Self-made systems

Maarten van Steen, Spyros Voulgaris, Elth Ogston, Frances Brazier
Vrije Universiteit Amsterdam

1 Introduction

Self-* systems are designed to adapt to a changing environment such that specific properties are automatically restored when that environment disturbs the normal behavior. We also see that designs generally have several global parameters, which are subsequently configured for specific applications or domains. The choice of a parameter value often affects the emergent properties of a self-* system.

Parameters may need to be carefully tuned in order to obtain the *desired* emergent behavior. Ideally, the parameter value can be found by means of a feedback loop based on monitoring system behavior. This approach effectively transforms a parameter into just another system variable and has also been referred to as self-tuning in [4].

We take the position that in designing self-* systems we need to strive for *parameterless* designs, which we denote as *self-made* systems.[1] In this paper, we discuss two different examples of systems that we are currently developing to see how parameter choices affect prominent emergent behavior and how their influence can be minimized.

2 Unstructured overlays

Consider a dynamically changing collection of nodes N that jointly maintain an overlay network. Each node n has a list $V(n)$ of c *(neighbor, hop count)* pairs, referred to as its *view*. As N may change over time, nodes communicate to update their view. To this end, each node n repeatedly executes the following exchange protocol (let $N(t)$ denote the set of nodes at the current time t):

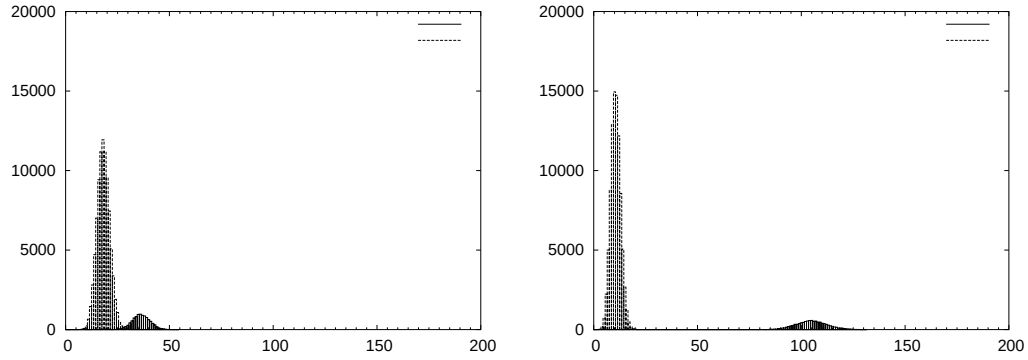
1. Randomly select a peer m with $(m,k) \in V(n)$. If $m \notin N(t)$, repeat with $V(n) = V(n) - (m,k)$
2. $V(n) \leftarrow V(n) \cup (n,0)$
3. Send $V(n)$ to m; receive $V(m)$ from m; $\forall (p,k) \in V(m) : (p,k) \leftarrow (p,k+1)$
4. $V^*(n) \leftarrow V(n) \cup V(m)$ such that no node is listed more than once.
5. $V(n) \leftarrow V^*(n)$ restricted to the c entries with the lowest hop count.

The contacted node m executes all but the first step as well. In the original version of this protocol, called Newscast [3], each node executes the exchange protocol once every ΔT time units. As it turns out, with $c = 20$ networks as large as 100,000 nodes remain connected, regardless of the initial topology. Moreover, these networks have been demonstrated to be highly robust.

There are several design parameters that influence the emergent behavior of the protocol, of which the two most prominent ones are the view size c and the cycle length ΔT. (A discussion and evaluation of other parameters can be found in [2].) The choice of c affects the connectivity of the network, as well as properties such as clustering and others related to complex networks [1], but has not been found critical in the sense that small changes lead to very different behavior.

More interesting, in this respect, are the—conflicting—factors that inflict the choice of the cycle length. The cycle length determines the rate at which views are exchanged, and thus the speed at which changes in the set of nodes are detected. In other words, a small cycle length is required to keep a rapidly changing set of nodes up to date,

[1] Main Entry: self-made. Function: adjective. Description: made such by one's own actions; especially: having achieved success or prominence by one's own efforts <a self-made man>.



(a) (b)

Figure 1: In-degree distribution when 10% of the nodes run (a) twice as fast, and (b) ten times as fast as the rest, respectively.

whereas it would be an overkill—in terms of processor and network resources—for a set of nodes changing at a significantly lower rate. In the extreme scenario of a non-changing overlay, exchanging lists is merely useless. Moreover, the value for ΔT has to be chosen such that all nodes can execute and complete the exchange protocol. That is, ΔT is bound to a minimum value that is dependent on the slowest node and the minimum internode communication speed across all pairs of nodes.

An alternative is therefore is to adapt ΔT such that it may vary in the course of time, but also that it is no longer a global value but simply *local* to each node. We are thus confronted with designing a system in which each node should be allowed to locally decide how often it initiates the exchange protocol, and when and how it changes its view-exchange rate.

Turning the cycle length from a global parameter to a local one has obvious benefits, but has also severe effects on the emergent behavior of the overall system. For example, without taking any counter measures the network will rapidly partition into many clusters. A promising solution is to modify the merge operation such that properties such as connectivity become invariant [6].

Even in this case, though, turning the cycle length to a local parameter can have undesirable effects for the quality of the overlay formed. We observed that non-uniform cycle lengths among nodes results in an unbalanced in-degree distribution. Figure 1 shows the in-degree distribution for an experiment where 10% of the nodes run at a faster speed than the rest, namely 2 times faster in 1(a), and 10 times faster in 1(b). Fast nodes tend to have respectively 2, or 10 times higher in-degree than the nodes running at normal speed.

Let us take a closer look at this example. In general, *fast* nodes run at a speed $speed_f$, and *slow* ones run at $speed_s$. Since the in-degree of a node increases by one each time it shuffles, the expected in-degree ratio between fast and slow nodes will be proportional to their shuffling speed ratio, that is,

$$\frac{indegree_f}{indegree_s} = \frac{speed_f}{speed_s}.$$

Also, the sum of in-degrees of all nodes is equal to the total number of outgoing links, that is $N \times c$. If the percentage of fast nodes is f, and, therefore, the percentage of slow ones is $1-f$, we have: $f \times indegree_f + (1-f) \times indegree_s = c$

From these formulas we can compute the expected in-degrees of fast and slow nodes to be: $indegree_f = speed_f \times A$, and $indegree_s = speed_s \times A$, where

$$A = \frac{c}{f \times speed_f + (1-f) \times speed_s}.$$

Our formula suggests that with $c = 20$, the in-degree for a fast and slow node is 36.36 and 18.18 for 1(a), respectively, and 105.26 and 10.53 for 1(b).

Furthermore, we have found that by simply overdimensioning the view size, while exchanging as few as only

2 randomly selected items from the cache, many desirable properties can be retained. This approach shifts the problem to membership management where the joining or leaving of a node should at the very least restore the original graph.

Where joining appears to be relatively simple, parameterless scalable detection of failing nodes remains a challenge. We are currently investigating under which circumstances the joining of a node can trigger enough events to also detect failed nodes. Such a scheme would prevent having to use a separate heartbeat protocol (with its inevitable probe interval).

3 Decentralized data clustering

As another example, consider a network of agents, each representing a data item. Agents proactively construct links of which the length reflects the semantic proximity of their respective data items. This approach effectively leads to a collection of graphs, each graph connecting semantically related agents and thus forming a data cluster. The details of this decentralized data clustering scheme are described in [5], where we also demonstrate that the quality of clustering is competitive with well-known centralized approaches.

In the original algorithm, we used two parameters to steer the clustering. First, we feed the agents with the maximum length of a good link, λ. Links above this length are considered bad and as such should not be used to place two agents in the same cluster. Second, like in many data clustering approaches, data clusters were not allowed to grow beyond a certain maximum size s. Such a maximum is necessary to prevent ending up with only a single giant cluster. Again, we see examples of application-dependent, global parameters that should be avoided.

Eliminating a fixed value for λ turned out to be relatively easy in the case an agent's data represented a point in a 2-dimensional Euclidean space (the situation we have investigated extensively so far). The essence of our approach is that we let agents learn an appropriate value for λ, as follows:

0. init: $\lambda \leftarrow 0$

1. recording phase: Watch a series of 50 links, recording the shortest length d. If during the recording phase a link is seen that is shorter λ, go to step 0. Otherwise, go to step 2.

2. update phase: When a link is rejected (i.e., its two agents are not put into the same cluster on account of their link), $\lambda \leftarrow \lambda + d/100$. Go to step 1 when a link is encountered with length $l < \lambda$, or when $\lambda \geq d$. Restart step 2 when a link is encountered with length $l < d$, setting $d \leftarrow l$.

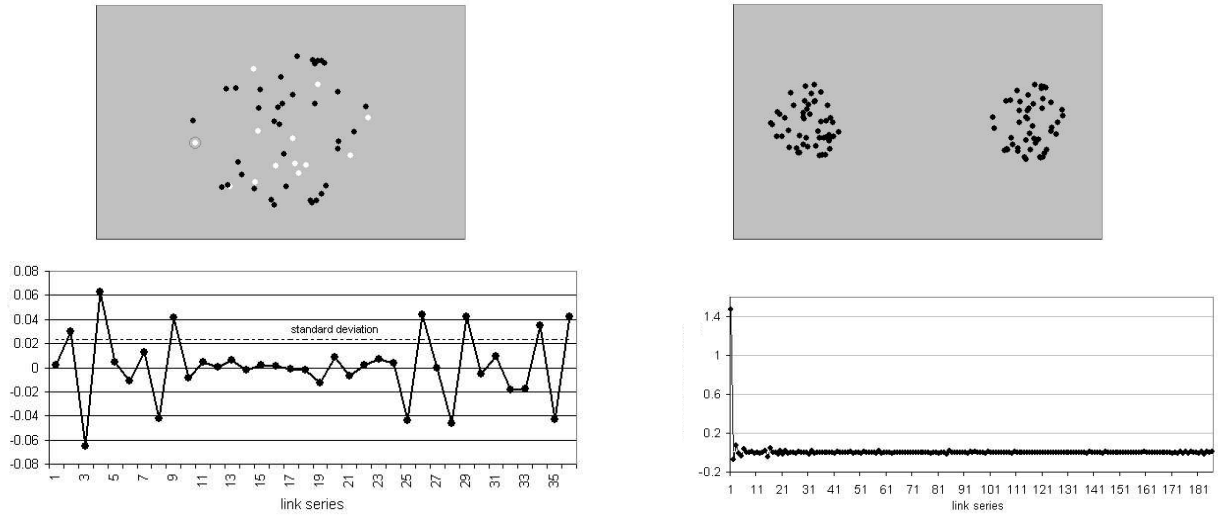
3. match found: Whenever a new match with length l is made, set $\lambda \leftarrow l$ and go to step 1.

With these adjustment rules, we have been able to let agents discover the correct value for λ. As a result, λ is no longer a design parameter but can be considered as another system variable that is optimized during runtime, in this case by means of a simple learning procedure.

Removing the maximum cluster size is a much more difficult problem. Rather than pessimistically preventing clusters from growing too large, we have chosen to allow clusters to grow to a point where it may be necessary to split them again. To this end, the links in a cluster are ordered by their length. A crucial observation is that when a cluster should be split, this series will generally show a pronounced gap. (Note that such a gap will generally occur at the *beginning* of a series.) To increase the accuracy of gap detection, each time a link is added between two agents that leads to a cycle, we remove the longest link in that cycle effectively aiming at the construction of a minimal spanning tree. As a result, the removal of any link will split a cluster into two.

The gap can be found by considering the second derivative of the series: $f''(x) = (y_2 - 2y_1 + y_0)$, where y_0, y_1, y_2 are consecutive lengths and x is the position in the series of y_0. However, taking a constant-valued threshold to determine whether or not a gap is large appears to be dependent on the data. To reduce this dependency, we compute the standard deviation σ of the series $\{f''(x)\}$. This value should account for the "normal" variation that we could expect in the link lengths of a cluster. Figure 2 shows this approach works in the case of a good and a bad cluster, respectively.

From this information we then decide on a reasonable value γ to take further decisions on to whether a gap actually indicates that the corresponding link should be removed from the cluster (and thus always splitting the original cluster into two parts). For our data sets, we experi-



(a) (b)

Figure 2: Example of detecting (a) good and (b) bad clusters using a "cluster separator" γ.

mentally found that setting $\gamma = 7\sigma$ does a wonderful job. However, although we can show that our approach is less sensitive to the type of data that is being clustered than many others, it is also clear that it cannot be easily generalized to handle arbitrary data sets.

In this case, we are gradually reaching a point at which we will have to conclude that a fully self-made system for decentralized data clustering may be impossible. Instead, we will have to separate application domains and find reasonable criteria within each domain for splitting clusters.

4 Conclusions

These two examples each illustrate the importance of striving for parameterless designs, but also that the road to these designs is often not evident. In general, we doubt that it is possible to develop completely self-made systems, such as in the case of decentralized data clustering where the semantics of the data may need to be taken into consideration. However, the first example shows that initial design decisions can be replaced by alternatives that lead to an improvement of the original system.

5 Acknowledgments

The work described in this paper is carried out in close collaboration with others. Special thanks goes to Daniela Gavidia Simonetti, Márk Jelasity, Wojtek Kowalcyck, and Benno Overeinder.

References

[1] R. Albert and A.-L. Barabasi. "Statistical Mechanics of Complex Networks." *Reviews of Modern Physics*, 74(1):47–97, Jan. 2001.

[2] M. Jelasity, W. Kowalczyk, and M. van Steen. "Newscast Computing." Technical Report IR-CS-006, Vrije Universiteit Amsterdam, Department of Computer Science, 2003.

[3] M. Jelasity and M. van Steen. "Large-Scale Newscast Computing on the Internet." Technical Report IR-503, Vrije Universiteit, Department of Computer Science, Oct. 2002.

[4] R. Mahajan, M. Castro, and A. Rowstron. "Controlling the Cost of Reliability in Peer-to-Peer Overlays." In *Second Int'l Workshop on Peer-to-Peer Systems*, volume 2735 of *Lect. Notes Comp. Sc.*, pp. 21–32. Springer-Verlag, Berlin, Feb. 2003.

[5] E. Ogston, B. Overeinder, M. van Steen, and F. Brazier. "A Method for Decentralized Clustering in Large Multi-Agent Systems." In *Proc. Second Int'l Joint Conf. Autonomous Agents and Multiagent Systems*, July 2003. ACM Press, New York, NY.

[6] A. Stavrou, D. Rubenstein, and S. Sahu. "A Lightweight, Robust P2P System to Handle Flash Crowds." *IEEE J. Selected Areas Commun.*, 22(1):6–17, Jan. 2004.