# Tree-based Overlay Networks for Scalable Applications

Dorian C. Arnold, Gary D. Pack, and Barton P. Miller
Computer Sciences Department
University of Wisconsin
1210 Dayton Street
Madison, WI, U.S.A. 53706-1685
Email: {darnold, pack, bart}@cs.wisc.edu

## Abstract

*The increasing availability of high-performance computing systems with thousands, tens of thousands, and even hundreds of thousands of computational nodes is driving the demand for programming models and infrastructures that allow effective use of such large-scale environments. Tree-based Overlay Networks (TBŌNs) have proven to provide such a model for distributed tools like performance profilers, parallel debuggers, system monitors and system administration tools.*

*We demonstrate that the extensibility and flexibility of the TBŌN distributed computing model, along with its performance characteristics, make it surprisingly general, particularly for applications outside the tool domain. We describe many interesting applications and commonly-used algorithms for which TBŌNs are well-suited and provide a new (non-tool) case study, a distributed implementation of the mean-shift algorithm commonly used in computer vision to delineate arbitrarily shaped clusters in complex, multi-modal feature spaces.*

## 1. Introduction

Today, there exist many high-performance computing systems – MPPs, and computational clusters and constellations – with thousands, or even tens or hundreds of thousands, of interconnected nodes [1]. The trend toward extremely large computing environments is expected to continue, making it imperative that we study efficient ways for tools and applications to run effectively in such environments.

The scalability challenge comes in three forms: dissemination of control information, data collection, and data analysis or computation. As the number of nodes in the system increases, both the computational and communication demands on the system increase – typically in a linear fashion. Without explicit attention, these increased demands cause tools to scale poorly.

Tree-based overlay networks (TBŌNs) have proven to provide a powerful parallel programming model that addresses scalability and efficiency in many tools [4, 11, 21–23, 25, 26]. A TBŌN (described more fully in Section 2.1) is a hierarchical organization of processes that use network transport protocols, like TCP, to implement data multicast, gather and reduction services. Applications explicitly insert application-level code into the TBŌN's *communication processes* using *filters*, which may execute data reduction operations on in-flight messages. Essentially, TBŌNs provide an effective platform for data communication since tree-based data communication scales logarithmically with the number of processes in the network. However by allowing computations to be distributed across the tree-structure, TBŌNs are also well-suited for scalable *data aggregation* or *reduction* as information propagates toward the root of the tree. Again, in tree-based networks, data reduction overheads vary logarithmically with respect to the total number of processes.

The TBŌN model of distributed computing is surprisingly flexible and can be generalized to support a wide range of usages beyond basic operations, like `min`, `max`, `count`, and `average`. Examples of more complex tree-based computations include distributed clock skew detection, time-aligned data aggregation, graph merging algorithms, and creating equivalence classes and data histograms [23, 24].

The features that make TBŌNs effective for a wide variety of distributed tools and applications:

- extensible data reductions and synchronizations,
- high-throughput and low-latency dataflow,
- customizable topologies, and
- a flexible data communication model.

1-4244-0054-6/06/$20.00 

Such generalization helps developers create scalable and efficient systems that easily can be extended in ways not envisioned during the original design.

The goal of this study is to demonstrate the broader applicability of TBŌNs outside the tool domains for which they have typically been used. We show that large classes of useful applications and common algorithms, particularly those that reduce to an *equivalence relation computation*, can be made scalable through straightforward TBŌN integration. We present a new, non-tool case study of TBŌN scalability – a distributed version of the *mean shift* algorithm [12] commonly used to delineate arbitrarily shaped clusters in complex multi-modal feature spaces. Our results show that distributing the algorithm improves the scalability and run-time over the single node version, as expected, and that multi-level TBŌNs provide improved scalability compared to flat, one-to-many process organizations.

In Section 2, we describe the TBŌN computational model, as well as MRNet (a prototype TBŌN), and discuss the various applications of TBŌNs. In particular, we give many examples of how TBŌNs are currently used and the types of algorithms and applications for which TBŌNs are well-suited. We then detail the application, environment and evaluation of our case study in Section 3. We end with a discussion of implications for future research in Section 4.

## 2. Tree-based Overlay Networks

This section provides a more detailed discussion of TBŌNs. We define the TBŌN model and describe MRNet, a TBŌN-based multicast, reduction network. We then discuss the many existing and future applications of TBŌNs for high-performance computing.

### 2.1 TBŌN Computational Model

In the TBŌN model, as shown in Figure 1, application processes are connected by a tree of internal or *communication processes*. We use the term application to describe the system that is directly leveraging the TBŌN infrastructure, be it a tool (as is most common) or an actual end-application. The application process at the root of the tree is the *front-end* process; those at the leaves are the *back-end* processes. Collectively, the front-end and back-ends are called *end-points*; the tree of processes, connected via FIFO channels, serve as conduits through which *application-level packets* flow to and from end-points.

The strength of the TBŌN model comes from the ability to place and control the execution of application-level logic throughout the TBŌN infrastructure. This feature, provided by the *data filter* abstraction, is used to perform data aggregation and reduction operations on *in-flight* application-level packets. A filter can be any function that inputs a set of packets and outputs a single packet[1]. *Persistent filter state*, used to carry side-effects from one filter execution to the next, increases the power of the filter abstraction. Given the flexible nature of filters, in addition to obvious data aggregation operations, like *min*, *max*, *count*, and *average*, filters can be used to perform quite complex data aggregations and even to enforce data synchronization policies, as discussed in this section.

The TBŌN model does not support direct back-end to back-end communication. However, similar support could be easily achieved, albeit in a sub-optimal manner, by using the internal process-tree to route back-end to back-end messages. Part of the strength of the TBŌN model is the simplicity of its communication structure. A compelling number of important computations fit this model, and we have been able to leverage it to provide highly efficient reliability mechanisms [2].

### 2.2. MRNet: A TBŌN Prototype

MRNet [23] is a TBŌN-based multicast/reduction. Using a MRNet tree network, the application front-end and back-ends communicate using virtual channels called *streams*. Applications specify *filters* to synchronize and transform data flowing on these stream. Toward the goal of generality, MRNet's key features are:

- *Extensible reduction and synchronization:*
  In MRNet, *transformation filters* are the key to supporting scalable distributed computation. MRNet has built-in transformation filters for common aggregations including `avg`, `sum`, `min`, `max` and `concat`. Additionally, MRNet allows developers to extend the filter set with application-specific filters. These new filters may be loaded on-demand into instantiated networks; an interface similar to `dlopen` is used to dynamically specify and load the filters into the running communication processes.

  MRNet uses *synchronization filters* to enforce the simultaneous delivery of packets regardless of the time they actually arrive at a communication process. MRNet has three built-in synchronization filters: `wait_for_all` delivers packets in groups based on packet receipt from all downstream children, `time_out` delivers packets received within a

[1] While the general TBŌN model does not constrain the number of output packets that a filter may produce, in practice we have not found the need for outputting multiple packets.

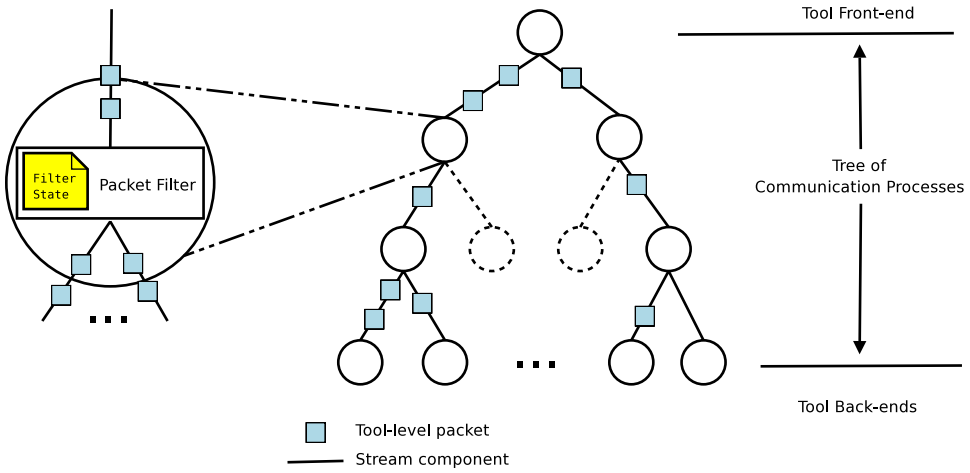


**Figure 1. A tree-based overlay network. Application-level packets flow upstream from back-ends to front-end via communication processes; stateful filters synchronize and transform packets along the way.**

specified window, and `null` delivers packets immediately upon receipt.

- *Flexible topologies:*
  MRNet allows a tool to specify the organization of its communication process-tree. The topology can be a tree organization of any shape or size including balanced ($k-ary$) and skewed ($k-nomial$) trees. Once the overall process-tree organization is specified, a tool can use MRNet streams to specify sub-trees: by using streams to connect a subset of back-ends, the tool can select different portions of the topology for different communication needs.

  MRNet also supports a more dynamic topology model in which the entire tree does not have to be statically specified – back-end processes may join after the internal tree has been instantiated. We are extending MRNet to support even more dynamic topologies where communication and back-end processes can show up or leave at any time (perhaps as a response to failures, recoveries, or load balancing) and the network properly reconfigures and re-routes traffic without any data loss.

- *High-performance communication:*
  High-performance means controlling both space and time usage. MRNet is optimized to support high-throughput data transmissions. It uses *counted packet references* to place a single packet object into multiple outgoing packet buffers and performs the requisite garbage collection when the packet is no longer referenced. MRNet also uses *zero-copy data paths* whenever possible to reduce messaging overhead.

- *Flexible communication model:*
  MRNet supports data communication across multiple, concurrent data streams that may overlap in end-point membership. This concurrency can reduce packet delays and provide a convenient mechanism for tools to communicate in different ways, perhaps using different data aggregations, amongst different or even the same end-points.

MRNet does not support filter chaining where a sequence of filters are applied at each communication process. A single "super filter" that propagates the packet flow to a sequence of filters could seamlessly mimic this functionality. Also, currently streams propagate data in a single direction upstream, toward the front-end, or downstream, toward the back-ends. We plan to extend MRNet so that a filter can propagate information along a stream in either direction.

MRNet was first evaluated by integrating it into Paradyn, a distributed performance profiling tool organized into a central manager that controls, collects, and analyzes performance data from remote daemons [23]. MRNet filters were used to implement an efficient tree-based clock-skew detection algorithm and equivalence class computations to suppress redundant information communicated by the daemons. With 512 daemons, these filters improved the tools startup time from over 1 minute to under 20 seconds (3.4 speedup). For data aggregation of a moderate flow (performance data of 32 functions), the front-end in Paradyn's original one-to-many architecture could not process data at the rate it was being produced by more than 32 daemons. Using MRNet, the front-end easily processed the loads of-

fered by 512 daemons (the highest node count we could use). Similar results for thousand node runs have been achieved for other filters, like a sub-graph folding algorithm (SGFA) for combining sub-graphs of similar qualitative structure into a composite sub-graph [24].

### 2.3. TBŌN Applications

It is well-understood that the tree abstraction is powerful for performance and scalability. To reinforce this idea, we now describe several other tools and tool infrastructures that use TBŌNs and discuss future application areas to which we believe the TBŌN model is well-suited.

**Middleware Infrastructures.** Like MRNet, Ygdrasil [3] and Lilith [11] are tree-based communication infrastructures for scalable tools. Ygdrasil, derived from the communication infrastructure used in the Ladebug parallel debugger [4], uses a tree of *aggregator nodes* to apply user-specified, Java-based plugins to in-flight data. Ygdrasil uses a synchronous request/response communication model, where data flows upward in response to downward control or request messages. Lilith provides a platform for distributing user code, generally system administrative tasks, and launching these tasks across heterogeneous systems. Lilith organizes these task processes into a tree; task output is propagated to the root of the tree and can be modified en-route by a single user-specified filter. Like MRNet, Ygdrasil and Lilith provide mechanisms for extensible data reductions, though neither Ygdrasil nor Lilith support extensible data synchronization operations. Lilith and Ygdrasil support balanced tree topologies and are implemented in Java to leverage Java's natural ability to load dynamically.

TAG [21] is a tree-based, aggregation infrastructure for sensor networks; TAG provides a database-like SQL interface that allows users to express simple, declarative queries that execute in a distributed manner on the nodes of the sensor network. Similar to MRNet, TAG supports multiple simultaneous aggregation operations and supports streams of aggregated data in response to an aggregation request. TAG uses ad-hoc routing to automatically organize its sensors into a tree.

**Distributed System Tools.** While the previous infrastructures were designed to be general-purpose TBŌNs, several tools for cluster and Grid systems use application-specific TBŌNs, for example, the distributed system monitors, Ganglia [25] and Supermon [26]. Ganglia uses a multi-level hierarchy in which the level furthest from the root, is used to represent a cluster of nodes and the higher levels represent federations of clusters. Within a Ganglia cluster, monitoring data is replicated across all nodes; upon request, a single replica propagates monitoring data for the entire cluster. In Supermon, monitoring servers can also act as clients allowing the system to be configured into hierarchies of servers. These servers can execute *data concentrators*, implemented using functional *symbolic expressions* from Lisp, on monitored data.

**Distributed Applications.** In many application domains, from areas like image processing to bioinformatics, scientists encounter the challenge of performing analyses on large (tera and peta scale) datasets. While the TBŌN model has proven its utility in distributed software and system tool environments, we believe there are large classes of software applications and algorithms from the scientific domains that can use this model for efficient large scale operations at such scale.

At large scales, the TBŌN computational model befits any algorithm with the following properties:

1. the algorithm's computational complexity is at least $O(n)$, where $n$ is the size of the input,
2. the algorithm's output is lesser in size than its total inputs, and
3. the algorithm's output is in the same form as the inputs, for example, if the inputs are sets of elements, the output should be a set of elements.

These properties describe general data reduction algorithms. The remainder of this section will show that many common algorithms used in scientific applications fit this category.

Data mining or information extraction, the process of distilling specific facts from large quantities of data, is an important element of many real world applications. Application areas include Internet information retrieval [17, 29], bio-information [16], intrusion detection and threat analysis [5], geographical information systems [13], business intelligence [18, 19], and organizing digital audio collections [27]. Typically, the desired information describes relationships amongst the elements in the datasets or frequencies and other statistics of classes of elements. Data clustering is the primary technique used for this information extraction.

Clustering algorithms are generally based on one of two strategies, *partitioning* or *agglomeration*. Partitioning algorithms classify a given data set into $k$ disjoint clusters, with $k$ fixed a priori. *K-means* [14, 20], the most common partitioning algorithm, defines and iteratively refines $k$ centroids, one for each cluster, associating each data point with its nearest centroid based

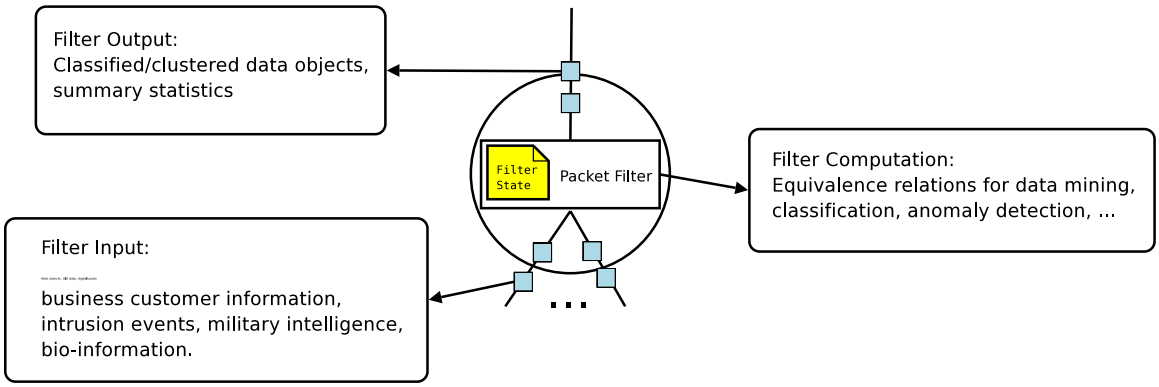


**Figure 2. A diagram showing the mapping of data clustering algorithms to a TBŌN equivalence class filter computation.**

on distance (similarity) measures. In agglomerative clustering [15], a data set with $N$ elements is initially partitioned into $N$ clusters each containing a single element. Larger clusters are formed by iteratively merging nearest-neighbor clusters. In clustering algorithms, (pre-built or learning) data models or statistical analysis are used to group related elements and distinguish unrelated ones. These general algorithms match our definition of a data reduction: the computational overhead increases with the number of elements to classify, and the result of the algorithm is a smaller summary representation of the inputs. The mapping of these algorithms to a TBŌN filter computation is shown in Figure 2. This figure shows that algorithms of this nature reduce to a *equivalence class* filter computation, where the inputs are elements to classify (or summarize), the computation is the application of data model or statistics to classify the data into the classes they represent, and the output is the classified data (or summary of the classified data).

The application of TBŌNs for scalable scientific data analyses appears compelling, and we expect the TBŌN model to be as effective in application domains as it has proven to be in tool domains. As further evidence, the next section presents a case study that demonstrates the use and benefit of TBŌN infrastructures for a commonly-used, general-purpose data clustering algorithm.

## 3. A New (Non-tool) Case Study

We use the MRNet prototype to develop a scalable, TBŌN-based version of *mean-shift* [12], a powerful clustering technique that can delineate arbitrarily shaped clusters in complex, multi-modal feature spaces. We then compare the performance of this distributed computation to the single-node version.

Mean-shift is an iterative procedure that shifts the center of a search window in the direction of greatest increase in the density of the data set being explored. It continues to shift the window until the window is centered on a region of maximum density. The algorithm is non-parametric in that it does not require a priori knowledge of the number of clusters in the data, as in the case of k-means clustering.

Mean-shift is becoming increasingly popular in computer vision applications, for example, color image segmentation [8], face tracking [6], image transformation [9], and more general cluster analysis and global optimization problems [7]. The drawback of mean-shift is that the computation becomes prohibitively expensive as the size and complexity (dimensionality) of the data space increases [7].

In this case study, the input data may be considered as image data. We investigate the use of mean-shift to find peaks, which can then be used to segment the input image into layers, for example, foreground and background, or to extract other information. Similar computations could be used for the other application areas mentioned above.

### 3.1. Implementation

We have implemented the mean-shift algorithm for two dimensional data; the core of the algorithm is given in Figure 3. The mean-shift kernel operates on a window of data centered on a given data point, estimated to be a centroid (center of density). As shown in the algorithm, the points in this window are used to move the current centroid toward areas of higher density, using a mean-shift density estimator. On each iteration, the

mean-shift density estimator calculates a vector that will move the current centroid toward higher density areas. For a given window of data, the algorithms stops when successive iterations do not yield a new centroid (or a maximum iteration threshold has been met).

```
1. do
2.    for all points in window around current centroid
3.       Calculate euclidean distance from current centroid
4.       Use distances to calculate mean-shift vector
         toward higher density estimate
5.    end for
6. while mean-shift vector is non-zero
```

**Figure 3. The Mean-shift Algorithm.**

The kernel estimator calculates the mean-shift vector over this region using a shape function to weight the data. We choose a Gaussian shape function, which gives greater weight to points nearer to the center; this effectively smooths the data giving improved performance when dealing with noisy data. Other options for shape functions would weight each data point uniformly, quadratically, or triangularly.

In our implementation, there are two parameters that must be specified or estimated. The first is a threshold that sets the minimum data density at which a mean shift search will begin. Low density areas are poor candidates for modes and the mean shift search will likely make little progress in these areas. The second parameter estimates the bandwidth of the data or, roughly speaking, the variability of the data. In many cases this is determined experimentally. We choose a fixed bandwidth of 50 which seems to work well with our data. For a discussion of a general approach to determining bandwidth adaptively, see Comaniciu, Ramesh and Meer's paper [10].

The single node (non-distributed) version of the algorithm works as follows. We scan across the data and calculate the density of the data using a fixed window. The regions where the density is above our chosen threshold are used as the starting points for the mean shift search. The mean shift algorithm runs until it converges on a local maximum that we keep as a peak.

For our experiments, we used a distributed algorithm where each leaf node gets a part of the data set. Each node applies the mean shift procedure then sends the resulting data set and the list of peaks to the next higher node in the network. Each parent node merges the data sets of it's children and then applies the mean shift procedure to the new data set using the peaks determined by child nodes as the starting points.

The data at the leaf nodes is synthetically generated. The data about each cluster center is generated using a random Gaussian distribution. The cluster centers are slightly shifted in each leaf node as they might be in feature tracking in video processing or when processing images with non-uniform illumination.

### 3.2. Evaluation

To evaluate the performance impact of the TBŌN computational model, we performed experiments to compare the running times of the implementation of mean-shift, discussed above. Our methodology is to increase the scale of the input data size and evaluate how well (or poorly) the algorithm performs when executed using a single node, 1-deep (shallow), and 2-deep (deep) trees, with varying numbers of leaves.

As data sizes increase, naturally we expect the workload to become prohibitively expensive for a single node. The shallow trees represent the simple scaling solution that directly distributes or farms out the computation. Unfortunately, this solution does not consider the cost of data aggregation at the central point and becomes prohibitively expensive as the fan-out of the front-end node increases. Using TBŌNs for deep trees allows the workload to be distributed in a fashion that can limit the fan-out of the nodes and thus improve data consolidation. Clearly, deep trees come with the cost of increased node usage; however, this penalty is moderate. For example, with a fan-out of 16, 16 (6.25% more) internal nodes are needed to connect 256 back-ends, or 272 (6.6%) for 4096 back-ends.

The experiments were performed on a cluster of 2.8 – 3.2 GHz Pentium 4 workstations, each with two gigabytes of RAM and inter-connected by a Gigabit Ethernet network. Each experiment used a fully-balanced tree, and we grouped experiments by the scale of the input data size. For every experiment, each back-end generates input data of the same size and distribution; that is, the input size scales with the number of back-ends, ranging from 16 to 324. Essentially, as we scale up the problem, we analyze increasingly larger images. If the images were being generated by an array of cameras, for example, a larger scale might mean a larger array [28]. The measured processing time starts with the broadcast of a control message message from the front-end that instructs the back-ends to initiate the mean-shift algorithm and ends when the results of the mean-shift have been calculated and are available at the front-end process.

Each experiment was run two to four times, and the average measured processing times are plotted in Figure 4. This figure shows that the distributed versions of the algorithm improves the scalability and perfor-

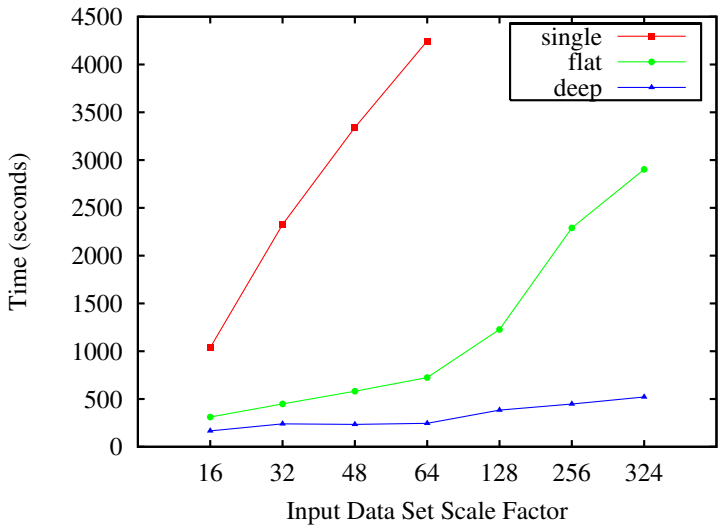


**Figure 4. Mean-shift Processing Times: The algorithm was implemented as a TBŌN filter. For the single-node case, the X axis is the scaling factor of the input data set. For the 1-deep and 2-deep trees, the X axis is both the scale factor for the input data set and the number of back-end processes.**

mance of the single-node, and that deep trees provide significant performance and scalability improvements over shallow trees at larger scales. As expected, the runtime of the single-node version of mean-shift algorithm increases linearly with the input data size. As the workload scales, the run-times of the shallow tree versions increase at a much lower rate, until the fan-out of the shallow tree becomes prohibitively large and data consolidation at the front-end becomes a bottleneck – somewhere between a fan-out of 64 and 128. However, the performance of the deep trees remain relatively constant for all scales of input data size. Beyond 64 leaves, we do observe a small linear increase in execution times. We have determined that beyond 64 leaves, the run-time is directly proportional to the fan-out of the tree. An open question is whether even deeper trees with limited fan-outs would yield a constant execution time as the scale increases.

## 4. Summary

In this paper, we posit that tree-based overlay networks (TBŌNs) provide a powerful and flexible platform for scalable control, data collection, and data analysis for applications as well as tools. We presented many examples of TBŌNs being used in tools for simple aggregation operations as well as complex operations like clock-skew detection, time-aligned data synchronization, sub-graph folding for scalable data presentation, and creating data histograms. We then showed how the TBŌN computational model can be applied to many types of applications and algorithms used in scientific data analyses and used the MRNet prototype to show the scalability of a TBŌN-based distributed data clustering algorithm.

Our experiments with MRNet are only the start of our work on more general uses of TBŌNs. We believe that many other computing tasks can be supported by these networks for applications in areas like data mining, bio-information, image processing, and geographic information systems. As future work we are looking at using TBŌNs as a general tool that can support other clustering algorithms, or data models such as decision and regression trees that can be built by passing data both directions in the tree. This bidirectional communication allows model cross-validation or refinement via operations performed directly on the models.

## 5. Acknowledgments

We are grateful to Sean Murphy and Miron Livny of the Condor Project at the University of Wisconsin who helped tremendously in procuring and setting up our experimental environment, sponsored by the National Science Foundation under Grant EIA-0320708.

## References

[1] Top 500 supercomputer sites. http://www.top500.org (last visited January 2006).

[2] D. C. Arnold and B. P. Miller. Zero-cost reliability for tree-based overlay networks. Technical report, University of Wisconsin – Madison, December 2005.

[3] S. M. Balle, J. Bishop, D. LaFrance-Linden, and H. Rifkin. Ygdrasil: Aggregator network toolkit for the grid, June 2004. Presented in a minisymposium at Workshop on State-of-the-art in Scientific Computing.

[4] S. M. Balle, B. R. Brett, C.-P. Chen, and D. LaFrance-Linden. A new approach to parallel debugger architecture. In *Applied Parallel Computing. Advanced Scientific Computing: 6th International Conference, PARA 2002*, Espoo, Finland, June 2002. Published as Lecture Notes in Computer Science 2367, J. Fagerholm et al (Eds), Springer, Heidelberg, Germany, August 2002, pp. 139-149.

[5] D. Barbara, editor. *Special Section on Data Mining for Intrusion Detection and Threat Analysis*, volume 30(4) of *ACM SIGMOD Record*, pages 4–64. ACM Press, New York, NY, USA, December 2001.

[6] G. Bradski. Real time face and object tracking as a component of a perceptual user interface. In *IEEE Workshop on Applications of Computer Vision*, pages 214–219, Princeton, NJ, USA, 1998.

[7] Y. Cheng. Mean shift, mode seeking, and clustering. *IEEE Transactions on Pattern Analaysis and Machine Intelligence*, 17(8):790–799, 1995.

[8] D. Comaniciu and P. Meer. Robust analysis of feature spaces: Color image segmentation. In *IEEE Computer Vision and Pattern Recognition*, pages 750–755, Puerto Rico, 1997.

[9] D. Comaniciu and P. Meer. Mean shift analysis and applications. In *IEEE Computer Vision (ICCV '99)*, pages 1197–1203, Kerkyra, Greece, 1999.

[10] D. Comaniciu, V. Ramesh, and P. Meer. The variable bandwidth mean shift and Data-Driven scale selection. In *8th International Conference on Computer Vision*, volume 1, pages 438–445, Vancouver, BC, Canada, July 2001.

[11] D. A. Evensky, A. C. Gentile, L. J. Camp, and R. C. Armstrong. Lilith: Scalable execution of user code for distributed computing. In *6th IEEE International Symposium on High Performance Distributed Computing (HPDC 97)*, pages 306–314, Portland, OR, USA, August 1997.

[12] K. Fukunaga and L. Hostetler. The estimation of the gradient of a density function, with applications in pattern recognition. *IEEE Transactions on Information Theory*, 21:32–40, 1975.

[13] R. H. Guting. An introduction to spatial database systems. *The VLDB Journal*, 3(4):357–399, 1994.

[14] J. A. Hartigan and M. A. Wong. A k-means clustering algorithm. *Applied Statistics*, 28:100–108, 1979.

[15] S. C. Johnson. Hierarchical clustering schemes. *Psychometrika*, 2:241–254, 1967.

[16] C. Kirbas and F. Quek. A review of vessel extraction techniques and algorithms. *ACM Computing Surveys*, 36(2):81–121, 2004.

[17] M. Kobayashi and K. Takeda. Information retrieval on the web. *ACM Computing Surveys*, 32(2):144–173, 2000.

[18] R. Kohavi and F. Provost. Applications of data mining to electronic commerce. *Data Mining Knowledge Discovery*, 5(1-2):5–10, 2001.

[19] R. Kohavi, N. J. Rothleder, and E. Simoudis. Emerging trends in business analytics. *Communications of the ACM*, 45(8):45–48, 2002.

[20] J. B. MacQueen. Some methods for classification and analysis of multivariate observations. In *Fifth Berkeley Symposium on Mathematical Statistics and Probability*, volume 1, pages 281–297, Berkeley, CA, USA, 1967. University of California Press.

[21] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. Tag: a tiny aggregation service for ad-hoc sensor networks. In *5th Symposium on Operating Systems Design and Implementation (OSDI)*, Boston, MA, USA, December 2002.

[22] J.-P. Navarro, R. Evard, D. Nurmi, and N. Desai. Scalable cluster administration: Chiba city i approach and lessons learned. In *IEEE International Conference on Cluster Computing (CLUSTER 2002)*, pages 215–221, Chicago, IL, USA, September 2002.

[23] P. C. Roth, D. C. Arnold, and B. P. Miller. Mrnet: A software-based multicast/reduction network for scalable tools. In *SC 2003*, Phoenix, AZ, USA, November 2003.

[24] P. C. Roth and B. P. Miller. The distributed performance consultant and the sub-graph folding algorithm: On-line automated performance diagnosis on thousands of processes. Submitted for publication.

[25] F. D. Sacerdoti, M. J. Katz, M. L. Massie, and D. E. Culler. Wide area cluster monitoring with ganglia. In *IEEE International Conference on Cluster Computing (CLUSTER 2003)*, pages 289–298, Hong Kong, September 2003.

[26] M. J. Sottile and R. G. Minnich. Supermon: A high-speed cluster monitoring system. In *IEEE International Conference on Cluster Computing (CLUSTER 2002)*, pages 39–46, Chicago, IL, USA, September 2002.

[27] W.-H. Tsai and H.-M. Wang. On the extraction of vocal-related information to facilitate the management of popular music collections. In *5th ACM/IEEE-CS Joint Conference on Digital Libraries (JCDL '05)*, pages 197–206, New York, NY, USA, 2005. ACM Press.

[28] B. Wilburn, N. Joshi, V. Vaish, E.-V. Talvala, E. Antunez, A. Barth, A. Adams, M. Horowitz, and M. Levoy. High performance imaging using large camera arrays. *ACM Transactions on Graphics*, 24(3):765–776, 2005.

[29] L. Xiao, D. Wissmann, M. Brown, and S. Jablonski. Information extraction from the web: System and techniques. *Applied Intelligence*, 21(2):195–224, 2004.