Beyond Replicated Storage: Eventually-Consistent Distributed Data Structures

Konrad Iwanicki *
University of Warsaw
Warsaw, Poland
iwanicki@mimuw.edu.pl

Abstract

While our understanding of eventual consistency for replicated data has improved considerably over the past few years, relatively little work focused on eventually-consistent distributed data structures. By revisiting PL-GOSSIP, our self-managed, decentralized algorithm for maintaining cluster hierarchies in wireless sensor networks, this paper illustrates sample problems one is likely to encounter when dealing with such structures. Given the growing interest in analyzing huge, dynamic, inter-dependent data sets, we believe that the paper will inspire research to facilitate understanding and devising eventually-consistent distributed data structures.

1. Original Problem

Systems based on wireless sensor networks are often classified as extreme distributed systems. On the one hand, it is not uncommon for such systems to consist of hundreds or even thousands of nodes, which have to collaborate to provide a desired functionality. On the other hand, the nodes are severely constrained in resources, namely RAM, ROM, computing power and network throughput, and the wireless communication they employ is highly dynamic and unreliable, not to mention mobility in some scenarios.

While in the past the functionality of such systems typically boiled down to fine-grained monitoring of physical spaces, increasingly it corresponds to an entire feedback loop encompassing sensing, decision making, and actuation. Considering that they also have to deal with the resource constraints of the nodes, as well as with potentially high loss rates, low correlation time, and partitions of the wireless links, the distributed algorithms implementing the functionality are becoming more and more intricate.

An approach taken by some of such algorithms relies on the nodes being organized in a hierarchical overlay network on top of the physical topology induced by the wireless inter-node links. An example of such an organization, employed among others by efficient point-to-point routing algorithms, spatial distributed hash tables, and multi-resolution in-network aggregation schemes, is an area/group/landmark/cluster hierarchy.

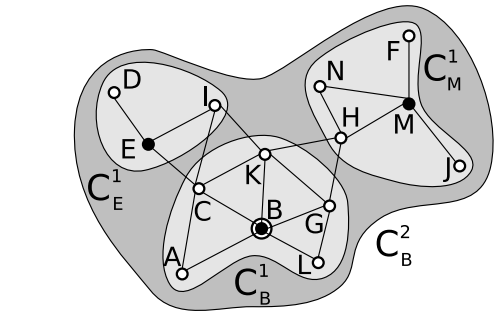
A sample cluster hierarchy is depicted in Figure 1. At level 0, each node corresponds to a singleton *cluster*: in Figure 1(a), nodes $A, B, C, \ldots$ correspond to level-0 clusters C^0_A, C^0_B, C^0_C, ..., respectively (omitted in the figure for clarity). Proximate level-0 clusters form larger clusters at level 1 (C^1_B, C^1_E, and C^1_M in the figure), which in turn form yet larger clusters at level 2, and so on at higher levels, so that typically a single top-level cluster covers the entire network (C^2_B in the figure). Each such cluster has a *head* node, H, which, in combination with the level, L, uniquely identifies the cluster, C^L_H. Based on the clusters it belongs to, each node can thus be given a unique *label* that is a concatenation of the node's cluster head identifiers at every level. In particular, the label of node D is $D.E.B$. The labels of all nodes can be viewed as a tree, such as the one depicted in Figure 1(b). Finally, based on its label, each node maintains information on its siblings in the hierarchy, as in Figure 1(c). The information can be a routing entry and a value of an aggregate to name just two examples. The advantage of organizing the network in such a hierarchy is that the amount of state maintained and exchanged by the nodes can scale poly-logarithmically with the network size.

Although utilizing such a hierarchy for routing, aggregation, or in-network storage is fairly straightforward, a major problem is building and maintaining it in the presence of network dynamics. Due to connectivity changes as well as node failures and arrivals, some clusters at different levels

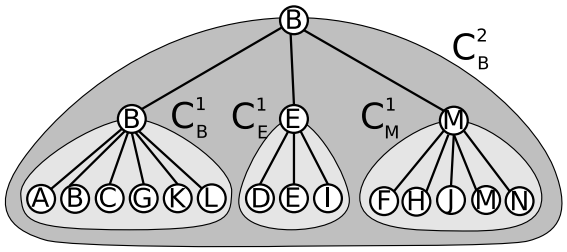
* Supported by the (Polish) National Science Center under grant no. DEC-2012/05/D/ST6/03582.

PaPEC'14, April 13–16, 2014, Amsterdam, Netherlands.

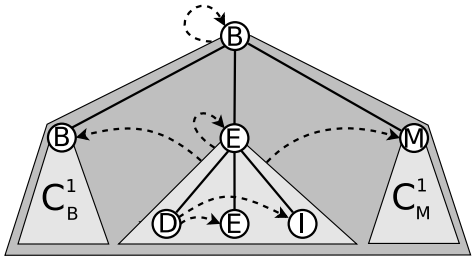
http://dx.doi.org/10.1145/2596631.2596639



(a) cluster hierarchy vs. physical connectivity



(b) node labels as a tree



(c) routing table of node D (label $D.E.B$)

Figure 1. A sample cluster hierarchy.

must be dissolved and new ones must be formed. As a sensor network should normally operate unattended, the entire process is expected to be self-managed and decentralized, that is, autonomous decisions of individual nodes should—on the global scale—yield a consistent hierarchy.

2. Consistency Perspective

Prior algorithms for hierarchy maintenance rely on each cluster head periodically flooding the network with advertisements, up to a radius that depends on the level of the cluster. Albeit simple, such an approach is costly in wireless sensor networks [14].

In many algorithms for wireless sensor networks, flooding is thus abandoned in favor of more efficient local gossiping. In essence, each node periodically broadcasts its local state to its neighbors (i.e., other nodes within its radio range). Likewise, it receives the states of the neighbors and merges them with its local state. The repeated broadcasts and merges allow information to propagate with less traffic than in the case of flooding [13]. Therefore, in the remainder of our discussion, we assume local gossiping as the base of a cluster hierarchy maintenance algorithm.

However, with gossiping, it is difficult to control precisely when a bit of information reaches a particular node, especially in the face of network dynamics. It is common for bits of information to be merged into a node's state in a different order than the one in which they are produced. Likewise, a bit of information can be merged multiple times. In other words, local gossiping guarantees that information will eventually have a chance to reach every node, but only if the state merging operation is designed properly. Combined with the autonomous node decisions on how the hierarchy should look like at a given moment, these assumptions bear some resemblance to highly-available, eventually-consistent storage systems.

This observation has motivated us to look at the problem of cluster hierarchy maintenance not from the networking, but rather consistency perspective. We treat the cluster hierarchy as a distributed data structure. The state of each node, that is, its label and routing table, is a part of this structure. Each node can autonomously update its local state, thereby altering the distributed structure. The nodes exchange and merge their states through gossiping, which should ensure that eventually the structure is globally consistent.

3. Eventually-Consistent Distributed State

However, compared to traditional models for eventual consistency, what is different in our model is that the local state of each node is not a replica. On the contrary, the state represents a piece of the distributed structure: some of its parts are globally unique; some others, in turn, are replicas of the corresponding parts at other nodes. Recall, for instance, the label tree from Figure 1(b). Consider the label of node D: $D.E.B$. The first part of D's label, $D.E$, informs that cluster C^0_D is a subcluster of C^1_E. It is stored only at node D. The second part, $E.B$, informs in turn that C^1_E is a subcluster of C^2_B. It is replicated in the labels of nodes D, E, and I. Therefore, when it changes due to an autonomous decision of some node, all the three nodes must eventually learn about the change and update their states accordingly. The difficulty is that updates to different or same parts of the structure can be concurrent, are often not independent, and propagate lazily through gossiping, not to mention the bandwidth and memory constraints of the nodes.

To cope with these problems, a hierarchy maintenance algorithm must address the following issues: *How to decide that a given piece of the distributed structure should be updated? How such updates should be performed and which node(s) should do them? How can other nodes detect and merge the updates to their corresponding pieces of the distributed structure?* Whereas the complete PL-GOSSIP al-

gorithm addressing these issues, including proofs and real-world experiments, can be found, for instance, in an earlier article [9] and Ph.D. dissertation [7], here we just give a brief overview of its major design features.

To begin with, the properties of the distributed structure are formalized as invariants. Sample invariants can be as follows:

Invariant 1 *Level-*0 *clusters correspond to individual nodes.*

Invariant 2 *There exists a single, top-level (*$\mathcal{H}$*) cluster containing all nodes.*

Invariant 3 *Level-*i+1 *clusters (where* $0 \leq i < \mathcal{H}$*) are composed out of level-*i *clusters, such that each level-*i *cluster is nested in exactly one level-*i+1 *cluster.*

Invariant 4 *Each level* i+1 *cluster (where* $0 \leq i < \mathcal{H}$*) has a* central subcluster *that is adjacent (in terms of the neighbor relation between nodes) to all other subclusters of this cluster.*

Maintaining the hierarchy boils down to detecting violations of the invariants and eliminating them. The invariants are global, and hence, they have to be maintained collaboratively by all nodes. However, each node is concerned with only those invariants that are relevant to its part of the distributed structure. To this end, it autonomously examines its local state whenever it changes and corrects it upon an invariant violation. As mentioned previously, the state comprises the node's routing table, which contains one entry for each sibling cluster, the node's label, and additional metadata.

A routing entry stored by a node for a cluster includes, among others, the length of the shortest path from the node to the cluster, the identifier of the next-hop neighbor on this path, and a flag indicating if the cluster is adjacent to the present node's same-level cluster. The creation and merge operations for the entries are arguably the most complex ones in the algorithm. In particular, to avoid update conflicts, the entry for a cluster can be created only by the cluster head. Moreover, to enable determining whether an entry a node receives in a gossip message is fresher than the node's own entry, the cluster head assigns a monotonic timestamp to each new version of the entry. Merging two entries is nevertheless far more intricate than merely comparing the timestamps, as it also has to minimize the routing path length, correctly compute the adjacency flag, and choose the next-hop neighbor, not to mention handling route poisoning that facilitates detecting failures of cluster heads.

Changes to the local routing entries, as resulting from the merges, enable detecting invariant violations and repairing them by changing the cluster hierarchy. There are two local operations for such changes, label cut and extension, which correspond to, respectively, removing a subcluster from a cluster and to joining a subcluster to a cluster. Again, to avoid update conflicts, these operations are executed only by cluster heads, while other nodes just adopt them. A major problem is again a label merge operation. In particular, a node with label $D.E.B$, seeing a label $I.E.C$, has to decide whether its label should stay as is or become $D.E.C$. To solve this problems, we devised a custom structure, update vectors, that enforces causal order of label operations albeit in a scalable manner as compared to, for instance, classic vector clocks.

4. General Lessons and Future Agenda

All in all, our PL-GOSSIP algorithm, resulting from abandoning flooding in favor of eventually-consistent gossiping, outperforms prior algorithms for cluster hierarchy maintenance in wireless sensor networks [9]. In particular, by employing it, we demonstrated scalable hierarchical routing [10] and aggregation [8] schemes.

More importantly, however, we hope that the lessons learned and problems encountered when devising our eventually-consistent distributed cluster hierarchy can inspire solutions in other areas. For example, in the field of big data, there is a growing interest in analyzing graphs representing social networks or tweets. Due to their sheer size, such graphs must be distributed across many machines. Moreover, they are changing constantly. A major challenge is thus to perform the analysis incrementally, by updating the data structures used for the analysis (e.g., clusters) together with the changes in the graphs, rather than rebuilding them from scratch [4, 12]. We believe that the ideas we touched upon in this paper can inspire such novel solutions relying on eventual consistency.

Furthermore, we believe that there is yet a lot of work ahead to fully understand eventually-consistent distributed data structures. The progress on eventual consistency in the context of replicated storage has given us replicated data types [1, 15, 16], formal models [2, 3, 6], and different merge techniques [5, 11, 17], to name just a few examples. The understanding of eventually-consistent distributed data structures is far inferior. In particular, formal models are lacking that would allow for proving the properties of such structures not only in quiescent states, but also during continuous changes. Likewise, solutions such as these in PL-GOSSIP have yet to be generalized and classified. Overall, eventually-consistent distributed data structures may constitute an exciting research agenda.

References

[1] A. Bieniusa, M. Zawirski, N. Preguiça, M. Shapiro, C. Baquero, V. Balegas, and S. Duarte. Brief announcement: Semantics of eventually consistent replicated sets. In *Proceedings of the 26th International Conference on Distributed Computing*, DISC'12, pages 441–442, Berlin, Heidelberg, 2012. Springer-Verlag. ISBN 978-3-642-33650-8. .

[2] S. Burckhardt, D. Leijen, M. Fähndrich, and M. Sagiv. Eventually consistent transactions. In *Proceedings of the 21st European Conference on Programming Languages and Systems*, ESOP'12, pages 67–86, Berlin, Heidelberg, 2012. Springer-Verlag. ISBN 978-3-642-28868-5. .

[3] S. Burckhardt, A. Gotsman, and H. Yang. Understanding eventual consistency. Technical Report MSR-TR-2013-39, Microsoft Research, March 2013.

[4] R. Cheng, J. Hong, A. Kyrola, Y. Miao, X. Weng, M. Wu, F. Yang, L. Zhou, F. Zhao, and E. Chen. Kineograph: Taking the pulse of a fast-changing and connected world. In *Proceedings of the 7th ACM European Conference on Computer Systems*, EuroSys '12, pages 85–98, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1223-3. .

[5] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels. Dynamo: Amazon's highly available key-value store. In *Proceedings of Twenty-first ACM SIGOPS Symposium on Operating Systems Principles*, SOSP '07, pages 205–220, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-591-5. .

[6] A. Fekete, D. Gupta, V. Luchangco, N. Lynch, and A. Shvartsman. Eventually-serializable data services. In *Proceedings of the Fifteenth Annual ACM Symposium on Principles of Distributed Computing*, PODC '96, pages 300–309, New York, NY, USA, 1996. ACM. ISBN 0-89791-800-2. .

[7] K. Iwanicki. *Hierarchical Routing in Low-Power Wireless Networks*. PhD thesis, Vrije Universiteit Amsterdam, Amsterdam, the Netherlands, June 2010.

[8] K. Iwanicki and M. van Steen. Using area hierarchy for multi-resolution storage and search in large wireless sensor networks. In *Communications, 2009. ICC '09. IEEE International Conference on*, pages 1–6, June 2009. .

[9] K. Iwanicki and M. van Steen. Gossip-based self-management of a recursive area hierarchy for large wireless sensornets. *Parallel and Distributed Systems, IEEE Transactions on*, 21 (4):562–576, April 2010. ISSN 1045-9219. .

[10] K. Iwanicki and M. Van Steen. A case for hierarchical routing in low-power wireless embedded networks. *ACM Trans. Sen. Netw.*, 8(3):25:1–25:34, Aug. 2012. ISSN 1550-4859. .

[11] P. R. Johnson and R. Thomas. Maintenance of duplicate databases, 1975.

[12] U. Kang, C. Tsourakakis, and C. Faloutsos. Pegasus: A peta-scale graph mining system implementation and observations. In *Data Mining, 2009. ICDM '09. Ninth IEEE International Conference on*, pages 229–238, Dec 2009. .

[13] P. Levis, N. Patel, D. Culler, and S. Shenker. Trickle: A self-regulating algorithm for code propagation and maintenance in wireless sensor networks. In *Proceedings of the 1st Conference on Symposium on Networked Systems Design and Implementation - Volume 1*, NSDI'04, pages 2–2, Berkeley, CA, USA, 2004. USENIX Association.

[14] S.-Y. Ni, Y.-C. Tseng, Y.-S. Chen, and J.-P. Sheu. The broadcast storm problem in a mobile ad hoc network. In *Proceedings of the 5th Annual ACM/IEEE International Conference on Mobile Computing and Networking*, MobiCom '99, pages 151–162, New York, NY, USA, 1999. ACM. ISBN 1-58113-142-9. .

[15] M. Shapiro, N. Preguiça, C. Baquero, and M. Zawirski. A comprehensive study of Convergent and Commutative Replicated Data Types. Technical Report RR-7506, INRIA, January 2011.

[16] M. Shapiro, N. Preguiça, C. Baquero, and M. Zawirski. Conflict-free replicated data types. In *Proceedings of the 13th International Conference on Stabilization, Safety, and Security of Distributed Systems*, SSS'11, pages 386–400, Berlin, Heidelberg, 2011. Springer-Verlag. ISBN 978-3-642-24549-7.

[17] D. B. Terry, M. M. Theimer, K. Petersen, A. J. Demers, M. J. Spreitzer, and C. H. Hauser. Managing update conflicts in bayou, a weakly connected replicated storage system. In *Proceedings of the Fifteenth ACM Symposium on Operating Systems Principles*, SOSP '95, pages 172–182, New York, NY, USA, 1995. ACM. ISBN 0-89791-715-4. .