Probabilistic Self-Stabilization

(Communication)

L. Becchetti[1], A. Clementi[2], E. Natale[1], and F. Pasquale[2]

[1]*Sapienza* Università di Roma, natale@di.uniroma1.it
[2]Università *Tor Vergata* di Roma, clementi@mat.uniroma2.it, pasquale@mat.uniroma2.it

February 19, 2015

Abstract

By using concrete scenarios, we present and discuss a new concept of *probabilistic Self-Stabilization* in *Distributed Systems*.

1 Introduction

A *distributed system* is a network of *agents* that coordinate their actions by exchanging messages [24]. In order to be effective, distributed systems have to be *self-stabilizing* [13]: A system is self-stabilizing if, starting from an arbitrary state, it will quickly reach a *legitimate* state and once in a legitimate state, it will only take on legitimate states thereafter. Self-stabilization has important consequences: for example it allows (fast) recovery when the system is prone to *transient faults* that might take into non-legitimate states [14, 16, 21]. Self-stabilization is a mature subject in the area of Distributed Computing [17] and self-stabilizing algorithms for classical computational tasks are nowadays well-understood.

The original concept of self-stabilization is too restrictive to properly describe some modern systems, e.g., P2P and social networks, which are *dynamic*. Indeed, several relaxations have been proposed so far: *probabilistic self-stabilization* [20], where randomized strategies for self-stabilization are allowed; *pseudo self-stabilization* [12], where the system is allowed to deviate from legitimate states for a finite amount of time; *k-self-stabilization* [5], where restrictions on the initial state are imposed (namely, all allowed initial states are those from which a legitimate state of the system can be reached by changing the state of at most k agents); *weak self-stabilization* [19], that only requires the *existence* of an execution that eventually converges to a legitimate state. However,

> all the above relaxations fail to capture the notion of a system that is self-stabilizing only *with high probability* and that is required to remain in legitimate states only over a *sufficiently long time interval*.

The main goal of this work is to discuss a new *probabilistic* notion of self-stabilization that is general enough to apply to a wide class of complex distributed systems and suitable to derive algorithmic principles that induce effective and useful self-stabilizing behavior in such systems. In general, we say a system is self-stabilizing, according to this revised notion, if

a. From any state it will (quickly) converge to a legitimate state *with high probability (w.h.p.)*, and

b. Once in a legitimate state, w.h.p. it will only take on legitimate states over a sufficiently long time span (for instance, over an arbitrarily-large polyinomial number of steps).

In the next section, we illustrate this new concept in reference to a specific and important scenario, in which probabilistic self-stabilization turned out to be an effective tool of analysis on one hand and naturally led to challenging open questions on the other.

2 Self-Stabilizing Repeated Balls-into-Bins

We consider the following *repeated balls-into-bins* process over an undirected graph $G(V, E)$ with $|V| = n$ nodes. Initially, n balls are assigned to the n nodes in an arbitrary way. Then, in every round, one ball is chosen from each non-empty node according to some strategy (random, FIFO, etc) and re-assigned to one of the node's neighbours uniformly at random. Thus, at every time t each node has a (possibly empty) queue of balls waiting to be forwarded, while every ball performs a sort of *delayed* random walk over the graph, the delay of each random walk depending on the sizes of the queues it encounters along its path. It thus follows that these random walks are correlated. The main issue here is to investigate the impact of such correlation on the maximum load.

Inspired by previous concepts of (load) stability [1, 11], we study the *maximum load* $M^{(t)}$, i.e., the maximum queue size at round t and we are interested in the largest $M^{(t)}$ achieved by the process over a period of (any) *polynomial* length. In the rest of this section we assume G is the complete graph.

Applying the notion of probabilistic self-stabilization. We next discuss an approach that relies on the notion of probabilistic self-stabilization to bound the maximum load, also resulting in tighter bounds to the parallel cover time on the complete graph. In this approach, the state of the process at any time t is completely specified by its *configuration*, specifying the queue size of each node at time t.[1] Our notion of probabilistic self-stabilization discussed above lends itself to a natural characterization of this process, for which it specializes as follows:

Definition 1 (Self-Stabilizing Repeated Balls into Bins.)

- *A configuration is* legitimate *if its maximum load is* $O(\log n)$ *and a process is* stable *if, starting from any legitimate configuration, it only takes on legitimate configurations over a period of* $\text{poly}(n)$ *length, w.h.p.*
- *A process is* self-stabilizing *if it is stable and if, moreover, starting from* any *configuration, it reaches a legitimate configuration, w.h.p.*
- *The* convergence time *of a self-stabilizing process is the maximum number of rounds required to reach a legitimate configuration starting from any configuration.*

It is important to observe that, unlike previous concepts of self-stabilization, here there is always a small chance that the system leaves legitimate states even if no "external" events (e.g. faults) do happen. This natural notion of (probabilistic) self-stabilization was also inspired by the one proposed in [20] for other distributed processes. On the other hand, stability impacts other

[1] Note that, at least to characterize the maximum load, we can assume that balls are indistinguishable.

important aspects of this process. For instance, if the process is stable, we can prove good upper bounds on the *progress* of a ball, namely, the number of rounds in which the ball is selected from its current queue and forwarded, over a sequence of $t \geqslant 1$ rounds. In turn, this provides a useful tool to bound the *parallel* cover time, i.e., the time required for every ball to visit *all* nodes (further details about this issue are given below along this section).

Repeated-balls-into-bins: past work. To the best of our knowledge, the repeated balls-into-bins process was first studied in [10] where it is used as a crucial sub-procedure to optimize the message complexity of a gossip algorithm over the complete graph. The previous analysis in [10, 18] (only) holds over very-short (i.e. logarithmic) periods. On the other hand, analysis in [7] considers periods of arbitrary length, but it (only) yields a bound on the maximum load that rapidly increases with time: after t rounds, the maximum load is $O\big(\sqrt{t}\big)$ w.h.p. By adopting the FIFO strategy at every bin queue, the latter result easily implies that the progress of any ball over t consecutive rounds is $\Omega(\sqrt{t})$ w.h.p. Moreover, it is well known that the cover time for the single-ball process is w.h.p. $\Theta(n \log n)$ (it is in fact equivalent to the *coupon's collector* process [23]). These two facts easily imply an upper bound $O\big(n^2 \log^2 n\big)$ for the parallel cover time of the repeated balls-into-bins process on the complete graph.

In this respect, previous analyses of the maximum load in [7,10,18] are far from tight, since they rely on rough approximations of the process via other, much simpler Markov chains: for instance, in [7], the authors consider the process - which obviously dominates the original one - where, at the beginning of every round, a new ball is added to every empty bin. Clearly, this approach does not exploit a key global invariant (the fixed number n of balls) of the original process. Previous results are thus not helpful to establish whether this process is stable (or, even more, self-stabilizing).

In [6], our group proposed a new, tight analysis of the repeated balls-into-bins process that significantly departs from previous ones, showing that the system is self-stabilizing in the sense of Definition 1. These results are summarized in the following

Theorem 2 ([6]) *Let c be an arbitrarily-large constant, and let the process start from any legitimate configuration. The maximum load $M^{(t)}$ is $O(\log n)$ for all $t = O(n^c)$, w.h.p. Moreover, starting from any configuration, the system reaches a legitimate configuration within $O(n)$ rounds, w.h.p.*

The above result strongly improves over the best previous bounds [7,10,18] and it is almost tight (since we know that maximum load is $\Omega(\log n / \log \log n)$ at least during the first rounds [25]). Moreover, under the FIFO forwarding strategy, the progress of any ball over a sequence of $t = \text{poly}(n)$ consecutive rounds is $\Omega(t/\log n)$ w.h.p. Consequently, the parallel cover time on the complete graph is $O(n \log^2 n)$ w.h.p., which is only a $\log n$ factor away from the lower bound following from the single-ball process.

Wrapping up: balls-into-bins, distributed computing and self-stabilization. As mentioned above, besides being interesting *per-se*, balls-into-bins processes are used to model and analyze several important randomized protocols in parallel and distributed computing [4,9,27]. In particular, the process we study models a natural randomized solution to the problem of *(parallel) resource (or task) assignment* in distributed systems (this problem is also known as *traversal*) [22, 26]. For a more detailed discussion of this potential application, the reader may refer to [6].

Theorem 2 also allows to study the adversarial model in which, on some *faulty* rounds, an adversary can re-assign balls to the bins in an arbitrary way. The self-stabilization property

and the linear convergence time shown in Theorem 2 in fact ensure that the $O(n \log^2 n)$ upper bound on the cover time still holds, provided faulty rounds occur with a frequency no higher than cn, for a sufficiently small constant c.

2.1 Open Questions

Below, we briefly discuss two challenging extensions to the basic problem discussed above, that appear natural in the probabilistic self-stabilization setting we defined.

More balls. Consider the Repeated Balls-into-Bins process with $m > n$ balls and n bins and define a state to be *legitimate* if the maximum load is $O(m/n \cdot \log n)$ (or, even better, $O(m/n + \log n)$). Then, we can pose the same questions: Is the system self-stabilizing? Can we get a good upper bound on the convergence time? Experimental results performed on systems with parameters $m \sim n \log n$ (and increasing values of n up to 1 million) seem to suggest that this might be the case. Also and quite surprsingly, starting from any initial state, the system seems to quickly reach and remain confined to configurations that result in a constant fraction of empty bins. On the one hand, if proved, this property would drastically distinguish the behaviour of this process from the memoryless one in which, in each round, *all* m balls are randomly assigned to the n bins independently of their current positions (i.e., there are no bin queues). Indeed, in the latter process, a well-known result implies that, w.h.p., all bins will be non-empty over any polynomial number of rounds. On the other hand, convergence towards a constant fraction of empty bins is a key-ingredient for proving self-stabilization in the case $m = n$ (i.e. Theorem 2). Thus, we believe that, at least when $m \leqslant n \log n$, the system is still self-stabilizing.

General Graphs. Analyzing the (probabilistic) self-stabilization properties of the repeated-balls-into-bins process turned out to be extremely challenging in networks other than the clique: This "hardness" seems to hold even for restricted classes of graphs. For instance, we tried to extend our analysis to the most symmetric, sparse case: the ring. Yet, while intuition and simulation results suggest that the process is self stabilizing in the sense defined above, we were not able to get any rigorous analytical results so far.

3 Further research directions

The notion of probabilistic self-stabilization applies to other fundamental problems in Distributed Computing. One of the most interesting is the classic problem of reaching (w.h.p.) a *stabilizing consensus* (or even more a stabilizing *majority* consensus [2, 3, 7, 14, 15]) in a distributed system consisting of a finite sets of agents. In a basic variant, each of the agents initially supports an opinion (say a value chosen from a fixed set K of legal values). The goal here is have the system converge to a state in which (w.h.p.) all nodes share the same opinion and this was present in the initial configuration. Moreover, solutions are required to be fault-tolerant (i.e. stable) w.r.t. some bounded adversary that can change the values of a subset of the nodes in each round of the consensus process. Important advances in probabilistic versions of stabilizing consensus were recently made [2, 7, 8, 15]. However, available concepts of stabilizing consensus do not fully reflect our proposed notion of probabilistic *self-stabilization*. Our weaker version of consensus may lead to more efficient and more robust protocols that still work in practice for most applications.

References

[1] A. Anagnostopoulos, A. Kirsch, and E. Upfal. Load balancing in arbitrary network topologies with stochastic adversarial input. *SIAM Journal on Computing*, 34(3):616–639, 2005.

[2] D. Angluin, J. Aspnes, and D. Eisenstat. A simple population protocol for fast robust approximate majority. In *Proc. of 21st DISC*, volume 4731 of *LNCS*, pages 20–32. Springer, 2007.

[3] J. Aspnes. Faster randomized consensus with an oblivious adversary. In *PODC'12*. ACM, 2012.

[4] Y. Azar, A. Z. Broder, A. R. Karlin, and E. Upfal. Balanced allocations. *SIAM journal on computing*, 29(1):180–200, 1999.

[5] J. Beauquier, C. Genolini, and S. Kutten. k-stabilization of reactive tasks. In *Proceedings of the Seventeenth Annual ACM Symposium on Principles of Distributed Computing, PODC '98, Puerto Vallarta, Mexico, June 28 - July 2, 1998*, page 318, 1998.

[6] L. Becchetti, A. Clementi, E. Natale, F. Pasquale, and G. Posta. Self-stabilizing repeated balls-into-bins. In *27th ACM Symposium on Parallelism in Algorithms and Architectures*, 2015. To appear.

[7] L. Becchetti, A. Clementi, E. Natale, F. Pasquale, and R. Silvestri. Plurality consensus in the gossip model. In *Proceedings of the 25th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2015.

[8] L. Becchetti, A. Clementi, E. Natale, F. Pasquale, R. Silvestri, and L. Trevisan. Simple dynamics for plurality consensus. *arXiv:1310.2858*, 2013. Preliminary version in ACM SPAA'14.

[9] P. Berenbrink, A. Czumaj, A. Steger, and B. Vöcking. Balanced allocations: The heavily loaded case. *SIAM Journal on Computing*, 35(6):1350–1385, 2006.

[10] P. Berenbrink, J. Czyzowicz, R. Elsässer, and L. Gasieniec. Efficient information exchange in the random phone-call model. In *Proceedings of the 37th International Colloquium on Automata, Languages, and Programming (ICALP)*, 2010.

[11] P. Berenbrink, T. Friedetzky, and L. A. Goldberg. The natural work-stealing algorithm is stable. *SIAM Journal on Computing*, 32(5):1260–1279, 2003.

[12] J. E. Burns, M. G. Gouda, and R. E. Miller. Stabilization and pseudo-stabilization. *Distributed Computing*, 7(1):35–42, 1993.

[13] E. W. Dijkstra. Self-stabilizing systems in spite of distributed control. *Commun. ACM*, 17(11):643–644, 1974.

[14] E. W. Dijkstra. Self-stabilizing systems in spite of distributed control. *Communications of the ACM*, 17(11):643–644, 1974.

[15] B. Doerr, L. A. Goldberg, L. Minder, T. Sauerwald, and C. Scheideler. Stabilizing consensus with the power of two choices. In *Proc. of 23rd SPAA*, pages 149–158. ACM, 2011.

[16] S. Dolev. *Self-stabilization*. MIT press, 2000.

[17] S. Dolev. *Self-Stabilization.* The MIT Press, 2000.

[18] R. Elsässer and D. Kaaser. On the influence of graph density on randomized gossiping. *Proceedings of the 29th IEEE International Parallel & Distributed Processing Symposium (IPDPS)*, 2015.

[19] M. G. Gouda. The theory of weak stabilization. In *Self-Stabilizing Systems, 5th International Workshop, WSS 2001, Lisbon, Portugal, October 1-2, 2001, Proceedings*, pages 114–123, 2001.

[20] A. Israeli and M. Jalfon. Token management schemes and random walks yield self-stabilizing mutual exclusion. In *Proceedings of the 9th annual ACM Symposium on Principles of Distributed Computing (PODC)*, 1990.

[21] L. Lamport. Solved problems, unsolved problems and non-problems in concurrency. *ACM SIGOPS Operating Systems Review*, 19(4):34–44, 1985.

[22] N. A. Lynch. *Distributed algorithms.* Morgan Kaufmann, 1996.

[23] M. Mitzenmacher and E. Upfal. *Probability and computing: Randomized algorithms and probabilistic analysis.* Cambridge University Press, 2005.

[24] D. Peleg. *Distributed Computing: A Locality-Sensitive Approach.* SIAM, 2000.

[25] M. Raab and A. Steger. "balls into bins"—a simple and tight analysis. In *Proceedings of the 2nd International Workshop on Randomization and Approximation Techniques in Computer Science (RANDOM)*, 1998.

[26] N. Santoro. *Design and analysis of distributed algorithms.* John Wiley & Sons, 2006.

[27] B. Vöcking. How asymmetry helps load balancing. *Journal of the ACM*, 50(4):568–589, 2003.