

Scalable Adaptive Hierarchical Clustering

Laurent Mathy, *Member, IEEE*, Roberto Canonico, Steven Simpson, and David Hutchison

Abstract—We propose a new application-level clustering algorithm capable of building an overlay spanning tree among participants of large multicast sessions, without any specific help from the network routers. The algorithm and associated protocol are shown to exhibit scalable properties.

Index Terms—Clusters, hierarchy, IP, multicast, network.

I. INTRODUCTION

MORE than a decade of research in multicast technologies demonstrates the need for large-scale (application-level) overlay structures.

Tree-based ACK's (TRACK's) have been identified as promising approach to providing real-time and scalable delivery guarantees to groups of receivers [1], [2]. In this scenario, the overlay provides a control structure.

More recently, reasons for the lack of widespread deployment of IP multicast have been identified [3]. These indicate that ubiquitous rollout of IP multicast services may, even if at all possible, take a very long time. In such circumstances, overlays represent an attractive alternative to IP multicast for data dissemination among members of multicast groups.

In this letter, we propose a new method design to build large-scale overlays, without requiring any special support from the network routers.

II. ADAPTIVE HIERARCHICAL CLUSTERING ALGORITHM

A. General Strategy and Goal

The algorithm described in this section is designed to build, recursively, a hierarchy of *clusters*. A cluster is represented by a *cluster head* and is composed of the cluster head and other nodes “close” to the cluster head. The algorithm is “recursive” in the sense that each cluster is divided into sub-clusters, whose (sub-)cluster heads are constituent nodes of the original cluster. The hierarchy of clusters is organized into *layers*, where layer L_i is composed of the cluster heads of (sub-)clusters that divide L_{i-1} -clusters (i.e., clusters whose head is in layer L_{i-1}). This is illustrated in Fig. 1(a). For instance, in this figure, the L_1 -cluster headed by C is composed of C , F and G . This cluster contains two L_2 -clusters, respectively headed by F and G .

The principle of the algorithm is that, starting at layer L_0 with a top-level cluster containing all the nodes in the hierarchy, and whose cluster head is a well known node called the *root* of the

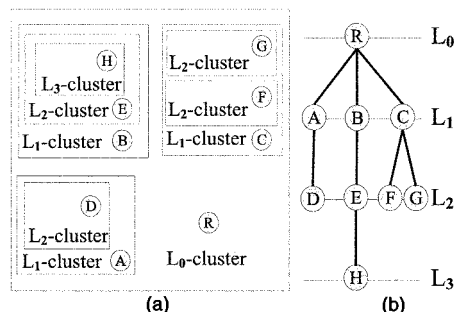


Fig. 1. Cluster hierarchy. (a) Clusters and layers. (b) Tree.

hierarchy, clusters are recursively divided into sub-clusters, until all clusters obtained are “singleton-clusters” containing only their cluster head.¹

The cluster hierarchy thus built forms a logical tree spanning all the cluster heads (e.g., all the nodes) in the hierarchy [see Fig. 1(b)]. Consequently, the state to build and maintain this hierarchy can be distributed among all the nodes in the hierarchy such that each node in layer L_i only needs to record its parent cluster head (i.e., the L_{i-1} -cluster head whose cluster it belongs to) and the L_{i+1} -cluster heads that are members of its own cluster. For instance, in Fig. 1(a), B records R as its parent cluster and E as its child.

B. Workings of the Algorithm

The algorithm is distributed and based solely on unicast communications. In other words, it does not rely on any special network support.

One of the central ideas in the algorithm is that any node (i.e., any cluster head) sees the rest of the world as a set of concentric rings (which we call *zones*), centered on the node itself. Each zone starts where the previous one finishes and the zones are numbered in increasing order, starting at 0 for the smallest ring (see Fig. 2). The actual size of each ring, as well as the distance measurement used to define it (e.g., delays, throughput, etc.), is unimportant for the general workings of the algorithm. With each zone, a distance called a *radius* is also defined. Again, the size of the radius is unimportant for the workings of the algorithm (but its distance measurement is the same as the measurements used for defining the zones).

The scalable hierarchical clustering algorithm works as follows. The cluster hierarchy is rooted on a well known entity called the *root*. A node desiring to join the cluster hierarchy first measures its distance to the root, and then sends to the root a JOIN message containing this distance. Based on this distance, the root determines the zone of the joining node. Here, two cases are possible.

¹Each node in the hierarchy is therefore the cluster head of a (sub-)cluster.

Manuscript received September 14, 2001. The associate editor coordinating the review of this letter and approving it for publication was Prof. D. Petr.

L. Mathy, S. Simpson, and D. Hutchison are with the Computing Department, Lancaster University, Lancaster LA1 4YR, U.K. (e-mail: laurent@comp.lancs.ac.uk; ss@comp.lancs.ac.uk; dh@comp.lancs.ac.uk).

R. Canonico is with the Computing Department, University Federico II, Napoli, Italy (e-mail: roberto.canonico@unina.it).

Publisher Item Identifier S 1089-7798(02)01919-1.

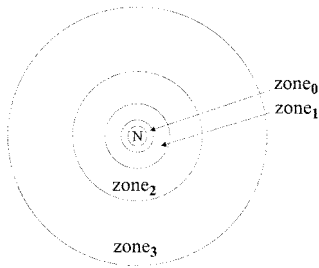


Fig. 2. Zones associated with a node.

- 1) The joining node is the first node joining in the corresponding zone.
- 2) Other nodes from the same zone have already joined.

In the former case, the root records the presence of the joining node in the corresponding zone and sends the node a `NEW_CLUSTER_ACK` message, indicating that the joining node has found its place in the hierarchy (this finishes the algorithm for the joining node). The joining node is now the cluster head of one of the sub-clusters dividing the cluster headed by the root (*albeit* a “singleton-cluster” for the time being).

In the latter case, the root sends to the joining node, in a `TRY` message, the list of the cluster heads in the same zone as the joining node, along with the radius associated with this zone. The joining node then measures its distance to each of the nodes in the list. Again, we consider two cases:

- 1) The distance of the joining node to at least one of the cluster heads in the list is smaller than the given radius. The clusters headed by these cluster heads are called *attracting clusters*.
- 2) The distance of the joining node to all the cluster heads in the list is greater than the given radius.

In the former case, the joining node chooses the closest attracting cluster and joins it: that is, the algorithm starts again with the corresponding cluster head acting as the root. In this case, the joining node is said to “go down one layer” (as the cluster it is heading will potentially be part of the partition of the attracting cluster) and it is important to note that the root does not record the presence of the joining node. In essence, from the root’s point of view, the members of the attracting cluster are “collapsed” into the attracting cluster head, as this cluster head is the only node in the attracting cluster remembered by the root.

In the latter case, the joining node creates a new sub-cluster by sending a `NEW_CLUSTER` message to the root (including its distance to the root). The root then keeps a record of the new cluster head (i.e., the joining node) and of its zone and replies with a `NEW_CLUSTER_ACK` message which finishes the algorithm for this joining node.

III. SCALABILITY CONSIDERATIONS

From the previous section, it should be clear that the state overhead imposed on each node in the hierarchy is proportional to the number of zones needed for that node to “span” its cluster, times the number of clusters per zone. This number of clusters per zone also influences the scalability of the join procedure,

as any joining node must measure its distance to all the cluster heads at the same zone, for all traversed layers. Also, the further away from the central node a zone is, the more nodes—and thus the more clusters—such a zone potentially contains. These observations favor the use of large clusters, within few zones.

On the other hand, large clusters tend to create many layers (as they can contain large sub-clusters which, in turn, will have to be divided), which has a negative impact on the latency of the join procedure.

In order to accommodate these conflicting requirements, we propose to define zones based on round-trip time (RTT) measurements, and whose sizes follow an “exponential distribution” (see Fig. 2):

$$\text{zone}_0: 0 < \text{dist} \leq 1 \quad (1)$$

$$\text{zone}_i: (1 + \Delta)^{i-1} < \text{dist} \leq (1 + \Delta)^i, \text{ with } \Delta > 0 \quad (2)$$

This, in turn, allows us to define the size (i.e., radius) of the clusters at zone_i as:

$$r_i = \frac{(1 + \Delta)^i - (1 + \Delta)^{i-1}}{2}. \quad (3)$$

The parameter Δ in the formulas could be either fixed or varied according to which layer the cluster, headed by the corresponding node, belongs to. Other size “distributions” for both zones and radii are of course possible, but the ones we propose prevent an explosion of the number of clusters in far zones while keeping the number of zones down and retaining the desirable property that “detail” (i.e., “fine grain positioning”) matters only for nodes close to a cluster head.

IV. PERFORMANCE EVALUATION

We have analyzed our algorithm by means of simulations. Both the NS-2 simulator [4] and the GT-ITM topology generator [5] were used to test scenarios involving groups of 500 members randomly distributed on topologies of 100 and 600 nodes, respectively (following a transit-stub model [5]). In these simulations, each group member (including the root) used the same value of Δ [see (2) and (3)] which was varied in unit steps from 1 to 5. Each scenario was repeated 20 times for each value of Δ .

In this letter, we are mainly interested in assessing the scalability of the proposed clustering algorithm. State overhead is of prime concern and Fig. 3 shows the maximum number of cluster heads “remembered” by any node in the hierarchy, as a percentage of the group size. This figure illustrates the influence of the value Δ in controlling the state overhead of the algorithm. For instance, for groups of 500 members in a network of 600 nodes, we see that the maximum state overhead at any cluster head can be reduced from 182 sub-clusters ($\approx 36\%$) when Δ is equal to 1, down to 36 sub-clusters ($\approx 7\%$) when Δ is equal to 5.

We also see that the efficiency of the algorithm increases with the group size, stabilizing the curves into gently decreasing slopes indicating a small increase in state as more members are added. For instance, in the scenario with 600 nodes and Δ of 5, the maximum state at any node increases from 30 (10%) for a group of 300 members to only 36 for a group of 500 members.

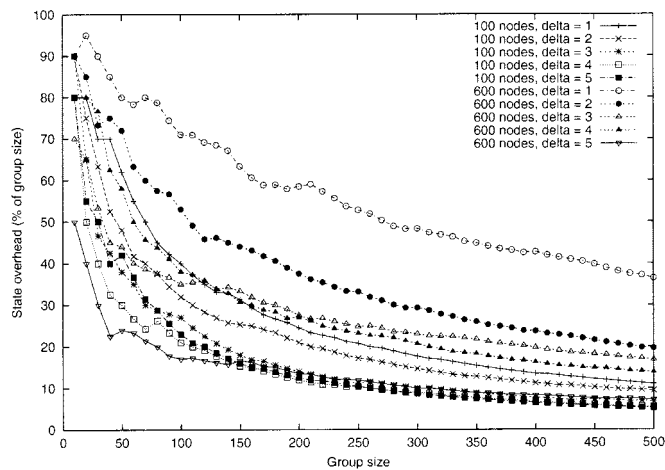


Fig. 3. Maximum state overhead.

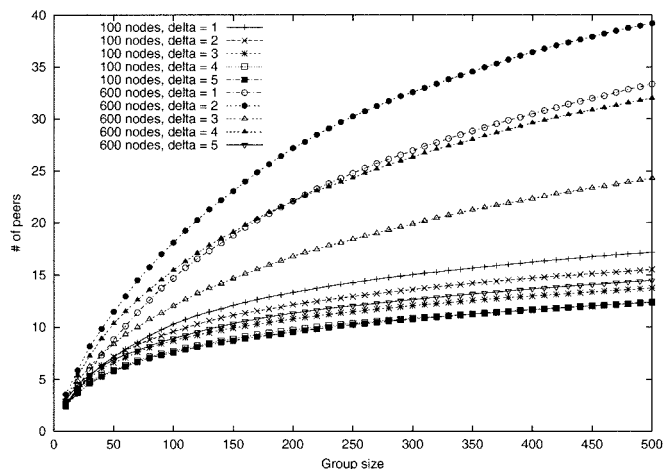


Fig. 4. Mean distance sampling overhead.

The number of cluster heads to which a new member measures its distance is also of prime concern, as this “sampling” has a direct bearing on the join latency and ultimately the scalability of the algorithm. This is because the message overhead, under static conditions, is proportional to the “sampling.” Fig. 4 depicts the average number of cluster heads that a newcomer samples during its join procedure. This figure shows that for judiciously chosen values of Δ , the increase in sampling overhead and thus in join latency is sublinear compared to the group size. This is, of course, an extremely desirable property of any scalable method and algorithm.

During these simulations, we have also observed that the maximum sampling by any joining node was as low as 42 peers. These results show that the maximum joining overhead observed by all newcomers is kept small.

V. DISCUSSION AND CONCLUSIONS

In this letter, we have proposed a method to build a hierarchy of nodes, based on the notion of proximity, in a distributed and scalable way. The hierarchy is built through a series of “local”

decisions involving only a small subset of the hierarchy’s population for each decision. This, coupled with an innovative adaptive cluster size distribution approach, yields a simple, yet powerful, approach to building overlay, application-level structures without relying on any special support from network routers.

The hierarchy thus built is loopless and spans all the nodes in it. Our scalable adaptive hierarchical clustering algorithm can, therefore, be seen as a new member in the category of application-level multicast tree building methods (e.g., [6]–[9]). The overlay application-level multicasting trees built with our scalable adaptive hierarchical clustering are unconstrained, meaning that nodes in the tree cannot explicitly control their number of children. This may not be a problem for overlay trees built for control purposes [1] but could yield a significant penalty for trees built for data distribution. However, the method presented in this letter can still be very useful in the context of application-level multicast data distribution.

Indeed, a constrained application-level multicast overlay tree can be built by having each cluster head and its sub-clusters (i.e., its members populating the next layer in the hierarchy) run any algorithm that builds a constrained overlay tree [6], [8], [9]. With this approach, each node in the cluster hierarchy would be a member of the overlay tree rooted at its parent cluster, as well as the root of the overlay tree spanning its own cluster. This would allow the building of very large constrained overlay trees.

Another application of the scalable adaptive hierarchical clustering presented in this letter is resource discovery. Indeed, a permanent hierarchy of resources could be built, rooted on a well known node, and “searched” by clients with a modified join procedure which does not declare the creation of a new (sub-)cluster when it finishes (see Section II-B). This could even substitute *expanding ring searches* in asymmetric network multicast circumstances or when network multicast is unavailable.

In future work, we will investigate the performance of the proposed algorithm under dynamic conditions (e.g., dynamic group membership, failures, etc.).

REFERENCES

- [1] B. Whetten and G. Taskale, “An overview of reliable multicast transport protocol II,” *IEEE Network Mag.*, vol. 14, no. 1, pp. 37–47, Jan. 2000.
- [2] M. Kadansky, D. M. Chiu, B. Whetten, B. Levine, G. Taskale, B. Cain, D. Thaler, and S. J. Koh. (2001, Mar.) Reliable multicast transport building block: Tree auto-configuration. IETF. [Online] Internet Draft draft-ietf-rmt-bb-tree-config-02, Work in progress
- [3] C. Diot, B. Levine, B. Lyles, H. Kassem, and D. Balensiefen, “Deployment issues for the IP multicast service and architecture,” *IEEE Network*, vol. 14, no. 1, pp. 78–88, Jan./Feb. 2000.
- [4] Ns-2 Network Simulator [Online]. Available: <http://www.isi.edu/nsnam/ns/>.
- [5] E. Zegura, K. Calvert, and S. Bhattacharjee, “How to model an internet network,” in *IEEE Infocom*, Mar. 1996, pp. 40–52.
- [6] Y.-H. Chu, S. Rao, and H. Zhang, “A case for end system multicast,” in *ACM SIGMETRICS*, Santa Clara, CA, June 2000, pp. 1–12.
- [7] J. Jannotti, D. Gifford, K. Johnson, F. Kaashoek, and J. O’Toole, “Overcast: Reliable multicasting with an overlay network,” in *USENIX OSDI*, San Diego, CA, Oct. 2000.
- [8] D. Pendarakis, S. Shi, D. Verma, and M. Waldvogel, “ALMI: An application level multicast infrastructure,” in *3rd USENIX Symp. on Internet Technologies*, San Francisco, CA, Mar. 2001.
- [9] L. Mathy, R. Canonico, and D. Hutchison, “An overlay tree building control protocol,” in *Proc. Int. Workshop on Networked Group Communication (NGC)*, Nov. 2001, pp. 76–87.