Automatic Clustering for Self-Organizing Grids

Weishuai Yang, Nael Abu-Ghazaleh and Michael J. Lewis
Department of Computer Science, Binghamton University
{wyang, nael, mlewis}@cs.binghamton.edu

Abstract

Computational grids have not scaled effectively due to administrative hurdles to resource and user participation. Most production grids are essentially multi-site supercomputer centers, rather than truly open and heterogeneous sets of resources that can join and leave dynamically, and that can provide support for an equally dynamic set of users.

Large-scale grids containing individual resources with more autonomy about when and how they join and leave will require self-organizing grid middleware services that do not require centralized administrative control. This paper considers one such service, namely the dynamic discovery of high-performance variable-size clusters of grid nodes. A brute force approach to the problem of identifying these "ad-hoc clusters" would require excessive overhead in terms of both message exchange and computation.

Therefore, we propose a scalable solution that uses a delay-based overlay structure to organize nodes based on their proximity to one another, using a small number of delay experiments. This overlay can then be used to provide a variable-size set of promising candidate nodes than can then be used as a cluster, or tested further to improve the selection. Simulation results show that this approach results in effective clustering with acceptable overhead.[1]

1 Introduction

Although a number of computational grids have begun to appear, truly large-scale "open" grids have not yet emerged or been successfully deployed. Current production grids comprise tens, rather than hundreds or thousands, of sites [1, 3]. The primary reason is that existing grids require resources to be organized in a structured and carefully managed way, one that requires significant administrative overhead to add and manage resources. This overhead is a significant barrier to participation, and results in grids comprising only large clusters and specialized resources; manually adding individual resources—especially if those resources are only intermittently available—becomes infeasible and unworthy of the effort required to do so.

An alternative model for constructing grids [4] lowers the barrier for resource and user participation by reducing various administrative requirements. In this *Self-Organizing Grids* (SOGs) model, resource owners would directly and dynamically add their resources to the grid. These resources may include conventional clusters that permanently participate in the grid, or that are donated by providers during off-peak hours. In addition, users may provide individual resources in much the same way that they add them to peer-to-peer networks and public resource computing projects such as SETI@home [2]. The grid would then consist of the currently participating resources. We envision SOGs that contain different tiers of resources, ranging from always connected large clusters, to individual PCs in homes, down to small-scale sensors and embedded devices. Thus, SOGs represent the intersection of peer-to-peer computing, grid computing, and autonomic computing, and can potentially offer the desirable characteristics of each of these models.

Constructing grid services that can operate in, let alone take advantage of, such an environment requires raises a number of challenges and requires different algorithms and designs [4]. This paper considers one of the primary challenges, namely how to automatically discover efficient clusters within SOGs to enable effective scheduling of applications to resources in the grid. A candidate collection of SOG nodes may not necessarily be a physical cluster of co-located machines under a single administrative domain connected by a high-speed network; but the nodes' proximity to one another—in terms of network connection performance characteristics—may allow them to *serve* as an *ad hoc* cluster to support some applications. A brute force approach to the problem of discovering *ad hoc* clusters would periodically test network performance characteristics between all pairs of resources in the grid. Clearly, this approach is not feasible for a large scale system; more scalable approaches are needed.

[1]This material is based upon work supported by the National Science Foundation under Grant ACI-0133838, CNS-0454298, and by the Air Force Research Laboratories under contract FA8750-04-1-0054.

1-4244-0328-6/06/$20.00

The need for clustering arises in P2P environments, where it has received significant research attention [8, 13, 5, 9]. In P2P environments, clusters are needed for scalability of document search and exchange. Clusters are created and maintained in a large and dynamic network, where neither the node characteristics nor the network topology and properties (such as bandwidth and delay of edges) are known a priori. To improve performance, cluster nodes must be close enough to one another, and must typically fulfill additional requirements such as load balancing, fault tolerance and semantic proximity. Some of these properties are also desirable for our system. However, the emphasis on proximity is much more important to SOGs, since the computational nature of grid applications may require close coupling. Further, to allow flexible application mapping, variable size clusters must be extractable; in contrast, the emphasis in P2P networks is usually on finding clusters of a single size.

Clustering in SOGs is more complicated than classical dominating set and p-center problems from graph theory, which are themselves known to be NP-complete. Simple strategies such as off-line decisions with global knowledge do not work because of the large scale and dynamic nature of the environment. Further, the importance of cluster performance (because of its intended use), along with the requirement to create variable size clusters, suggest the need for different solutions. An optimal solution that measures the quality of connections between all pairs of nodes, and that then attempts to extract the optimal partition of a given size, requires $O(n^2)$ overhead in the number of messages to measure the connections, and an NP-complete optimal clustering solution. Further, the dynamic nature of the problem in terms of the network graph and processor and network loads requires lighter weight heuristic solutions. The problem and related work are presented in Section 2.

In this paper, we propose a scalable solution to automatic clustering in SOGs. A flexible overlay structure, called a Minimum-delay Dynamic Tree (MDTree), is built and maintained to allow an initial sorting of the nodes based on a small number of delay experiments for every joining node. The MDTree organizes nodes as they join, keeping nearby nodes close together in the tree. As nodes join, a grouped set of nodes may exceed the allowed threshold as nodes and the group must be split. We show that effective partitioning when splits occur is critical to the performance of the approach; because the problem is NP-complete, we use a genetic algorithm for bi-partitioning. We describe the organization and maintenance of an MDTree in Section 3.

The MDTree overlay structure is then used when users generate requests for clusters, to identify effective clusters of a given size efficiently; Section 4 describes variable size on-demand clustering using MDTrees. As a result, it becomes possible to find clusters of specified sizes with low average delay among the nodes. We use simulation to study the performance of this approach, and describe the results of this study in Section 5. We show that by using an MDTree, the message overhead for finding a cluster can be kept linear with respect to cluster size, and the average link delay within the formed cluster is close to optimal.

We note that traditional computational grids that comprise multiple physical clusters may still benefit from an automatic clustering approach like the one we describe. In particular, when a large-scale application requires a set of machines that exceeds the size of the largest available cluster, our approach will consider the delay between nodes at different sites, and can help identify a large multi-organizational collection of machines to support the application. We discuss our conclusions and future work in Section 6.

2 Problem Definition and Related Work

To support general large-scale parallel processing applications, SOGs must self-organize in a way that allows effective scheduling of offered load to available resources. When an application request is made for a set of nodes, SOGs should be able to dynamically extract a set of resources to match the request. Since these resources are often added separately and possibly by multiple providers, SOGs should be able to identify and track relationships between nodes. In addition, to support effective scheduling, the state of resources in the grid must be tracked at appropriate granularity and with appropriate frequency.

An important initial question is "What represents an effective cluster?" Clearly, the capabilities of the individual nodes are important. However, the influence of communication often has a defining effect on the performance of parallel applications in a computational cluster. Moreover, it is straightforward to filter node selection based on node capabilities, but it is much more challenging to do so based on communication performance, which is a function of two or more nodes. In this paper, we assume that all SOG nodes are capable of participating in clusters, and focus on cluster selection based on communication performance. In future work, we plan to investigate how our approach can be extended to consider both communication and computation characteristics of candidate cluster nodes.

The automatic clustering challenge is to extract the structure of the SOG from a performance perspective; out of the unorganized or partially organized set of SOG resources, how can the structure that is available to conventional grids be dynamically and automatically discovered? The proposed solution is to create a hierarchy within the system and to localize most of the interactions to a small number of nearby nodes. The base problem in constructing the overlay that exposes structure is how to use distributed algorithms to

organize the nodes into leaders and peers according to some performance-related criteria, without global knowledge. A description of some clustering work that has been done in the P2P community follows:

Highways [8] presents a basic solution for creating clusters through a beacon-based distributed network coordinate system. Such an approach is frequently used as the basis for other P2P clustering systems. Beacons define a multi-dimensional space with the coordinates of each node being the minimum hop-count from each beacon (computed by a distance vector approach or a periodic beacon flood). Distances between nodes are measured as Cartesian distances between coordinates. *Highways* serves as the basis for several other clustering approaches. Shortcomings include the fact that the distance in the multi-dimensional space may not correspond to communication performance, and that markers must be provided and maintained.

Agrawal and Casanova [5] describe a pro-active algorithm for clustering in P2P networks. They use distance maps (multi-dimensional coordinate space) to obtain the coordinates of each peer, and then use a marker space (not the same concept as in Highway) as the cluster leader by using the K-means clustering algorithm. The algorithm chooses the first marker (leader) randomly, then repeatedly finds a host of distance at least D from all current markers, and adds it into the marker set. Nodes nearest to the same marker are clustered together, and are split if the diameter becomes too large. This strategy results in message flooding and its associated high overhead.

Zheng et. al. [13] present T-closure and hierarchical node clustering algorithms. The T-closure algorithm is a controlled depth-first search for the shortest paths, based on link delay. Each node learns all shortest paths starting from itself, with distance not larger than T. The hierarchical clustering algorithm uses nomination to select a supernode within some specified distance. These two strategies require high overhead and do not support node departure.

Xu and Subhlok describe automatic clustering of grid nodes [9] by separating the clustering problem into two different cases. Their approach uses multi-dimensional virtual coordinates to cluster inter-domain nodes, and uses n^2 direct measures to cluster intra-domain nodes. This strategy can classify existing nodes into clusters according to physical location, but cannot extract variable sized clusters according to user requirements.

3 Overlay Pre-Clustering with Minimum-delay Dynamic Trees

We classify clustering algorithms into two categories: pro-active and on-demand. Most existing algorithms are pro-active; that is, given a set of nodes that join dynamically, the goal is to organize them into clusters as they join. On-demand systems do not maintain clusters in advance but construct them from scratch when required. However, we intend to support SOGs whose diverse applications may lead to users requesting clusters of various sizes. Therefore, either different size clusters must be built pro-actively (significantly increasing the overhead), or an on-demand approach must be employed. A purely pro-active system results in high overhead and inflexibility, whereas a purely on-demand system requires significant dynamic overhead that can introduce scheduling delay. The approach that we take in this work is to pro-actively organize the nodes into an overlay that makes on-demand construction of variable size clusters efficient.

The problem of finding an optimal variable size cluster is NP-complete [13]; $O(n^2)$ delay experiments (ping messages) are needed to collect the full graph edge information. Therefore, the objective of our approach is to find an approximation of the optimal solution. Thus, adaptive heuristic approaches that can provide efficient solutions with more acceptable overhead in terms of communication and runtime are desired.

We use the work by Banerjee et. al. [6] as our starting point for hierarchically arranging peers in tiers, and extend it for more effective operation with respect to computational clustering, and to enable dynamic cluster formation. The tree is maintained dynamically as nodes join and leave. To better balance the tree, we use a genetic algorithm to partition groups of nodes under a common parent (i.e. neighborhoods of a super-node). This enables the tree to maintain relatively small groups of mutually close nodes.

Our approach is to pre-cluster the nodes using an overlay organization that we call a Minimum-Delay Tree (MDTree). Nearby nodes in the tree have small delay to each other; thus, on-demand variable size clustering considers only a small subset of the nodes. Each level in an MDTree consists of a neighborhood in which each node is a representative of another neighborhood at a lower level, recursively down to the leaf nodes. Inter-node delays among nodes within the same neighborhood are relatively small.

An MDTree makes it easier to find a specified number of nodes with minimum average delay. By using a hierarchical tree overlay structure, MDTree controls the complexity of node joins and cluster extraction to $O(Log_k N)$, where K is the size of neighborhood on each layer in the tree and N is the number of nodes.

3.1 MDTree Architecture

An MDTree employs a hierarchically layered tree structure. Nodes on the same branch of the tree are organized such that they are close to one another in terms of link delay. This structure helps requests to be satisfied with clusters that have small internal average link delays.

We start by describing the terminology used in this description.

- **MDTree:** All the SOGs nodes are organized in a structure that facilitates resource sharing, information exchange, and cluster formation. This structure is the MDTree.

- **Layer:** All nodes at distance j edges from the root of the MDTree are said to be at layer $L_(H - j)$ of the tree, where H is the height of the tree. Total number of layers in an MDTree is approximately $O(Log_k N)$, where N is the total number of nodes in the tree, and K is predefined neighborhood size, which is defined below.

- **Peer-node:** Any participating node is a peer-node.

- **Super-node:** A super-node is the leader of a neighborhood. "Super-node" and "peer-node" are relative concepts. A node can be a peer-node on one layer, and a super-node on another. The super-node of a lower layer neighborhood is also a participant in the neighborhood of the above layer. In other words, every node on layer L_i is a super-node on layer L_{i-1}. On the other side, a super-node of layer L_{i+1}, must be a super-node for exactly one neighborhood in layers L_1 through L_i. Super-nodes are key nodes in the structure; they controlls peer-nodes in its neighborhood, and they are the gateway to the outside of the neighborhood.

- **Neighborhood:** A neighborhood consists of a super-node and all other controlled nodes on a specified layer. Numerous neighborhoods controlled by different super-nodes exist on a specified layer. Lower layers than the layers above them. On each layer, nodes within the same neighborhood exchange information with each other, which helps electing new super-node when the current super-node is missing. However, nodes on the same layer but under the controll of different super-node, i.e., belonging to different neighborhoods, do not directly communicate, and they do not know the existence of one another.

- **Community:** A community consists of a super-node and the subtree comprising all the neighborhoods on lower layers controlled by that super-node.

- **Entry Point:** A special super-node used to direct new joining nodes to the neighborhood on the highest layer. The entry point is the super-node on the highest layer, and the only participant in this layer.

- **K:** Each neighborhood has a pre-set maximum number of nodes that it can contain; this maximum value, K, is currently a constant of the overlay. Once a neighborhood on layer L_i grows to contain K nodes, the neighborhood eventually splits into two, and the newly generated super-node is promoted into layer L_{i+1}. A split may happen immediately after a neighborhood grows to contain K nodes, or at a specified interval, depending on the implementation.

Figure 1 depicts an example of an MDTree consisting of 16 nodes with $K = 4$. Super-nodes keeps information of number of nodes it is controlling on each level, as is shown in the figure, and, in its neighborhood on each layer, the number of nodes controlled by each of its peer-nodes. This information is useful for cluster formation.

MDTree construction and maintenance consist of four components: (1) the Node Join Protocol governs how nodes join the tree; (2) Neighborhood Splitting splits a neighborhood into two neighborhoods, when its size exceeds K; (3) a Tree Adjustment process allows nodes to move to more appropriate layers if they get misplaced by the neighborhood splitting process (or otherwise, for example, as nodes leave); and (4) Tree Maintenance mechanisms maintain the tree as nodes leave, by promoting nodes if their super-node leader disappears. We discuss these aspects of the design in the remainder of this section.

3.2 Node Join Protocol

To join the MDTree structure, a new node first queries the Entry Point, which replies with a complete list of top layer nodes. Then the node pings each node in the returned list. As a result of the pings, it finds out the closest node and sends a query to it. From this node, it gets a list of its neighborhood at the lower level. The process is repeated recursively until a layer L_1 node is found; the joining node then attaches itself on layer L_0 to the found node. When nodes join the system, they are always initially attached to layer L_0. Once a neighborhood consists of K nodes, it must eventually be split. Higher layer nodes result from layer splitting.

3.3 Neighborhood Splitting

An MDTree's layer structure is dynamic, with layers and super-nodes potentially changing roles and positions when nodes join and leave. When one super-node's number of children reaches K, this neighborhood is split into two. The layer splitting algorithm has significant impact on the performance of the tree; a random split may cause ineffective partitioning, as relatively distant nodes get placed in the same layer. The effect is compounded as additional splits occur. Ideally, when a split occurs, the minimum delay criteria of the tree would be preserved. In other words,

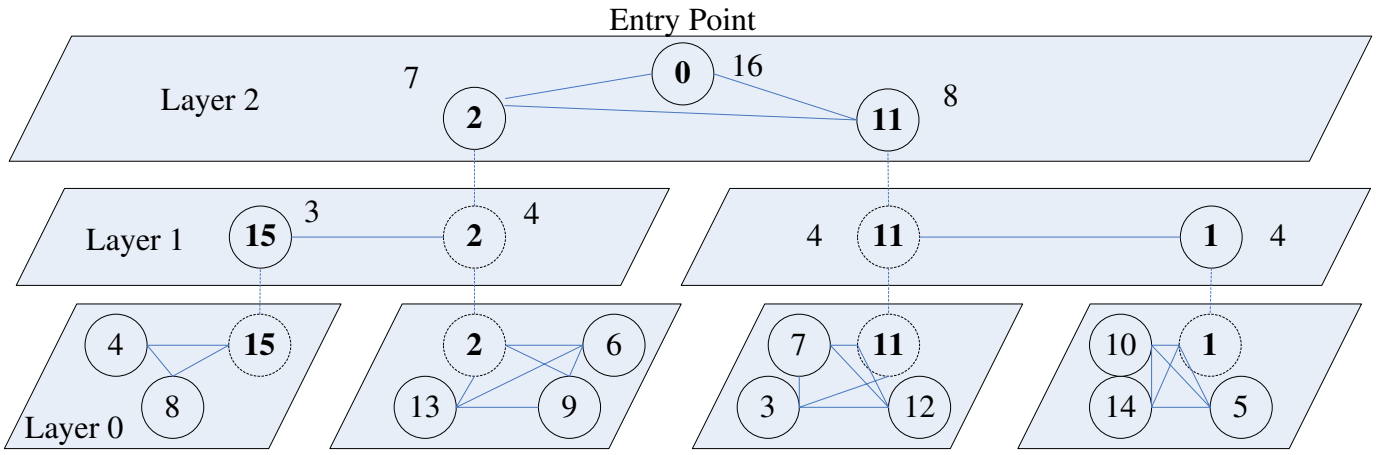

Figure 1. An MDTree of 16 nodes with maximum neighborhood size of $K = 4$. Each super-nodes is in bold, and is labelled with the number of nodes it controls.

the average link delay for each new neighborhood should be minimized.

Because of previous information exchange within the neighborhood, the super-node has all the information about its peer-nodes, including their distance to each other; the presence of this information allows the super-node to effectively partition the neighborhood. Effective partitioning of the neighborhood is critical to the performance of the MDTree. However, optimal bi-partitioning is known to be NP-complete, and it is impractical to enumerate all the combinations and calculate average link delays for each of them when K is relatively large. For this reason,we use an optimized genetic partitioning algorithm to achieve effective partitioning. Section 3.3 describes this algorithm. However, any heuristic that can efficiently and effectively partition the neighborhood may be used here. We discuss this in Section 5.1.

After nodes are partitioned into two new smaller neighborhoods, just for simplicity and avoiding update this node on all above layers, the super-node N_a of the neighborhood at layer L_i remains the super-node of the neighborhood to which it belongs after the split. Here N_a continues to be a super-node because it may reside on a higher layer. After splitting, if N_a is found to be not the best suitable super-node on that layer, it can be replaced be the best fit node. In the newly generated neighborhood, N_b, the node having the minimum link delay to all other nodes, is appointed to be the super-node of layer L_i. (However, for perfection, both super-nodes can be selected like the newly generated neighborhood.) Now both N_a and N_b will participate in the same neighborhood on layer L_{i+1} under the same super-node N_c. Now N_b, the new super-node, becomes its sibling and a new peer-node of N_c. N_a informs all related nodes about the change of leadership. Upon receiving the split message, the new super-node N_b requests to attach to layer L_{i+1} and join N_c's neighborhood. N_c, the super-node of N_a now becomes the common super-node of both N_a and node N_b.

Such a split reduces the number of nodes on layer L_i, but increases the number of nodes on layer L_{i+1}. If the number of nodes in a neighborhood on layer L_{i+1} reaches K, that neighborhood splits.

3.4 Tree Adjustment

In general, heuristic approaches do not necessarily consider the full solution space, and can therefore result in sub-optimal configurations. For example, a node may unluckily get placed in the wrong branch of a tree due to an early split. Further, neighborhood splitting results in MDTree structure changes, and nodes being promoted to higher layers. However, this may separate nearby nodes into different neighborhoods, and they may eventually migrate away from each other in the tree. Heuristics may allow nodes to recover from such unfortunate placement. For example, a node can through its super-node at layer L_i discover the super-node's neighborhood on L_{i+1}. The node can then ping all the nodes in that neighborhood at a fixed infrequent interval to check for a peer of lower link delay, and move itself into that neighborhood (and merge all of its community into the new community).

3.5 Tree Maintenance

It is important to recover from node and super-node failure (or more commonly, departure from the SOG). We note that in a SOG, most nodes may be well behaved and announce their intent to depart. This may allow soft reconfiguration of the tree, by removing the peer-node and electing an alternate super-node for the layers where it serves this duty. The tree provides an efficient structure for multicasting such messages. However, since failures and unannounced departures are possible we also have nodes in the same neighborhood exchange *heartbeat* messages. A node is considered absent if it fails to respond to some predefined number of consecutive heartbeat messages; this can trigger tree reconfiguration. Recovery from peer-node departures is handled differently from recovery from super-node departures, as described below.

- **Peer-node Departure:** The departure of a peer-node P simply results that the super-node and other peer-nodes in P's neighborhood remove P from their records. If the number of nodes in the neighborhood falls below a predefined threshold, the super-node of layer L_i may try to demote itself to a peer-node on layer L_i, and join its entire community into that of another super-node on layer L_i. This approach can keep the tree structure balanced; however, we do not explore MDTree shrinking in this paper.

- **Super-node Departure:** Because all MDTree structure information is broadcast within the neighborhood, all peer nodes have the knowledge of the neighborhood. Thus, a new super-node can be elected directly from the neighborhood and promoted in place of the departed super-node.

4 Cluster Formation

When user on a node requests a cluster of size R, it checks if the number of nodes it controls is larger than the requested size multiplied by a predefined candidate scale factor S, where $S > 100\%$ so that the requester may select the R most suitable nodes from among a set of more than R nodes should it decide to do so. If it cannot satisfy the request, the request is forwarded recursively to super-nodes at higher and higher layers without DETERMINED flag, until it arrives at a super-node that controls a community that contains more than $R * S$ nodes. This super-node then decides which part of the community under its control should join the cluster, and forwards the request, with DETERMINED flag being set, to those nodes. A cluster request message with DETERMINED flag requires the receiver and all the nodes controlled by the receiver to respond to the original requester with no further judgement.

After receiving enough responses from cluster candidates, the requester can then ping each responder and select the closest R nodes; or, the cluster can choose to select a random subset of R nodes, or the first R responders. In our current implementation, we set a default value of $S = 180\%$, so requesters receive 1.8 times as many candidate nodes for their cluster as they request, and the requester picks top R responders with least link delay to itself, thus leads to a solution favoring minimum diameter.

The original requester only knows the link delay between itself and responders, but not the delay among responders. This is another sacrifice of optimality for performance; a perfect selection would require a solution to the NP-complete clustering problem, and $O(n^2)$ tests (here, however, n reflects the size of the cluster, not the much larger size of the SOG). In the future, we will explore alternative heuristics for final selection of the cluster from among candidate nodes.

In this paper, we assume that the requester is interested in a nearby cluster, which reduces the application launch delay, and acts as a crude geographical load-balancing technique. However, alternative approaches for cluster formation can be directly supported on top of an MDTree. For example, the tree can track the load at a coarse granularity, and map the request to a lightly loaded portion of the SOG.

5 Experimental Evaluation

We conducted simulation experiments to evaluate the proposed approach using the GPS simulation framework [10, 11] and Transit-Stub networks generated from the GT-ITM topology generator [12]. We extended GPS to model MDTrees, and to support our cluster formation algorithm. The topology studied consists of 600 nodes (due to run-time and memory usage considerations). Link delay within a stub is 5 milliseconds (ms), between stubs and transits it is 10 ms, and between transits is set to 30 ms. We evaluate cluster requests of sizes 8, 16, 32, 64, 128, and 256 nodes. Pings are used to determine the link delay between node pairs. In our experiment, we used 25 as the value of K for the MDTree, and 180% as candidate scale factor S.

We use the following metrics to measure the quality of the cluster that an MDTree helps discover:

- **Average link delay among nodes within the cluster:** The average link delay is likely to be the most important criterion for the quality of the clustering, especially for fine-grained applications. Such applications require frequent communication among nodes within the cluster and their performance is bound by the latency of communication.

- **Maximum link delay to the cluster requester:** This criteria is important for clusters in which the most fre-

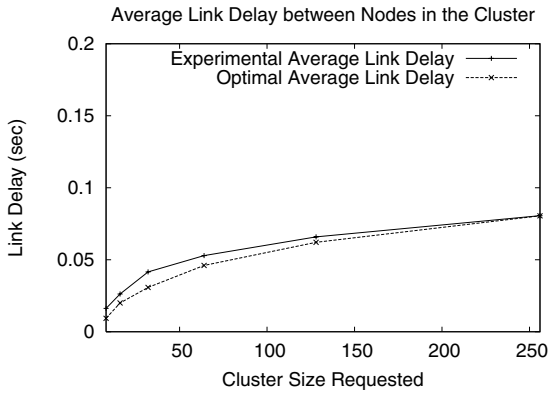

Figure 2. Average Link Delay in the cluster

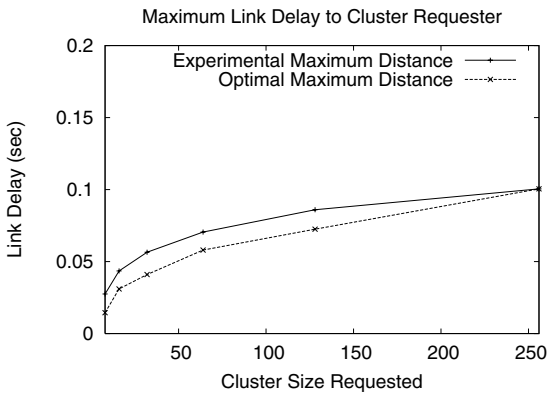

Figure 3. Maximum Link Delay to the cluster requester

quent communication is between the cluster requester and the other nodes.

- **Cluster diameter:** The largest link delay between any pair of nodes in the cluster.
- **Cluster Formation Overhead:** The overlay performance is measured by the number of messages sent during the process of requesting a cluster. These messages include cluster request messages, cluster responses, pings, and cluster confirmation. Since the MDTree is constructed pro-actively, the cost is amortized over all the requests generated for clusters; it can be considered as a fixed cost. New node joining only cost approximately $O(Log_k N)$ messages, which includes attach queries and pings, where, again, N is the number of nodes in the SOG, and K is the maximum number of nodes in any neighborhood.

Figure 2 shows the average link delay in the extracted cluster, compared to the optimal cluster for the topology (found through exhaustive search). The average delay, in general, is quite good compared to the optimal available. However, especially at small size clusters (smaller than the layer size), the quality of the solution can be improved. We believe that this argues for supporting mechanisms to allow nodes to change their location in the tree if they are not placed well. We note that at 256 nodes, the large size of the cluster relative to the topology size may contribute to the two graphs converging.

Figures 3 and 4 show the maximum link delay to the cluster requester, and the cluster diameter respectively. These figures contain a similar result to that of Figure 2. In general, the results show that the proposed approach performs well with respect to the optimal solution according to all three metrics.

The complexity of the clustering stage (i.e. the messages that are exchanged after a cluster is requested, as op-

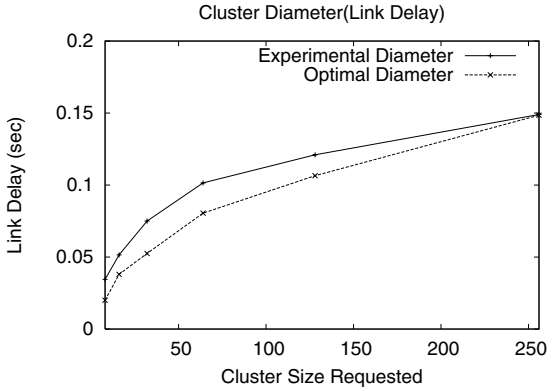

Figure 4. Cluster Diameter

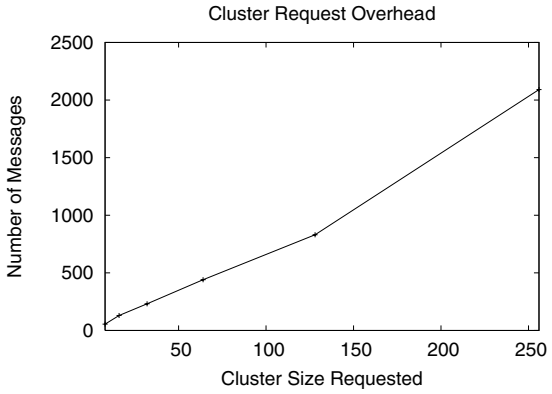

Figure 5. Cluster Requesting Overhead, messages include requests, responses, pings, and cluster confirms

posed to the pro-active MDTree setup costs associated with join messages) depends on the options used in representing the clusters and the MDTree structure (e.g. the values of S and K described earlier). Figure 5 shows that the overhead for requesting a cluster appears to be linear with the size of the requested cluster. Building and maintaining the MDTree structure also requires overhead. Node joining costs approximately $O(Log_k N)$ messages and pings. However, the main overhead comes from the periodic *heartbeat* messages, since it broadcasts to each nodes in the neighborhood. This overhead can be reduced by piggybacking and merging update messages.

5.1 Discussion

Clearly, super-nodes and regular peer-nodes have different levels of responsibilities in MDTrees. A super-node is a leader on all layers from 1 to the second highest layer it joins. Each super-node must participate in query and information exchange on all the neighborhoods it joins, which can make it heavily burdened. However, if higher layer super-nodes did not appear within neighborhoods at lower layer neighborhoods, it would be inefficient to pass information down to neighborhoods at lower layers.

Graph bi-partitioning is known to be NP-complete [7]. In an MDTree, we used a genetic algorithms for neighborhood splitting. Our algorithm generates approximately optimal partitioning results within hundreds or thousands of generations, which is a small number of computations compared to the NP-complete optimal solution (and these computations take place locally within a super-node, requiring no inter-node messages). Figure 6 shows that our genetic algorithm (or any other effective bi-partitioning heuristic) has a significant impact on the quality of the solution when compared to random partitioning for neighborhood splitting.

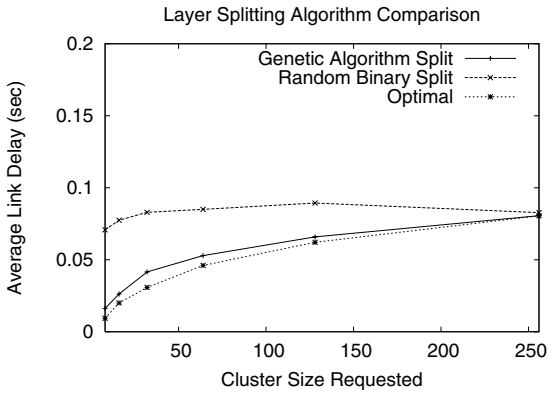

Figure 6. Comparison of Genetic Split and Random Split

6 Summary and Future Work

This paper introduces an efficient data structure and algorithm for implementing automatic node clustering for self-organizing grids, which will contain clusters of high performance "permanent" machines alongside individual intermittently available computing nodes. Users can ask for an "ad hoc" cluster of size N, and our algorithm will return one whose latency characteristics come close to those of the optimal such cluster. Automatic clustering is an important service for SOGs, but is also of interest for more traditional grids, whose resource states and network characteristics are dynamic (limiting the effectiveness of static cluster information), and whose applications may require node sets that must span multiple organizations.

Our MDTree structure organizes nodes based on the link delay between node pairs. The proposed approach is distributed, scalable, efficient, and effective. We use a genetic algorithm for neighborhood splitting to improve the efficiency and effectiveness of partitioning.

Significant future work remains in many areas. First, we plan to further study the proposed solution and the impact of parameters and design decisions within it. In particular, the impact of MDTree parameters such as the neighborhood size, the heartbeat frequency, and the neighborhood splitting algorithm may have significant influence on overhead and performance.

Previous sections of this paper mention several other areas for future exploration; we have implemented some of them, but have not yet evaluated them in detail. Examples include tree optimization to revisit placement decisions, the effect of node departure on clustering, and re-balancing to recover from incorrect placement decisions. Future work also includes using multiple criteria to identify candidate cluster nodes, instead of just inter-node delay. These criteria could include computing capabilities and current load, and the measured bandwidth between nodes.

We envision tiered SOG resources, ranging from conventional clusters that are stable and constantly available, to user desktops that may be donated when they are not in use. This variation in the nature of these resources can be accounted for, both in the construction of the MDTrees (e.g., by associating super-nodes with stable nodes) and during the extraction of clusters (e.g., by taking advantage of known structure information like the presence of clusters, instead of trying to automatically derive all structure).

Significant challenges also remain to make the SOG model feasible. These include the ability to do resource monitoring for co-scheduling in SOGs. Resource monitoring and co-scheduling have significant overlap with automatic clustering; their intersection is an interesting area of

research. Effective SOG operation also requires service and application deployment, fault tolerance, and security.

References

[1] Enabling grids for e-science (egee). http://public.eu-egee.org.

[2] Seti@home. http://setiathome.berkeley.edu.

[3] Teragrid. http://www.teragrid.org.

[4] N. Abu-Ghazaleh and M. J. Lewis. Short paper: Toward self-organizing grids. In *Proceedings of the IEEE International Symposium on High Performance Distributed Computing (HPDC-15)*, pages 324–327, June 2006. Hot Topics Session.

[5] A. Agrawal and H. Casanova. Clustering hosts in p2p and global computing platforms. In *The Workshop on Global and Peer-to-Peer Computing on Large Scale Distributed Systems*, Tokyo, Japan, April 2003.

[6] S. Banerjee, C. Kommareddy, and B. Bhattacharjee. Scalable peer finding on the internet. In *Global Telecommunications Conference, 2002. GLOBECOM '02*, volume 3, pages 2205–2209, November 2002.

[7] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1979.

[8] E. K. Lua, J. Crowcroft, and M. Pias. Highways: Proximity clustering for scalable peer-to-peer network. In *4th International Conference on Peer-to-Peer Computing (P2P 2004)*, Zurich, Switzerland, 2004. IEEE Computer Society.

[9] Q. Xu and J. Subhlok. Automatic clustering of grid nodes. In *6th IEEE/ACM International Workshop on Grid Computing*, Seattle, WA, November 2005.

[10] W. Yang. General p2p simulator. http://www.cs.binghamton.edu/~wyang/gps.

[11] W. Yang and N. Abu-Ghazaleh. Gps: A general peer-to-peer simulator and its use for modeling bittorrent. In *Proceedings of 13th Annual Meeting of the IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS '05*, pages 425–432, Atlanta, GA, Sept 2005.

[12] E. W. Zegura, K. L. Calvert, and S. Bhattacharjee. How to model an internetwork. In *IEEE Infocom*, volume 2, pages 594–602, San Francisco, CA, March 1996. IEEE.

[13] W. Zheng, S. Zhang, Y. Ouyang, F. Makedon, and J. Ford. Node clustering based on link delay in p2p networks. In *2005 ACM Symposium on Applied Computing*, 2005.