Towards very large, self-managing distributed systems

Extended abstract

Maarten van Steen

Vrije Universiteit Amsterdam

Introduction

As distributed systems tend to grow in the number of components and in their geographical dispersion, deployment and management are increasingly becoming problematic. For long, there has been a tradition of developing architectures for managing networked and distributed systems [2]. These architectures tend to be complex, unwieldy, and indeed, difficult to manage. We need to explore alternative avenues if we want to construct a next generation of distributed systems.

Recently, solutions have been sought to develop self-managing systems. The basic idea here, is that a distributed system can continuously monitor its own behavior and take corrective action when needed. As with many new, or newly introduced, concepts, it is often difficult to separate hype from real content. In the case of self-management (or other forms of *self-∗-ness*), the low signal-to-noise ratio can be partly explained by our poor understanding of what self-management actually means.

A self-managing user-centric CDN

In our own research on large-scale distributed systems at the Vrije Universiteit Amsterdam, we have been somewhat avoiding the problem of systems management. However, one of the lessons we learned from building Globe [8], is that supporting easy deployment and management is essential. Partly based on our experience with Globe, we are currently developing a user-centric Content Delivery Network to further explore facilities for self-management. This CDN, called Globule, is designed to handle millions of users, each providing Web content by means of a specially configured Apache Web server.

An important aspect of Globule is that a server can automatically replicate its Web documents to other servers. For each document, a server evaluates several replication strategies, and selects the best one on a per-document basis. This approach allows for near-optimal performance in terms of client-perceived delays as well as total consumed bandwidth [6]. In a recent study, we have also demonstrated that continuous re-evaluation of selected strategies is needed, and that this can be done efficiently [7]. The approach we follow is to regularly perform trace-driven simulations for a specific document, where each simulation entails a single replication strategy. Using a linear cost function defined over performance metrics such as client-perceived latency and consumed bandwidth, we can then compare the effects of applying different strategies. These simulations take in the order of tens of milliseconds in order to select the best

strategy for a given document. Clearly, Globule should be able to manage itself when it comes to replicating documents, and as far as static content is concerned, such self-management appears to be feasible.

However, much more is needed to develop a CDN such as Globule. For one thing, if we are to replicate documents to where they are needed, it is mandatory that we can locate clients and replica servers in the proximity of those clients. One problem that needs to be solved is letting a Web server determine how close two arbitrary nodes in the system actually are. Fortunately, it turns out that if we consider latency as a distance metric, we can represent the nodes of a widely dispersed distributed system in an N-dimensional Euclidean space [4, 5]. In this way, estimating latency is nothing more than a simple computation. In contrast to existing systems, latency estimations in Globule can be obtained in a fully decentralized manner, which, in turn, simplifies overall system management.

By introducing locations and easy-to-compute distances, it becomes feasible to automatically partition the set of nodes comprising a distributed system into manageable parts. For example, by grouping nodes into geographical zones (where the geography is fully determined by the Euclidean space mentioned before), we can assign special nodes to zones in order to manage services, resources, etc. These special nodes, called brokers in Globule, are elected as *super peers* from all available nodes, and together form a separate overlay network using their own routing protocol. Whenever a node in zone A requires services from a zone B (such as, for example, a list of potential replica servers) it simply sends a request to a broker in A which will then forward the request to a broker for B. In Globule, a zone is defined implicitly: it consists of the servers that are closest to a given broker. As a consequence, zones do not overlap. Moreover, adding and removing servers, be they brokers or not, is fully decentralized.

There are many variations on this theme, but it should be clear that grouping nodes into zones and electing brokers for zones are things that can be done in a fully decentralized fashion. There is no need for manual intervention, although there are many unresolved details concerning how this organization can be automatically done.

Epidemic-based solutions

One could argue that the description of a self-managing system given so far is largely dictated by automating tasks that are currently handled manually. In this sense, self-management is just a next step in the evolution of distributed systems. The question comes to mind if there are radically different alternatives. We are currently exploring epidemic-based systems for management tasks.

In an epidemic-based system, we are generally concerned with reaching eventual consistency: in the absence of any further updates, all nodes should eventually reach the same state. Data are spread by letting each node regularly contact an arbitrary other node, after which the two exchange updates [1]. The problem with this approach for very large systems, is that, in principle, every node should know the entire set of nodes in order to guarantee random selection of a peer. One solution is to maintain, per node, a small list of peers that represents a random sample from all nodes. Maintaining this list is now the key to successfully applying epidemics in very large systems.

In the Newscast system, we take a simple approach. Every node maintains a fixed-size list of length c of *news items*. A news item contains data, the address of its source, and the time when it was published. Once every ΔT time units, each node executes the following steps [3]:

1. Add a fresh (node-specific) news item to the local list.
2. Randomly select a peer from those found in the list.
3. Send all entries to the selected peer, and, in turn, receive all that peer's list entries.
4. Out of the (up to) $2c$ cache entries, keep the c newest ones, and discard the rest.

The selected peer from step 2 executes the last two steps as well, so that after the exchange both nodes have the same list. Note that as soon as any of these two nodes executes the protocol again, their respective lists will most likely be different again.

The lists induce a communication graph with a link from node a to b if b is contained in the list of a. As it turns out, for even a relatively small list size ($c = 20$), we can show that communication graphs are strongly connected for up to hundred thousands of nodes, and by simply increasing c, very large overlay networks of millions of nodes can be accommodated. These networks exhibit properties common to what are known as *small worlds* [9]. We have discovered that these networks are extremely robust: in general, only after the removal of at least 70% of the nodes (and much more for $c > 20$), the network is partitioned into one giant cluster and several (very) small clusters. Most important, however, is that there is no need for centralized control — Newscast networks are completely self-managing.

To illustrate this point, consider how membership is managed in Newscast. First, nodes that want to leave can simply stop communication: they do not initiate a list exchange, nor do they react to exchange requests. This behavior is exactly the same as that of a failing node, and indeed, Newscast treats these two cases identically.

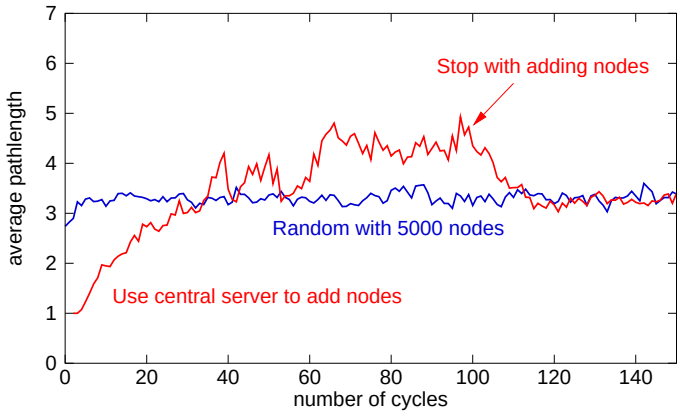

Fig. 1. The effect of continuously adding nodes that contact the same initial node.

Second, when a node wishes to join the network, it need only know the address of one node, with whom it then seeks contact by executing the list exchange protocol (note that its own list is empty). To see to what extent the selection of the initial contact node affects overall behavior, we conducted a series of experiments in which all joining nodes contact *the same* initial node. Clearly, this is a worst-case scenario. Figure 1 shows the effect on the average path length while continuously adding 50 nodes until 5000 nodes have joined. Compared to the "ideal" case, in which each new node contacts another, randomly selected node, we see that soon after nodes are no longer added, the average path length converges to that of the random case. Again, we see self-management at work.

We have used Newscast for various system monitoring tasks, such as estimating the size of the network, load balancing, and broadcasting alarms. In all cases, only a few list exchanges per node are required to reach consistency. As such, it forms a powerful tool for management of large distributed systems, notably because the protocol itself barely requires any management at all.

Conclusions

Self-managing systems are in many cases not well understood and as a consequence there is often much ado about nothing. However, it is also clear that we need concepts such as self-management, self-healing, self-organization, and so on, in order to arrive at a next generation of very large distributed systems. Fortunately, self-$*$ matters can be made concrete and show to form a promising field that warrants further research.

References

1. A. Demers, D. Greene, C. Hauser, W. Irish, J. Larson, S. Shenker, H. Sturgis, D. Swinehart, and D. Terry. "Epidemic Algorithms for Replicated Database Maintenance." In *Proc. Sixth Symp. on Principles of Distributed Computing*, pp. 1–12. ACM, Aug. 1987.
2. H.-G. Hegering, S. Abeck, and B. Neumair. *Integrated Management of Networked Systems*. Morgan Kaufman, San Mateo, CA, 1999.
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. E. Ng and H. Zhang. "Predicting Internet Network Distance with Coordinates-Based Approaches." In *Proc. 21st INFOCOM Conf.*, June 2002. IEEE Computer Society Press, Los Alamitos, CA.
5. M. Pias, J. Crowcroft, S. Wilbur, T. Harris, and S. Bhatti. "Lighthouses for Scalable Distributed Location." In *Proc. Second Int'l Workshop on Peer-to-Peer Systems*, Feb. 2003. Springer-Verlag, Berlin.
6. G. Pierre, M. van Steen, and A. Tanenbaum. "Dynamically Selecting Optimal Distribution Strategies for Web Documents." *IEEE Trans. Comp.*, 51(6):637–651, June 2002.
7. S. Sivasubramanian, G. Pierre, and M. van Steen. "A Case for Dynamic Selection of Replication and Caching Strategies." In *Proc. Eighth Web Caching Workshop*, Sept. 2003.
8. M. van Steen, P. Homburg, and A. Tanenbaum. "Globe: A Wide-Area Distributed System." *IEEE Concurrency*, 7(1):70–78, Jan. 1999.
9. D. J. Watts. *Small Worlds, The Dynamics of Networks between Order and Randomness*. Princeton University Press, Princeton, NJ, 1999.