A Case for Epidemic Fault Detection and Group Membership in HPC Storage Systems

Shane Snyder[1(✉)], Philip Carns[1], Jonathan Jenkins[1], Kevin Harms[1], Robert Ross[1], Misbah Mubarak[2], and Christopher Carothers[2]

[1] Argonne National Laboratory, Argonne, IL, USA
{ssnyder,carns,jenkins,rross}@mcs.anl.gov, harms@alcf.anl.gov
[2] Rensselaer Polytechnic Institute, Troy, NY, USA
{mubarm,chrisc}@cs.rpi.edu

Abstract. Fault response strategies are crucial to maintaining performance and availability in HPC storage systems, and the first responsibility of a successful fault response strategy is to detect failures and maintain an accurate view of group membership. This is a nontrivial problem given the unreliable nature of communication networks and other system components. As with many engineering problems, trade-offs must be made to account for the competing goals of fault detection efficiency and accuracy.

Today's production HPC services typically rely on distributed consensus algorithms and heartbeat monitoring for group membership. In this work, we investigate epidemic protocols to determine whether they would be a viable alternative. Epidemic protocols have been proposed in previous work for use in peer-to-peer systems, but they have the potential to increase scalability and decrease fault response time for HPC systems as well. We focus our analysis on the Scalable Weakly-consistent Infection-style Process Group Membership (SWIM) protocol.

We begin by exploring how the semantics of this protocol differ from those of typical HPC group membership protocols, and we discuss how storage systems might need to adapt as a result. We use existing analytical models to choose appropriate SWIM parameters for an HPC use case. We then develop a new, high-resolution parallel discrete event simulation of the protocol to confirm existing analytical models and explore protocol behavior that cannot be readily observed with analytical models. Our preliminary results indicate that the SWIM protocol is a promising alternative for group membership in HPC storage systems, offering rapid convergence, tolerance to transient network failures, and minimal network load.

1 Introduction

As the scale of modern distributed systems continues to grow, so too does the frequency of system component failures. Ensuring efficient and correct behavior in the presence of such failures requires both a reliable fault detection mechanism and a suitable strategy for fault recovery. Example distributed services that rely on efficient and accurate fault detection include distributed storage

S.A. Jarvis et al. (Eds.): PMBS 2014, LNCS 8966, pp. 237–248, 2015.
DOI: 10.1007/978-3-319-17248-4_12

systems [16,21] and reliable multicast protocols [6,7]. Fault detection is one component of broader group membership protocols [6,11,19] that are used to maintain a global view of available participants as they enter or leave the system. An HPC storage system might use this view to determine the set of available servers for data placement, and changes to the group membership can be used to trigger the re-replication of data in order to maintain resilience. An inefficient failure detector (i.e., one that takes too long to disseminate failure notifications to the group) could lead to data loss if data is not re-replicated before additional failures occur, while an inaccurate failure detector could lead to costly, unnecessary rebuilds of the storage system.

Group membership protocols often use heartbeat mechanisms to detect faults [1,8,15,20]: each participant sends out periodic "heartbeat" messages to inform other participants that it is alive. If no new heartbeat messages are received for some prescribed duration, the participant is declared faulty and removed from the group. Unfortunately, the scalability of heartbeat protocols has proven unacceptable for group sizes exceeding more than a few hundred participants [7]. This limitation arises from the network load imposed by group membership protocols in order to provide complete and efficient detection of failures [13]. In practice, failure detector implementations usually divide systems into smaller groups with independent failure domains (introducing artificial limitations on the range of failures the system can tolerate) or delegate group membership maintenance to a specialized subset of participants (increasing engineering complexity and failing to leverage the full network capacity of the system).

In this work, we analyze the efficiency and scalability of the SWIM (Scalable Weakly-consistent Infection-style Process Group Membership) protocol. Previous work [11] proposed the SWIM protocol and evaluated it using both analytical models and a prototype implementation, but to the best of our knowledge it is not used in production on any present-day system. SWIM achieves scalability through the use of a randomized, probe-based failure detection mechanism coupled with an epidemic-style (also known as infection-style or gossip-style) failure dissemination component. As a result, neither the expected network load per participant nor the expected time to first detect a failed participant will depend directly on the size of the group. While much of the analysis given in [11] assumes a distributed peer-to-peer environment, we instead explore how to adapt SWIM to a horizontally scalable data center storage environment as would be used for HPC or big data applications. This environment is characterized by lower network latency and lower churn rate but also higher expectations of consistency and responsiveness. Although failure modes such as silent data corruption are important considerations in HPC storage system design, in this work we focus on total server failures and assume that additional mechanisms will be used to detect silent errors.

The rest of this paper is organized as follows. In Sect. 2, we summarize the SWIM group membership protocol. Section 3 explores the implications of using the SWIM protocol in an HPC storage system, while Sect. 4 analyzes how to tune its parameters for that environment. In Sect. 5 we provide initial simulation

results to confirm its performance and to explore its behavior in lossy network environments. In Sect. 6 we summarize our findings and propose avenues for future research.

2 Background: SWIM

As defined in [11], the SWIM group membership protocol can be functionally decomposed into two primary components: a failure detector and a mechanism for disseminating group membership updates. The failure detection mechanism is based on the periodic probing of random group participants, while the failure dissemination component is implemented by using an epidemic protocol.

To provide a high-level overview of the SWIM failure detector, we outline its operation at an arbitrary participant P_i. The failure detection protocol is governed by two key parameters: protocol period length T' and size of failure detection subgroups k. At the beginning of each of its protocol periods, P_i will select a random participant (which we refer to as P_j) from its local group membership view and probe it using a direct ping request. P_i then waits a prespecified timeout duration to receive an ack from P_j. If no ack is received, the protocol selects k more participants at random and sends an indirect ping request to each of them. Each participant in this subgroup will then ping P_j on behalf of P_i, forwarding any received acks back to P_i to inform that P_j is alive. The indirect ping requests are used to circumvent potential congestion on the network path between P_i and P_j and other phenomena that may have caused the loss of the original direct ping request or response. At the end of the protocol period (of duration T'), if no ack has been received by P_i (whether from direct or indirect probes), then a subprotocol is triggered that marks P_j as *suspected*, and this update is passed to the SWIM dissemination component to be communicated to the rest of the group.

After a participant is declared as *suspected* by the SWIM failure detector, the protocol continues normal operation—the *suspected* participant may still be selected as a probe target in future iterations of the protocol. However, if a participant P_j remains *suspected* for more than s iterations (i.e., the *suspicion timeout*) of the protocol on P_i, then P_i will mark P_j as *failed* and disseminate that information to other participants. If a *suspected* participant becomes responsive again before the suspicion timeout expires, it will be marked as *alive* with a corresponding update disseminated to rejuvenate it in other participants' membership views.

While it seems natural to disseminate membership updates throughout the group by using traditional multicast primitives (e.g., hardware, IP), this approach is unlikely to work at larger scales because of the cost of implementing multicast portably in unreliable networks. For this reason, SWIM disseminates membership information using a gossip-style strategy [20], where information propagates similarly to the way that gossip propagates through society. Compared with typical multicast protocols, gossip-style protocols offer higher efficiency and robustness to failures, although at the cost of a higher dissemination latency. In the SWIM

protocol, group membership updates are disseminated by piggybacking this data on the ping and ack messages already generated by the failure detection protocol. This dissemination therefore introduces no extra packets and imposes minimal additional network load. The information then spreads through the group as participants randomly ping (and ack) each other, ultimately resulting in complete dissemination of the update.

3 Implications of Using SWIM in HPC Storage Systems

Current production HPC storage systems typically use distributed consensus algorithms on subsets of servers to maintain a coherent view of group membership; examples include the Totem single-ring protocol [3] in Corosync [10] (used by a variety of distributed services) and the PAXOS protocol [17] in Ceph [21]. The SWIM protocol semantics differ from such protocols in two notable ways with respect to storage system design. First, SWIM does not provide a strongly consistent view of membership among all participants. At any given time, two participants may have different views of the system. Second, it does not guarantee that updates are disseminated in a consistent order.

SWIM *does* guarantee, however, that all participants will converge to agreement on the state of a failed participant. SWIM also guarantees time-bounded strong completeness when using a randomized round-robin ping strategy [11]. We can therefore calculate both an upper bound and an expected amount of time needed to disseminate a membership update.

Based on these properties, we propose the following design recommendations for fault recovery in storage systems using SWIM for group membership. Note that we leave fault detection and group membership entirely to storage system servers; storage clients are excluded as participants in order to simplify the fault recovery process.

- Avoid the use of fault response protocols that require strict ordering of group updates across servers.
- Allow each server to initiate its own fault response (e.g., generating replicas or recalculating parity) once it has confirmed a fault.
- Validate state agreement between pairs of servers that coordinate during recovery by piggybacking state information on recovery messages. This approach ensures consistency while limiting synchronization overhead.

In general, we observe that the SWIM protocol is not a drop-in replacement for existing fault detection mechanisms in today's storage systems. We will contrast with conventional approaches and explore their impact on storage system design in future work based on the outcome of this preliminary study.

4 SWIM Parameter Selection

Before evaluating SWIM's performance in the context of a large-scale HPC storage system, we must select appropriate protocol parameters. As we vary the

number of storage servers (n), we focus on two key input parameters: the protocol period length (T') and the suspicion timeout in periods (s). These parameters can be used in conjunction with existing analytical models for SWIM to calculate the expected time before a fault is detected by a single server (t_{detect}) as well as the expected time for a given status update to be disseminated to all alive servers (t_{dissem}). We define t_{detect} (derived entirely from analytical models in [13]) as follows, where q_f is the probability that a server is not faulty.

$$t_{detect} = T' \times \frac{1}{1 - e^{-q_f}}$$

We obtain t_{dissem} using the following equation from [11], where x is the number of infected servers (initially 1), n is the group size, and t is time (in protocol periods): $x = \frac{n}{1+(n-1)e^{-(2-\frac{1}{n})t}}$. Then, t_{dissem} may be given as follows, where p_{dissem} is the number of complete protocol periods t from above that results in total dissemination to all alive storage servers.

$$t_{dissem} = T' \times p_{dissem}$$

We further define the total time elapsed from the occurrence of a fault to all servers being aware of the confirmed failure as follows.

$$t_{total} = t_{detect} + (T' \times s) + t_{dissem}$$

We observe the following constraints in order to select SWIM parameters (particularly s and T') that are appropriate for HPC storage systems:

- **Network RTT:** According to the original SWIM protocol definition [11], a participant must wait at least three round-trip times for a ping response from a remote peer. This produces the constraint that $T' > 3 \times RTT$.
- **Network Load:** The minimum value of T' is further bounded by the network capacity of the system. If the period length is too short, then the SWIM network traffic (defined by analytical models in [13] in terms of average number of messages per time unit per participant) may perturb the I/O performance. The acceptable network traffic load threshold depends on the available network capacity.
- **Fault Response Time:** The ultimate value of t_{total} should be complementary to the time needed by the storage system to assess a fault and plan a fault response. Otherwise the fault detection may become a bottleneck to system availability. In this study we propose a goal of $t_{total} \leq 30\,\text{s}$. For comparison, popular moderate-scale group membership implementations such as Pacemaker [5] are often deployed with a 30-s monitoring interval, which does not include time to reach consensus.
- **Transient Failure Sensitivity:** If the suspicion time in seconds ($T' \times s$) is too short, then the protocol may be susceptible to false positives due to congestion or transient network card errors. We propose a goal of $(T' \times s) \geq 10\,\text{s}$, which is long enough to account for the default network driver transmission timeout of 5 s as of Linux 3.15.

Table 1. Effect of SWIM group size, period length, and suspect timeout on expected performance. †

n	T' (ms)	s	Network Load (avg. messages/s per server)	t_{detect} (ms)	Suspicion Time $T' \times s$ (ms)	t_{dissem} (ms)	t_{total} (ms)
1024						.70	15000.86
2048	.10	150000	20976.0	.16	15000.00	.80	15000.96
4096						.90	15001.06
1024						70.00	15085.91
2048	10.00	1500	209.8	15.91	15000.00	80.00	15095.91
4096						90.00	15105.91
1024						7000.00	23591.28
2048	1000.00	15	2.1	1591.28	15000.00	8000.00	24591.28
4096						9000.00	25591.28
2048	200.00	75	10.5	318.26	15000.00	1600.00	16918.26

† Note: The analytical results here are calculated by assuming a 1% chance of a node being faulty, a 1% chance of packet loss, and a subgroup size (k) of 1.

Table 1 shows the impact of protocol period length (T') for storage system sizes (n) ranging from 1,024 to 4,096 servers, given a constant suspicion time ($T' \times s$). Smaller values of T' lead to faster failure detection and dissemination times at the cost of a higher network load. For a given value of T', tolerance to transient failures can be tuned by setting s such that $T' \times s$ is larger than the expected transient failure duration. We can therefore use these parameters to balance performance, network load, and transient failure tolerance.

We selected the example system size (2,048 servers) and parameters ($T' =$ 200 ms, $s = 75$) highlighted in gray for in-depth analysis via parallel discrete event simulation. The period length of 200 ms allows us to detect faults and disseminate notifications rapidly in 318 ms and 1.6 s, respectively. Further, the average network load imposed by this configuration is still negligible compared with the bandwidth of typical network interconnects in high-performance data centers. A suspicion timeout of 75 protocol period lengths allows the protocol to be resilient to transient failures of up to 15 s, depending on how fast subsequent *alive* updates are disseminated to the group. This greatly reduces the probability of unnecessary recovery actions, such as rebuilding storage system data. Note that Das et al. [11] recommend a shorter s value of $(3\lceil \log(n+1) \rceil)$, but we extend it in this context to account for shorter period intervals (T') while still remaining tolerant of transient failures. The total time expected for the protocol to reach global consensus on a failed server is approximately 17 s. This combination of parameters readily meets the constraints described at the beginning of this section. We believe these constraints to be a reasonable starting point for configuring SWIM for use in a large-scale data center, although in-depth characterization of data center failure scenarios could warrant further parameter tuning.

One detail of the SWIM protocol that has been neglected thus far is the number of membership updates to piggyback on each ping and ack message. This piggyback buffer must be large enough to effectively disseminate (potentially numerous) membership updates throughout the system. This requirement

is particularly important in groups where membership is continually changing or in systems with high message loss rates, since the dissemination component may become overwhelmed by the volume of membership updates. However, it is also important to bound the size of this piggyback buffer as part of minimizing the network load imposed by the protocol. For our simulation model, we use a piggyback buffer size of 12, which yields a total message size of 256 bytes if we assume a 64-byte base message and 16 bytes per membership update. This message size in conjunction with the selected configuration parameters in Table 1 produces an expected network consumption of roughly 2.5 KiB/s per server.

5 Simulation Analysis

We developed a parallel discrete event model of the protocol in order to perform an in-depth analysis of an example configuration with the following goals: validation of the analytical model results from Sect. 4 and analysis of the protocol's performance in failure scenarios that are not captured by the analytical model. This simulation will also enable integration with complete storage system models in future work. We constructed our model using the CODES [9,18] storage simulation framework. CODES is built on top of ROSS [4], a high-performance parallel discrete event simulator capable of processing billions of events per second. To our knowledge this is the first discrete event simulation of the SWIM protocol.

Our simulator uses a LogGP network model [2] to calculate network delays. The model assumes full-duplex network cards with independent send and receive queues and infinite buffering in the switch complex. The parameters for our LogGP model were obtained by using the `netgauge` utility [14] on the Tukey Linux cluster at the Argonne Leadership Computing Facility. Each Tukey node uses a single-port Mellanox ConnectX 2 QDR InfiniBand NIC. The `netgauge` utility assumes that the overhead parameter (o) (representing the CPU time consumed during transmission) overlaps with network fabric transmission costs, so we do not apply the o parameter to the communication time calculation. We also take advantage of the fact that `netgauge` calculates LogGP parameters independently for a range of message sizes, creating a lookup table to reflect varying protocol characteristics.

Figure 1 compares the empirically measured point-to-point bandwidth on the Linux cluster (measured by using `mpptest` [12]) with a simulation of the point-to-point performance using our simulation framework. We see that the simulated performance closely matches the performance trends on the example system, including protocol crossover points.

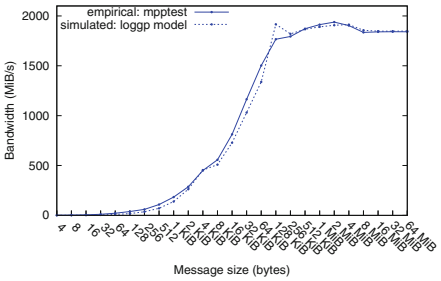

Fig. 1. Point-to-point empirical and simulated bandwidth on QDR InfiniBand network with MPI.

5.1 Sensitivity to Message Loss

We executed a collection of 30-min., 2,048-server simulations of the SWIM protocol in order to evaluate the protocol's sensitivity to message loss. The simulation was configured such that no server failed completely, but the probability of packet loss was varied between 0.2 %, 1 %, and 5 %. A 5 % message loss rate would be an extraordinary occurrence in a data center environment, but we include it as a demonstration of SWIM behavior in extreme conditions. Figure 2 illustrates several performance and accuracy metrics as the SWIM subgroup size k is varied from 1 to 6. The first two figures are accuracy metrics: the number of false positives (i.e., the number of servers falsely confirmed as failed) and the number of servers falsely suspected as failed. The last two are performance metrics: the message rate for each server and the average number of membership updates piggybacked on each message. We gathered the performance metrics from the beginning of the simulation (the first 15 to 30 s) before any false positives were generated that would reduce the number of alive servers.

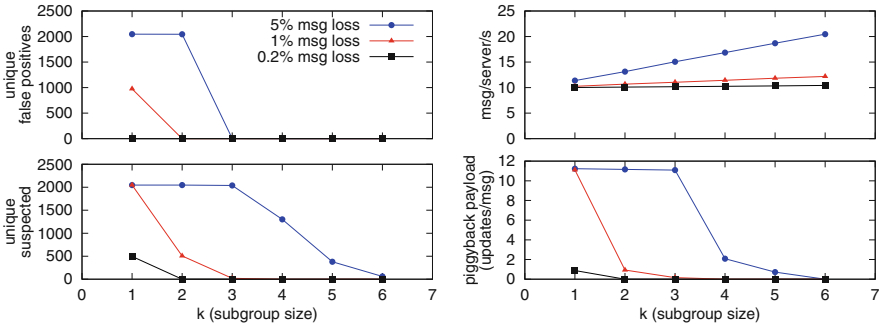

Fig. 2. Accuracy and network load metrics over a 30 min interval with 2,048 servers, a piggyback buffer size of 12, and varying message loss rates.

Protocol accuracy is particularly poor at k values of 1 or 2 for high message loss rates. With a 5 % message loss rate almost all servers are falsely confirmed as failed. In this configuration, the probability of failed direct and indirect pings generates a large volume of failure suspicions that overwhelms the capacity of the dissemination component to correct them. With a subgroup size of 3 there is only a single false positive at a message loss rate of 5 %, although we still observe that nearly all servers are suspected at some point. The piggyback buffer size is near capacity (as evidenced by the piggyback/message metric) but the increased subgroup size allows the dissemination component to more effectively propagate membership updates. We further observe that the protocol can easily manage the 1 % message loss rate at this subgroup size. At a subgroup size of 4, the number of suspect servers declines and the protocol no longer produces any false positives at any message loss rate. With subgroup sizes of 5 or 6 the number of suspected servers diminishes because of the decreased likelihood of all direct and indirect

pings failing for a given target. We observe that the network load imposed by the protocol (measured in the average number of messages per server per second) scales linearly with k, while the accuracy of the protocol increases exponentially with k.

5.2 Validation

Based on our findings from the previous section, we set $k = 6$ to make the protocol more robust against message loss. Using this configuration, in conjunction with the parameters derived in Sect. 4, we performed a set of simulation experiments to measure the response time to single server failures (with no message loss) as we varied the storage system size. We configured our simulation to choose a random server to fail at a random time. We also configured each server in the model to begin its period at a random point within the first T' seconds of the simulation, in order to prevent the SWIM algorithm from producing synchronized bursts of ping traffic. Figure 3 compares the performance measured by simulation with expected values based on the existing analytical models. To be concise, we consider only the time taken to detect a fault (t_{detect}) and the time to disseminate updates to all servers (t_{dissem}). The overall time from fault occurrence to global convergence (t_{total}) is dominated by the suspicion time $T' \times s$, which is a fixed value. We observe that, on average, t_{detect} remains roughly constant regardless of scale and tracks closely with the expected detection time calculated by using the analytical models. In some instances, however, the measured time to detect a server failure is significantly slower (multiple protocol period lengths) than expected. The analytical model includes simplifying assumptions (e.g., it assumes immediate delivery of all pings and acks) that we believe accounts for some of this deviation. The time taken to first detect a failure also depends on when the failure occurs relative to the start of the next protocol period. The t_{dissem} shown in Fig. 3 exhibits $O(\log(n))$ scaling as expected but is consistently faster than predicted by the analytical model. One factor contributing to this discrepancy is that we deliberately desynchronized the

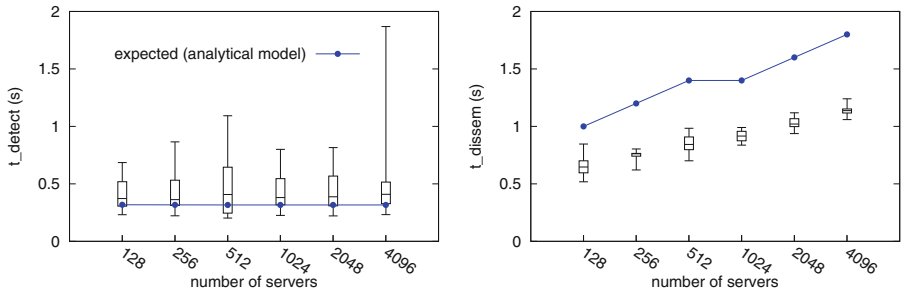

Fig. 3. t_{detect} and t_{dissem} as measured via simulation using highlighted parameters from Table 1. Each box plot shows the minimum, median, maximum, and Q1 and Q3 quartiles for 15 simulation samples. The predicted values according to analytical models are also shown for comparison.

period start times for each server by a random amount, meaning that it typically does not take a full $T' = 200$ ms for a given update to be relayed between two servers. We are also using a more efficient round-robin probing strategy (as suggested by Das et al. [11]) that is not accounted for in the analytical dissemination time calculation. This round-robin probing causes wider dispersal on average than does purely random selection of ping targets in each interval.

These results confirm that both t_{detect} and t_{dissem} are relatively minor components of performance and that they scale well with system size. The largest factor influencing overall performance will be the suspicion timeout s. This SWIM configuration with 4,096 servers would reliably propagate fault notifications to all servers in roughly 17 s while still remaining resilient to transient faults of up to 15 s and imposing negligible network load (still about 2.5 KiB/s, since the load does not scale with the group size). In addition, the constraints from Sect. 4 could readily be modified to accommodate other use cases.

6 Conclusion

In this work we explored the feasibility of adapting peer-to-peer style epidemic fault detection and group membership protocols for use in large-scale HPC storage systems. We identified a set of characteristics necessary for using eventually consistent group membership protocols such as SWIM in HPC storage systems. We used a combination of analytical models and simulation to select appropriate SWIM parameters for an HPC environment while still being tolerant of extraordinary message loss rates. We also studied the SWIM protocol response time as we varied the number of storage servers from 128 to 4,096, and we confirmed that the protocol scales well for basic failure cases. We found that the SWIM protocol could be configured to detect and fully disseminate failure notifications in an exemplar 4,096-server storage system in roughly 17 s, while remaining resilient to transient failures of up to 15 s and imposing a negligible network load. These results suggest that the SWIM protocol is a promising solution for fault detection and group membership in future HPC storage architectures.

We intend to analyze more complex, statistically generated failure scenarios across extended time spans in future work. We also plan to develop models for more traditional group membership protocols, such as those based on the PAXOS family of distributed consensus protocols, in order to perform head-to-head comparisons.

Acknowledgments. This research was supported by the U.S. Department of Defense. This material also was based on work supported by the U.S. Department of Energy, Office of Science, Advanced Scientific Computer Research Program under contract DE-AC02-06CH11357. The research used resources of the Argonne Leadership Computing Facility at Argonne National Laboratory, which is a DOE Office of Science User Facility.

References

1. Aguilera, M.K., Chen, W., Toueg, S.: Heartbeat: A timeout-free failure detector for quiescent reliable communication. In: Mavronicolas, Marios (ed.) WDAG 1997. LNCS, vol. 1320, pp. 126–140. Springer, Heidelberg (1997)
2. Alexandrov, A., Ionescu, M.F., Schauser, K.E., Scheiman, C.: LogGP: Incorporating long messages into the LogP model – one step closer towards a realistic model for parallel computation. In: Proceedings of the Seventh Annual ACM Symposium on Parallel Algorithms and Architectures, SPAA 1995, pp. 95–105. ACM, New York (1995). http://doi.acm.org/10.1145/215399.215427
3. Amir, Y., Moser, L.E., Melliar-Smith, P.M., Agarwal, D.A., Ciarfella, P.: The totem single-ring ordering and membership protocol. ACM Trans. Comput. Syst. **13**(4), 311–342 (1995)
4. Barnes, Jr., P.D., Carothers, C.D., Jefferson, D.R., LaPre, J.M.: Warp speed: Executing time warp on 1,966,080 cores. In: Proceedings of the 2013 ACM SIGSIM Conference on Principles of Advanced Discrete Simulation, SIGSIM-PADS 2013, pp. 327–336. ACM, New York (2013). http://doi.acm.org/10.1145/2486092.2486134
5. Beekhof, A.: Pacemaker: a scalable high availability cluster resource manager. http://clusterlabs.org/. Retrieved July 2014
6. Birman, K.P.: The process group approach to reliable distributed computing. Commun. ACM **36**(12), 37–53 (1993). http://doi.acm.org/10.1145/163298.163303
7. Birman, K.P., Hayden, M., Ozkasap, O., Xiao, Z., Budiu, M., Minsky, Y.: Bimodal multicast. ACM Trans. Comput. Syst. **17**(2), 41–88 (1999). http://doi.acm.org/10.1145/312203.312207
8. Chen, W., Toueg, S., Aguilera, M.K.: On the quality of service of failure detectors. IEEE Trans. Comput. **51**(5), 561–580 (2002)
9. Cope, J., Liu, N., Lang, S., Carns, P., Carothers, C., Ross, R.: Codes: Enabling co-design of multilayer exascale storage architectures. In: Proceedings of the Workshop on Emerging Supercomputing Technologies (2011)
10. Dake, S.C., Caulfield, C., Beekhof, A.: The Corosync cluster engine. In: Linux Symposium, vol. 85 (2008)
11. Das, A., Gupta, I., Motivala, A.: Swim: Scalable weakly-consistent infection-style process group membership protocol. In: Proceedings of the 2002 International Conference on Dependable Systems and Networks, DSN 2002, pp. 303–312. IEEE Computer Society Press, Washington, DC (2002). http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1028914
12. Gropp, W., Lusk, E.: Reproducible measurements of MPI performance characteristics. In: Margalef, T., Dongarra, J., Luque, E. (eds.) PVM/MPI 1999. LNCS, vol. 1697, pp. 11–18. Springer, Heidelberg (1999)
13. Gupta, I., Chandra, T.D., Goldszmidt, G.S.: On scalable and efficient distributed failure detectors. In: Proceedings of the Twentieth Annual ACM Symposium on Principles of Distributed Computing, PODC 2001, pp. 170–179. ACM Press, New York (2001). http://doi.acm.org/10.1145/383962.384010
14. Hoefler, T., Mehlan, T., Lumsdaine, A., Rehm, W.: Netgauge: a network performance measurement framework. In: Perrott, R., Chapman, B.M., Subhlok, J., de Mello, R.F., Yang, L.T. (eds.) HPCC 2007. LNCS, vol. 4782, pp. 659–671. Springer, Heidelberg (2007)

15. Jahanian, F., Fakhouri, S., Rajkumar, R.: Processor group membership protocols: specification, design and implementation. In: Proceedings of the 12th Symposium on Reliable Distributed Systems, pp. 2–11, October 1993
16. Lakshman, A., Malik, P.: Cassandra: A decentralized structured storage system. SIGOPS Operating Sys. Rev. **44**(2), 35–40 (2010). http://doi.acm.org/10.1145/1773912.1773922
17. Lamport, L.: The part-time parliament. ACM Trans. Comput. Syst. (TOCS) **16**(2), 133–169 (1998)
18. Liu, N., Cope, J., Carns, P., Carothers, C., Ross, R., Grider, G., Crume, A., Maltzahn, C.: On the role of burst buffers in leadership-class storage systems. In: 2012 IEEE 28th Symposium on Mass Storage Systems and Technologies (MSST), pp. 1–11. IEEE (2012)
19. Reiter, M.K.: A secure group membership protocol. IEEE Trans. Softw. Eng. **22**(1), 31–42 (1996)
20. van Renesse, R., Minsky, Y., Hayden, M.: A gossip-style failure detection service. In: Davies, N., Jochen, S., Raymond, K. (eds.) Middleware 1998, pp. 55–70. Springer, London (1998). http://dx.doi.org/10.1007/978-1-4471-1283-9_4
21. Weil, S.A., Brandt, S.A., Miller, E.L., Long, D.D.E., Maltzahn, C.: Ceph: A scalable, high-performance distributed file system. In: Proceedings of the 7th Symposium on Operating Systems Design and Implementation, OSDI 2006, pp. 307–320. USENIX Association, Berkeley (2006). http://dl.acm.org/citation.cfm?id=1298485