier than in the "pull" algorithm. Therefore, the probability that a node in the vector has an age lower than t is $\frac{X(t)}{n}$ in the "pull" algorithm rather than $\frac{X(t+0.5)}{n}$ in the "push" algorithm. As a result, $t + 0.5$ in the expressions for the "push" algorithm should be replaced by t and $e^{0.5}$ should be replaced by 1 in the expressions for the "pull" algorithm.

The average age of all the master entries in the "pull" algorithm is calculated by Equation (8), except that, as the requests for information from the nodes are sent simultaneously at the beginning of each time unit and the average is calculated half a time unit later, the expected time lapse is $\frac{1}{2}$ unit of time. Thus, the expression $-\frac{1}{k} + (1+\frac{1}{k})e^{-k}$ in Equation (8) should be replaced by $\frac{1}{2}$ and the following derivatives are used:

For $t \leq T$:

$$\frac{dW_M(t)}{dt} = \frac{k(n-1)e^t}{n-1+e^t}\left[\frac{n}{n-1+e^t}\right]^k \ , \tag{10}$$

and for $t > T$:

$$\begin{aligned}\frac{dW_M(t)}{dt} &= k(n-1)\left[\frac{n}{n-1+e^T}\right]^k e^{-k\left\{t-T+\frac{n-1}{e^T}\left(e^{-\frac{W(T)}{n}(t-T)}-1\right)\right\}} \\ &\quad \left(1-\frac{n-1}{n-1+e^T}e^{-\frac{W(T)}{n}(t-T)}\right) \ .\end{aligned} \tag{11}$$

5 RESULTING MASTER AVERAGE AGE

This section presents comparisons between the results of approximations, simulations and measurements of the average age of all the master entries using the two algorithms.

While for approximations and simulations, the number of colonies makes no difference (due to symmetry), it could affect the results of the measurements due to possible increased network contention. Due to limited resources, we only measured the master's average age using a single colony, relying on the symmetry between the colonies. This ignores possible added network contentions when many colonies report simultaneously to one master. However, we expect that where more computational resources (such as in an exascale cluster) are available, an adequate network will

also be available; or alternately it is possible to drop the assumption that the TBON has only two layers.

5.1 Using the "push" algorithm

For each colony size and value of T, the "Approximation" row in Table 4 shows the average age based on the approximations of $W_M(t)$ obtained by Equations (7) and (8), for $k = 1$, i.e., when each colony sends one vector to the master each unit of time. The second and third rows show the corresponding results of the simulations and the measurements. The rightmost column shows the approximations when circulating whole vectors among the colony nodes using Equation (9), which is the absolute minimum. Observe that there is a fixed difference of ≈ 0.76 units of time between the values in the rightmost column of Table 4 and the corresponding values in the rightmost column of Table 3. This difference can be explained by the fact that the master has the advantage of receiving freshly-updated vectors, whereas the average age of colony-nodes is sampled at random intervals.

Table 4: Average age of the master entries using the "push" algorithm, $k = 1$.

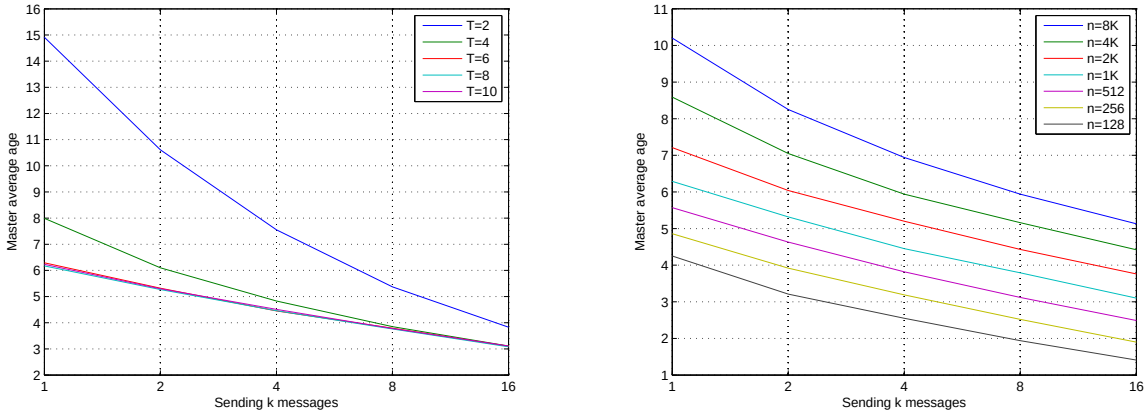
Colony size (nodes)	**Method**	**Circulating windows not exceeding age**					**Circulating the whole vector**
		2	**4**	**6**	**8**	**10**	
128	Approximation	5.83	4.28	4.15	4.15	4.15	4.15
	Simulation	5.75	4.41	4.25	4.13	4.11	
	Measurement	5.66	4.33	4.17	4.19	4.16	
256	Approximation	7.98	5.18	4.83	4.82	4.82	4.82
	Simulation	7.92	5.15	4.86	4.74	4.85	
	Measurement	8.09	5.13	4.98	4.87	4.83	
512	Approximation	11.03	6.36	5.53	5.49	5.49	5.49
	Simulation	11.03	6.44	5.57	5.45	5.52	
	Measurement	11.04	6.30	5.50	5.49	5.54	
1K	Approximation	15.34	7.98	6.32	6.18	6.18	6.18
	Simulation	15.15	7.99	6.29	6.17	6.23	
	Measurement	15.31	7.91	6.39	6.30	6.28	
2K	Approximation	21.45	10.24	7.27	6.88	6.87	6.87
	Simulation	21.48	10.05	7.21	6.95	6.90	
	Measurement	21.46	10.11	7.30	6.96	7.13	
4K	Approximation	30.09	13.43	8.51	7.60	7.56	7.56
	Simulation	30.05	13.06	8.59	7.67	7.53	
	Measurement	29.81	13.48	8.58	7.75	7.68	
8K	Approximation	42.31	17.93	10.20	8.41	8.25	8.25
	Simulation	42.63	17.59	10.20	8.46	8.24	
	Measurement	41.71	17.79	10.27	8.70	8.56	
1M	Approximation	472.70	176.26	68.47	30.08	17.21	13.10
1G	Approximation	1331.40	1320.36	1234.15	753.02	285.29	20.03

The results in Table 4 shows a good match between the approximations, simulations and the measurements.

In the next set of tests we studied the average age of the master when every unit of time it receives on average $k = 1 - 16$ vectors from each colony. The results of these tests show a good match between the approximations and the simulations.

Figure 1 provides insight about the pattern of the master's average age using the simulation results, when the rate k is increased from 1 to 16. Figure 1(a) shows the master's average age for colonies with 1,024 nodes and T between 2 to 10 units of time. From this figure it can be seen that when increasing the value of k, the average age for all values of T, converges towards a constant that can be determined by Equation (9). Figure 1(b) shows for colony sizes between 128 to 8,192 and $T = 6$, how the master's average age is decreased as k increases.

As predicted by Equation (9), when T is large enough and $k \geq 2$, about the same master's average age can be achieved for larger colonies by increasing k at the same ratio as the colony size.



(a) Colony size 1,024 nodes, T = 2 - 10.

(b) All colony sizes, T = 6.

Figure 1: Average age of the master using the "push" algorithm, k=1-16.

This was verified by the simulation results. For example, when circulating the whole vector, the average age for 256 nodes with $k = 2$ was 3.86; while it was 3.84 for 512 nodes with $k = 4$, 1,024 nodes with $k = 8$ and 2,048 nodes with $k = 16$.

5.2 Using the "pull" algorithm

For each colony size and value of T, the "Approximation" row in Table 5 shows the average age of the master entries for the "pull" algorithm with $k = 1$, based on the approximations $W_M(t)$ obtained by Equations (10) and (11). The corresponding simulations and measurement results, taken halfway into the master's unit of time, are shown in the second and third rows respectively.

Table 5: Average age of the master entries using the "pull" algorithm with $k = 1$.

Colony size (nodes)	**Method**	**Circulating windows not exceeding age**					**Circulating the whole vector**
		2	**4**	**6**	**8**	**10**	
128	Approximation	6.06	4.52	4.39	4.39	4.39	4.39
	Simulation	6.23	4.73	4.55	4.61	4.61	
	Measurement	6.25	4.69	4.59	4.65	4.58	
256	Approximation	8.22	5.43	5.08	5.07	5.07	5.07
	Simulation	8.26	5.67	5.27	5.33	5.32	
	Measurement	8.43	5.59	5.37	5.32	5.29	
512	Approximation	11.28	6.62	5.79	5.75	5.75	5.75
	Simulation	11.36	6.85	6.05	6.00	6.00	
	Measurement	11.42	6.82	6.03	6.00	6.02	
1K	Approximation	15.60	8.24	6.58	6.44	6.44	6.44
	Simulation	15.68	8.39	6.88	6.69	6.68	
	Measurement	15.91	8.43	6.84	6.73	6.71	
2K	Approximation	21.71	10.50	7.53	7.14	7.13	7.13
	Simulation	21.92	10.65	7.79	7.33	7.36	
	Measurement	21.95	10.86	7.81	7.39	7.45	
4K	Approximation	30.35	13.69	8.77	7.87	7.82	7.82
	Simulation	30.65	13.94	9.04	8.05	8.06	
	Measurement	30.67	14.02	9.11	8.19	8.18	
8K	Approximation	42.57	18.19	10.46	8.68	8.51	8.51
	Simulation	42.77	18.42	10.79	8.87	8.71	
	Measurement	42.80	18.54	10.78	9.11	9.02	
1M	Approximation	472.97	176.52	68.73	30.34	17.48	13.36
1G	Approximation	1331.33	1320.30	1234.16	753.27	285.55	20.29

Observe that the averages in Table 5 are higher than those in Table 4 because in the "push" algorithm the vector is received by the master right after the sending colony-node was itself updated,

whereas in the "pull" algorithm it was received with an average delay of 0.5 units of time. We also note that while the information is older using the "pull" algorithm, its variance in both simulations and measurements was much smaller than in the "push" algorithm.

6 AVERAGE AGE WHEN SOME NODES ARE DOWN

This section examines the implications on the average age when some nodes are down, e.g. failed. For simplicity, it is assumed that the number of nodes that are down is the same in all the colonies.

Using the "push" algorithm, Table 6 shows the simulation results of the average vector ages for colonies with $n = 1,024$ nodes, of which up to 32 nodes are down.

Table 6: Simulations of the average vector age with failed nodes.

Number of failed nodes	**Circulating windows not exceeding age**				
	2	**4**	**6**	**8**	**10**
0	140.20	22.64	8.32	7.04	6.99
1	140.12	22.87	8.35	7.06	7.00
2	140.50	22.95	8.36	7.06	7.01
4	140.97	23.06	8.38	7.08	7.02
8	142.15	23.42	8.46	7.11	7.04
16	144.44	24.03	8.61	7.16	7.10
32	149.21	25.31	8.92	7.29	7.21

From Table 6 it can be seen that for each T there is a gradual increase in the average age of the vector as more nodes are down. Note that the rate of this increase is larger for smaller windows. For example, for $T = 2$, when the number of nodes that are down is increased to 32, the average age is gradually increased to 6.4%, while the corresponding increase for $T = 10$ is only 3.1%.

The corresponding results for the master's average age are shown in Table 7. Observe that the master's average age in Table 7, is more strongly influenced by the number of nodes that are down than in Table 6. where the colony's average vector age is shown.

Table 7: Simulations of the average master age with failed nodes.

Number of failed nodes	**Circulating windows not exceeding age**				
	2	**4**	**6**	**8**	**10**
0	15.15	7.99	6.29	6.17	6.23
1	14.96	8.01	6.37	6.27	6.20
2	15.32	7.97	6.30	6.30	6.24
4	15.52	8.02	6.34	6.25	6.27
8	15.61	7.98	6.39	6.27	6.44
16	16.24	8.31	6.45	6.25	6.27
32	16.55	8.48	6.63	6.45	6.50

7 FROM THEORY TO PRACTICE

This section describes how the results of the previous sections can be used in large clusters and also provides estimates of various parameters. For simplicity, we only consider the "push" algorithm and a cluster of 1M ($\approx$ million) nodes.

Some gossip parameters are determined by the information needs of the colony's internal management system, such as detecting load imbalances, overheating, memory requirements, process-affinity, etc. These parameters are: entry and vector sizes, the gossip-interval, and the circulated window size.

- **Entry and vector sizes**: Local entries contain collected information about each node, such as the node's speed, number of cores, installed and free memory, load and temperature. A reasonable local entry-size is ≈100 bytes (B). This is slightly larger than the the 64B used by Bohm et al. [8] for monitoring and the 84B used in MOSIX [9]

  Recall that the global information must be a subset of this local information, usually a small subset. Note that the master may also require some static information, such as the number of cores per node: this type of information, however, is best sent once at boot time and should not be part of the gossip activities.

- **Gossip-interval**: The internal management of colonies requires frequent updates. Examples of gossip intervals are *m*emberlist [10], a library that manages cluster membership and failure detection - 200ms; the gossip++ protocol [11] for boosting live streaming - 200ms; Soltero et al. [12], evaluating the use of gossip for power-management in large clusters - 1 Sec.; and MOSIX [9], dynamic resource management and load-balancing in clusters - 1 Sec.

- **Circulated window size**: Within colonies, the window size is only a function of the colony's size and needs. If for example, the colony's management system requires information about all the colony-nodes, of which the average age is below a given value, then Tables 1 and 3 can be use to determine its window size. To illustrate, given a colony of 1,024 nodes with a desired average vector age of ≈ 8.3, then from Table 3 we obtain $T = 6$, then from Table 1 we obtain a window of ≈278 entries.

Another gossip parameter is k, the rate of sending vectors to the master. In most cases, we used $k = 1$, while in Figure 1 we presented results for $k = 1 - 16$. However, since sending large vectors to the master is expensive, we also examined the impact of sending less frequent vectors on the master's average age, using $k = \frac{1}{2},\ \frac{1}{4},\ \frac{1}{8}$, etc. We found that the same trends shown in Figure 1 are also present for $k < 1$, including the nearly-logarithmic reduction shown in Figure 1(b) and the fact that the average is more sensitive to k with smaller values of T, as in Figure 1(a).

The last gossip parameter, the colony size, is constrained by all the above parameters as well as by the network's hardware and software properties. The average amount of data that the master receives is proportional to k. Since the amount of data that the master can handle is limited, this imposes an upper bound on the value of k. The value of T is also limited, this time by the amount of data circulated within colonies. Assuming that the master requires information that is on average no older than a given age Λ and having verified that for each value of k and T, Av_M is a monotonously increasing function of n, we can find a value of n under which $Av_M \leq \Lambda$ and over which $Av_M > \Lambda$. This value of n defines an upper limit for the colony size. Practically, if node failures are expected, this upper limit can be adjusted downwards, so that the master's average will remain below Λ despite a given number of failures.

Other possible constraints on the colony size may occur due to operational requirements within each colony. A lower limit, may be required for better operation of the colony's management such

as allowing a sufficient number of spare nodes to replace faulty or overheated nodes. An upper limit, for example, may occur due to the interference of the intra-colony gossip with networked applications that run in the colony, as can be seen from Table 2 with colony sizes $4K$ and $8K$ with other parameters described in Section 3.1.3.

To give an idea of typical parameters, the 1M-node cluster could be divided into 1,024 colonies, each with 1,024 nodes. Of the 100B local-information entry, 24B could also be global information. In this case, each node needs to maintain a vector of size 100KB for local information while the master needs to maintain 24MB for global information. When circulating windows with 278 entries (for a desired average vector age ≈ 8.3, see above), the intra-colony message size is ≈28KB and the master is sent messages of ≈24KB. Assuming $k = 1$, the master receives on average 24MB per unit of time.

8 RELATED WORK

Distributed randomized gossip algorithms disseminate information through various types of network topologies. Their simplicity, low overhead and fault-tolerance make them attractive for many applications such as membership management in peer-to-peer networks [2, 13, 14], data exchange in linear algebra [15] or computation of aggregate functions [16]. They are also used in sensor networks [17, 18] and for resource management in large clusters [3, 9] and clouds [19].

The MOSIX system [9] combines gossip-based information dissemination and optimization algorithms to provide load balancing in Linux clusters. The MOSIX nodes run daemon processes that exchange with each other information about resources (e.g., available nodes, free memory, CPU load, etc.). This information is used to improve the overall performance of the cluster and individual applications by adaptive allocation and migration of processes among nodes.

Another information passing scheme is MRNet [1, 20], a Tree-Based Overlay Network (TBON) software that supports scalable communication between nodes. MRNet can pass filtered information up and down the tree, which can be processed at each tree node. It has been used for scalable parallel performance-monitoring and profiling of HPC applications [8, 21]. However, this is inefficient when information need to be circulated between nodes of the same tree-level, because it requires messages between two such nodes to pass through a common parent, instead of directly.

Cluster management systems are known to perform well on Linux. The management overhead on large-scale supercomputers is not well understood, as these systems are more susceptible to network contentions. Bhatele et al. [22] identify the contention for shared network resources between jobs as the primary reason for runtime variability of batch jobs in a large Cray system. On Blue Gene systems, however, each job is assigned a private contiguous partition of the torus network, so that contention is reduced.

Menon and Kalé evaluated the performance of GrapevineLB [23], a load balancer that uses gossip algorithms on top of the Charm++ runtime system, on up to 128K cores and showed that the overall performance is improved substantially, but did not discuss the overhead caused by gossip-related messages. Soltero et. al. evaluated the suitability of gossip-based information dissemination for system services of exascale clusters [12]. Their simulations showed that good accuracy can be achieved for power-management services with up to a million nodes. However,

their experiments were emulating only 1000 nodes and did not include measurements of network or the gossip overhead.

9 CONCLUSIONS AND FUTURE WORK

Large scale networked systems, ranging from exascale clusters to mobile devices and Internet of Things are expected to scale up to billions of communicating devices [24] that need to be managed and monitored.

This paper presents a new paradigm for online management of scalable clusters, consisting of many compute nodes and a relatively small number of servers that manage global information about these nodes. The paper presents the details of gossip algorithms for sharing local information within subsets (colonies) of nodes and for sending selected global information to a master server, which in turn holds the latest information on all the nodes of all the colonies. The presented algorithms are decentralized, scalable and can work well even when some nodes are down.

We developed formal expressions for approximating the average age of the local information at each node and the average age of the collected information at the master. We then showed that the results of these approximations closely match the results of simulations and measurements of the above averages in colonies with 128 - 8,192 nodes, thus pushing ahead towards an exascale cluster with a million compute nodes.

The algorithms presented in this paper can be extended to even larger clusters, e.g., with billions of nodes, by using multiple layers of servers that communicate along a TBON topology. In such cases, each server can be configured to receive a fixed number of messages, while each set of servers send their acquired information to the next layer.

In a recent paper [25], it was shown that the steepest decrease in the average age of the vector is obtained when increasing the window size from 10% to 20% of the vector, while larger windows provide only marginal decrease in the average age at the cost of sending increased amounts of data. This warrants further research, investigating fixed window-sizes, perhaps proportional to the colony size.

Similarly, given that sending large vectors to the master is expensive, we considered the possibility of sending only partial information. Specifically, we considered sending only those entries whose age is under a given threshold, beyond which the master does not gain much. Finding optimal thresholds is an interesting subject for further research. Substituting other criteria for information quality, in place of the average age, both within the colonies and in the master, is another subject for further research.

Acknowledgements

The authors wish to thank Carsten Weinhold, Technische Universität Dresden, Germany for his help. The authors gratefully acknowledge the Gauss Centre for Supercomputing (GCS) for providing computing time through the John von Neumann Institute for Computing (NIC) on the GCS share of the supercomputer JUQUEEN at Jülich Supercomputing Centre (JSC). The research presented in this paper is supported by the DFG through the Priority Program 1648 “Software for exascale Computing” (SPPEXA), research project FFMK [26]; by the DFG through the cluster of excellence ‘Center for Advancing Electronics Dresden’ (*cfaed*) and by the EU and the state of

Saxony through the ESF young researcher group IMData.

APPENDIX A: The average window size

In Section 3 it was shown that the average window size $W(T)$ can be found by solving the differential equation:

$$\frac{dW}{dT} = [n - W]\frac{W}{n} \ .$$

Since the right hand side of this equation has only the variable W, it can be solved as follows:

$$\frac{dT}{dW} = \frac{n}{W(n-W)} = \frac{1}{W} + \frac{1}{n-W} \ .$$

By integration,

$$T = \ln W - \ln(n-W) + C = \ln\frac{W}{n-W} + C.$$

Since for $T = 0$ and $W = 1$, $C = \ln(n-1)$, we get:

$$\ln\frac{W}{n-W} = T - \ln(n-1) \ ,$$

implying that:

$$\frac{W}{n-W} = \frac{e^T}{n-1} \ ,$$

and thus,

$$W(T) = \frac{ne^T}{n-1+e^T} \ .$$

An interesting related question is when would all nodes have recent information on all other nodes in their colony. This can be expressed by $W(T)$ approaching n:
Lemma 1: $W(T) > n - n(n-1)e^{-T}$.

Proof: $W(T) = \frac{ne^T}{n-1+e^T} = \frac{n}{1+(n-1)e^{-T}} = \frac{n(1-(n-1)e^{-T})}{1-(n-1)^2e^{-2T}} > n - n(n-1)e^{-T}$. □

Corollary 1: *For* $T > 1.386(10+p)$, *where* $n = 2^p$, $\quad W(T) > n - 10^{-6}$.

APPENDIX B: The average window age

Let $A_w(T)$ denote the average window age, which includes all the entries not exceeding age T. Integration by parts yields:

$$A_w(T) = \frac{\int_0^T t\frac{dW(t)}{dt}dt}{\int_0^T \frac{dW(t)}{dt}dt} = \frac{TW(T) - \int_0^T W(t)dt}{W(T)-1} \ ,$$

where

$$\int_0^T W(t)dt = \int_0^T \frac{ne^t dt}{n-1+e^t} = n\ln\left(n-1+e^t\right)\Big|_0^T = n\ln\frac{n-1+e^T}{n} \ ,$$

and

$$\frac{W(T)-1}{W(T)}A_w(T) = \frac{T\frac{ne^T}{n-1+e^T} - n\ln\frac{n-1+e^T}{n}}{\frac{ne^T}{n-1+e^T}} = T - \frac{n-1+e^T}{e^T}\ln\frac{n-1+e^T}{n} \ .$$

Since $n-1+e^T = \frac{ne^T}{W(T)}$, it follows that the average window age is:

$$A_w(T) = \frac{n\ln(W(T)) - T(n-W(T))}{W(T)-1} \ .$$

APPENDIX C: Finding the value of $W_M(t)$

The solution of Equation (6) is:

$$\begin{aligned}\frac{\frac{dW_M(t)}{dt}}{[n-W_M(t)]} = k\frac{X(t+0.5)}{n} \quad &\rightarrow \quad -\frac{d}{dt}\ln\left[n-W_M(t)\right] = k\frac{X(t+0.5)}{n} \\ &\rightarrow \quad W_M(t) = n - e^{-\frac{k}{n}\int X(t+0.5)dt} \ . \end{aligned} \tag{12}$$

We distinguish between two cases:

1. For $t+0.5 \leq T$:
   By Equation (3), $X(t+0.5) = \frac{ne^{t+0.5}}{n-1+e^{t+0.5}}$ and thus $\int X(t+0.5)dt = n\ln\left[n-1+e^{t+0.5}\right]+C$. This leads to:

$$W_M(t) = n - \frac{C}{\left[n-1+e^{t+0.5}\right]^k} \ .$$

   We find C for a given $W_M(0)$ and get for $t+0.5 \leq T$:

$$W_M(t) = n - (n - W_M(0))\left[\frac{n-1+e^{0.5}}{n-1+e^{t+0.5}}\right]^k \ . \tag{13}$$

2. For $t+0.5 > T$:
   By Equation (3), $X(t+0.5) = n-(n-W(T))e^{-\frac{W(T)}{n}(t+0.5-T)}$. Note that $W(T)$ is a constant.

   $\int X(t+0.5)dt = nt + \frac{n}{W(T)}(n-W(T))e^{-\frac{W(T)}{n}(t+0.5-T)} + C$.

   By Equation (12):

$$\begin{aligned} W_M(t) &= n - e^{-\frac{k}{n}\left\{nt+\frac{n}{W(T)}(n-W(T))e^{-\frac{W(T)}{n}(t+0.5-T)}+C\right\}} \\ &= n - e^{-k\left\{t+\left(\frac{n}{W(T)}-1\right)e^{-\frac{W(T)}{n}(t+0.5-T)}+C\right\}} \\ &= n - e^{-k\left\{t+\frac{n-1}{e^T}e^{-\frac{W(T)}{n}(t+0.5-T)}+C\right\}} \ . \end{aligned} \tag{14}$$

   In order to determine the constant C, we equate Equations (13) and (14) for $t+0.5=T$:

$$(n-W_M(0))\left[\frac{n-1+e^{0.5}}{n-1+e^T}\right]^k = e^{-k\left\{T-0.5+\frac{n-1}{e^T}\right\}}e^{-kC} \ ,$$

   thus,

$$e^{-kC} = (n-W_M(0))\left[\frac{n-1+e^{0.5}}{n-1+e^T}\right]^k e^{k\left\{T-0.5+\frac{n-1}{e^T}\right\}} \ ,$$

   yielding for $t+0.5 \geq T$:

$$W_M(t) = n - (n-W_M(0))\left[\frac{n-1+e^{0.5}}{n-1+e^T}\right]^k e^{-k\left\{t+0.5-T+\frac{n-1}{e^T}\left(e^{-\frac{W(T)}{n}(t+0.5-T)}-1\right)\right\}} \ .$$

APPENDIX D: Calculating the derivative $\frac{dW_M(t)}{dt}$

The derivative $\frac{dW_M(t)}{dt}$ is calculated separately for $t+0.5 \leq T$ and for $t+0.5 > T$.
By algebraic manipulations, for $t+0.5 \leq T$ we get:

$$\frac{dW_M(t)}{dt} = \frac{k(n-W_M(0))e^{t+0.5}}{n-1+e^{t+0.5}} \left[\frac{n-1+e^{0.5}}{n-1+e^{t+0.5}}\right]^k ,$$

while for $t+0.5 > T$, we get:

$$\begin{aligned}\frac{dW_M(t)}{dt} &= k(n-W_M(0)) \left[\frac{n-1+e^{0.5}}{n-1+e^{T}}\right]^k e^{-k\left\{t+0.5-T+\frac{n-1}{e^T}\left(e^{-\frac{W(T)}{n}(t+0.5-T)}-1\right)\right\}} \\ &\left(1-\frac{n-1}{n-1+e^T}e^{-\frac{W(T)}{n}(t+0.5-T)}\right) .\end{aligned}$$

Note that the two derivatives are identical when $t+0.5 = T$. In order to substitute these expressions into Eq. (8) multiplicative constants such as $k(n-W_M(0))$ can be removed because they cancel out.

The graphs for $\frac{dW_M(t)}{dt}$ and $t\frac{dW_M(t)}{dt}$, for $n = 1,024$ and $T = 5, 10$ are depicted in Figure 2. The integrations required to explicitly obtain Av_M by Equation (8) are complicated and may be intractable. As can be seen from the graphs, both functions are "very smooth" and thus numerical integration is very effective.

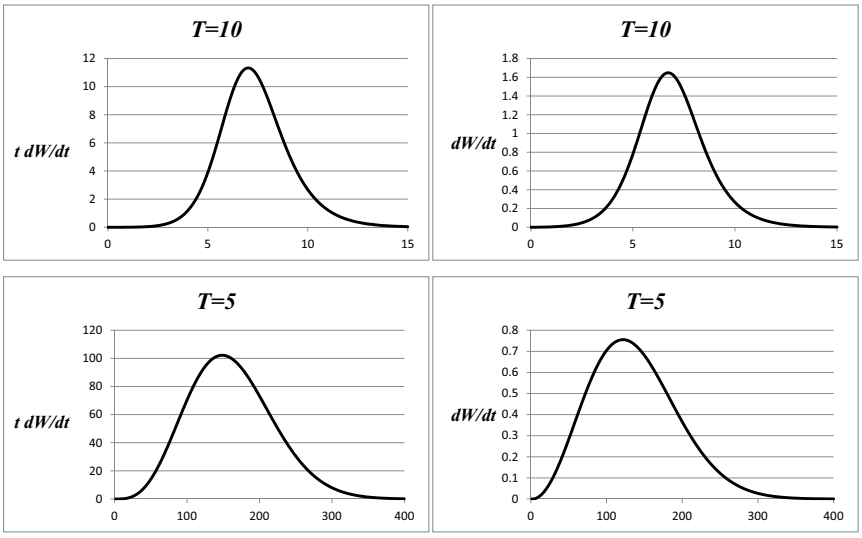


Figure 2: The integrand of Equation (8).

We calculated Av_M by the extended Simpson rule (for example see [27]) of Equation (8) using these expressions for the derivatives. The error in the integral over a segment of length h is $\frac{h^5}{90}f^{(\mathrm{iv})}(\xi)$ where ξ is a point in the interval. Since the fourth derivative is relatively small, judging by the smoothness of the integrand, when doubling the number of segments, the error is expected to be 16 times smaller. We start with the whole range as h and the results stabilize following a few halving of h.

APPENDIX E: Limit of the master's average age

For the "push" algorithm, using Equation (8) with $t + 0.5 \leq T$ we get:

$$Av_M = \frac{\int_0^\infty t\frac{k(n-1)e^{t+0.5}}{n-1+e^{t+0.5}}\left[\frac{n-1+e^{0.5}}{n-1+e^{t+0.5}}\right]^k dt}{\int_0^\infty \frac{k(n-1)e^{t+0.5}}{n-1+e^{t+0.5}}\left[\frac{n-1+e^{0.5}}{n-1+e^{t+0.5}}\right]^k dt} - \frac{1}{k} + (1+\frac{1}{k})e^{-k} \ .$$

The denominator of this equation is $-\frac{k(n-1)\left[n-1+e^{0.5}\right]^k}{k\left[n-1+e^{t+0.5}\right]^k}\Big|_0^\infty = n-1$, thus yielding:

$$\begin{aligned}
Av_M + \frac{1}{k} - (1+\frac{1}{k})e^{-k} &= \int_0^\infty t\frac{ke^{t+0.5}}{n-1+e^{t+0.5}}\left[\frac{n-1+e^{0.5}}{n-1+e^{t+0.5}}\right]^k dt \\
&= k\left[n-1+e^{0.5}\right]^k \int_0^\infty t\frac{e^{t+0.5}}{\left[n-1+e^{t+0.5}\right]^{k+1}}dt \\
&= k\left[n-1+e^{0.5}\right]^k \left\{ -t\frac{1}{k\left[n-1+e^{t+0.5}\right]^k}\Big|_0^\infty + \int_0^\infty \frac{dt}{k\left[n-1+e^{t+0.5}\right]^k} \right\} \\
&= \left[n-1+e^{0.5}\right]^k \int_0^\infty \frac{dt}{\left[n-1+e^{t+0.5}\right]^k} \ .
\end{aligned}$$

Define: $\mu_k = \left[n-1+e^{0.5}\right]^k \int_0^\infty \frac{dt}{\left[n-1+e^{t+0.5}\right]^k}$. Then for $k=1$ we get:

$\mu_1 = \left[n-1+e^{0.5}\right] \int_0^\infty \frac{dt}{n-1+e^{t+0.5}} = \left[n-1+e^{0.5}\right] \int_0^\infty \frac{e^{-(t+0.5)}dt}{1+(n-1)e^{-(t+0.5)}} = \frac{n-1+e^{0.5}}{n-1}\ln\frac{n-1+e^{0.5}}{e^{0.5}}$.

For general values of k we get:

$$\begin{aligned}
\frac{\mu_k}{\left[n-1+e^{0.5}\right]^k} &= \int_0^\infty \frac{dt}{\left[n-1+e^{t+0.5}\right]^k} = \int_0^\infty \frac{(n-1+e^{t+0.5})dt}{\left[n-1+e^{t+0.5}\right]^{k+1}} \\
&= \frac{(n-1)\mu_{k+1}}{\left[n-1+e^{0.5}\right]^{k+1}} + \int_0^\infty \frac{e^{t+0.5}dt}{\left[n-1+e^{t+0.5}\right]^{k+1}} \\
&= \frac{(n-1)\mu_{k+1}}{\left[n-1+e^{0.5}\right]^{k+1}} + \frac{1}{k\left[n-1+e^{0.5}\right]^k} \ .
\end{aligned}$$

Therefore, $\mu_{k+1} = \frac{n-1+e^{0.5}}{n-1}\left(\mu_k - \frac{1}{k}\right)$, yielding: $\mu_k = \left[\frac{n-1+e^{0.5}}{n-1}\right]^{k-1}\mu_1 - \sum_{j=1}^{k-1}\frac{\left[\frac{n-1+e^{0.5}}{n-1}\right]^{k-j}}{j}$, and thus,

$$Av_M = \left[\frac{n-1+e^{0.5}}{n-1}\right]^k \ln\frac{n-1+e^{0.5}}{e^{0.5}} - \sum_{j=1}^{k}\frac{\left[\frac{n-1+e^{0.5}}{n-1}\right]^{k-j}}{j} + (1+\frac{1}{k})e^{-k} \ .$$

References

[1] Roth PC, Arnold DC, Miller BP. MRNet: A software-based multicast/reduction network for scalable tools. SC'03, 2003.

[2] Voulgaris S, Gavidia D, van Steen M. CYCLON: inexpensive membership management for unstructured p2p overlays, J. Network and Systems Management, 13(2): 197–217, 2005.

[3] Amar L, Barak A, Drezner Z, Okun M. Randomized gossip algorithms for maintaining a distributed bulletin board with guaranteed age properties. Concurrency and Computation: Practice & Experience, 21(15): 1907–1927, 2009.

[4] Jülich Supercomputing Center, http://www.fz-juelich.de/jsc/

[5] Chen D, Eisley NA, Heidelberger P, Senger RM, Sugawara Y, Kumar S, Salapura V, Satterfield DL, Steinmacher-Burow B, Parker JJ. The IBM Blue Gene/Q interconnection network and message unit. Proc. SC'11, 2011.

[6] HPC challenge benchmark suite, http://icl.cs.utk.edu/hpcc/

[7] Luszczek, P, Bailey D, Dongarra J, Kepner J, Lucas R. Rabenseifner R, Takahashi D. The HPC Challenge (HPCC) benchmark suite. SC'06 Conference Tutorials, 2006.

[8] Bohm S, Engelmann C, Scott L. Aggregation of real-time system monitoring data for analyzing large-scale parallel and distributed computing environments. Proc. 2010 IEEE 12th Int'l Conference on High Performance Computing and Communications, pp. 72–78, 2010.

[9] Barak A, Shiloh A. The MOSIX cluster operating system for distributed computing on Linux clusters, multi-clusters and clouds, http://www.mosix.org/pub/MOSIX2_wp.pdf

[10] memberlist, https://github.com/hashicorp/memberlist

[11] Frey D, Guerraoui R, Kermarrec AM, Monod M. Boosting Gossip for Live Streaming, 2010 IEEE 10th Int'l Conf. on Peer-to-Peer Computing, pp. 1–10, 2010.

[12] Soltero P, Bridges P, Arnold D, Lang M. A gossip-based approach to exascale system services. Proc. 3rd Int'l Workshop on Runtime and Operating Systems for Supercomputers (ROSS'13), 2013.

[13] Ganesh AJ, Kermarrec AM, Massoulie L. Peer-to-peer membership management for gossip-based protocols. IEEE Tran. on Computers, 52(2):139–149, 2003.

[14] Cuenca-Acuna FM, Peery C, Martin RP, Nguyen TD. Planetp: Using gossiping to build content addressable peer-to-peer information sharing communities. Proc. 12th IEEE Int'l Symp. on High Performance Distributed Computing, pp. 236–246, 2003.

[15] Straková H, Gansterer WN, Zemen T. Distributed QR factorization based on randomized algorithms. Parallel Processing and Applied Mathematics, LCNS Vol. 7203, pp. 235–244. Springer, 2012.

[16] Kempe D, Dobra A, Gehrke J. Gossip-based computation of aggregate information. Proc. 44th Annual IEEE Symp. on Foundations of Computer Science, pp. 482–491, 2003.

[17] Dimakis ADG, Sarwate AD, Wainwright MJ. Geographic gossip: Efficient averaging for sensor networks. IEEE Tran. on Signal Processing, 56(3):1205–1216, 2008.

[18] Kyasanur P, Choudhury RR, Gupta I. Smart gossip: An adaptive gossip-based broadcasting service for sensor networks. IEEE Int'l Conf. on Mobile Adhoc and Sensor Systems (MASS), pp. 91–100, 2006.

[19] Wuhib F, Stadler R, Spreitzer M. A gossip protocol for dynamic resource management in large cloud environments. IEEE Tran. on Network and Service Management, 9(2):213–225, 2012.

[20] http://www.paradyn.org/mrnet/

[21] Nataraj A, Malony AD, Morris A, Arnold DC, Miller BP. A framework for scalable, parallel performance monitoring. Concurrency and Computation: Practice & Experience, 22(6): 720–735, 2010.

[22] Bhatele A, Mohror K, Langer SH, Isaacs KE. There goes the neighborhood: Performance degradation due to nearby jobs. Proc. SC'13, 2013.

[23] Menon H, Kalé L. A distributed dynamic load balancer for iterative applications. Proc. SC'13, 2013.

[24] Schmidt E, Cohen J. The new digital age: Reshaping the future of people, nations and business, A.A. Knopf, New York, 2013.

[25] Levy E, Barak A, Shiloh A, Lieber M, Weinhold C, Härtig H. Overhead of a decentralized gossip algorithm on the performance of HPC applications. Proc. Int'l workshop on Runtime and Operating Systems for Supercomputers (ROSS), 2014.

[26] FFMK Website, http://ffmk.tudos.org

[27] Abramowitz M, Stegun IA. *Handbook of Mathematical Functions With Formulas, Graphs, and Mathematical Tables*, Dover Publications, 1965.