Automated Information Aggregation for Scaling Scale-Resistant Services

Philip Gross and Gail Kaiser
Columbia University
Department of Computer Science
New York, NY 10027
{phil|kaiser}@cs.columbia.edu

Abstract

Machine learning provides techniques to monitor system behavior and predict failures from sensor data. However, such algorithms are "scale resistant" — high computational complexity and not parallelizable. The problem then becomes identifying and delivering the relevant subset of the vast amount of sensor data to each monitoring node, despite the lack of explicit "relevance" labels. The simplest solution is to deliver only the "closest" data items under some distance metric.

We demonstrate a better approach using a more sophisticated architecture: a scalable data aggregation and dissemination overlay network uses an influence metric *reflecting the relative influence of one node's data on another, to efficiently deliver a mix of raw and aggregated data to the monitoring components, enabling the application of machine learning tools on real-world problems. We term our architecture* Level of Detail *after an analogous computer graphics technique.*

1. Introduction

Organizations are deploying larger and larger numbers of networked computational units, with node counts planned to increase into the millions [1]. Examples include large utilities (energy, telecoms) installing intelligent controls, sensors, and home meters, military organizations giving ruggedized Personal Digital Assistants to every soldier, as well as the basic inventory of desktop PCs of large organizations such as the U.S. federal government.

Monitoring and control of such massive distributed systems is a challenge. Ideally, we would like to be able to proactively identify and prevent problems before they occur. When we fail to do so, we need to characterize the problem as rapidly and frequently as possible so we can take action to maintain availability, reliability, response time, or other desiderata. However, for these large systems and their accompanying flood of noisy sensor data, even confirming the existence of a problem may be a challenge.

Further, there is the challenge of non-local component interactions. A switch that would normally open to protect a minor piece of equipment, or a software component that needs to reset the system to complete a reconfiguration, may need to inhibit their default actions to avoid causing problems for other, distant components. Such a decision to inhibit needs to be taken quickly, and possibly in the presence of a partial system failure, which may have changed the dynamics of the system.

Handling such higher-order behaviors requires a system model, so that consequences for the wider system can be evaluated. The inputs to such a model will be streams of data coming from a variety of sources, from simple sensors to intelligent components. Given the scale of the systems in question, and the need to rapidly generate models (to adjust to large-scale system problems, and also to simply keep up with organic changes in the system), such models will need to be automatically generated.

1.1 Machine and Reinforcement Learning

The field of Machine Learning (ML) has, in recent years, developed algorithms such as Support Vector Machines [4] and Adaboost [20] which are useful for characterization of complex system behavior, e.g., classifying a component as having a high risk of failure or not, given its current environment, or ranking a set of components according to their susceptibility to failure. These models can be generated from the same data streams that are used to evaluate current system state, are designed to generalize well, even in the presence of noisy or missing data, and have a sound theoretical basis [23]. Further, Reinforcement Learning (RL) algorithms can use these models to evaluate hypothetical situations and establish a set of policies for future

21st IEEE International Conference on Automated Software Engineering (ASE'06)
0-7695-2579-2/06 $20.00 © 2006 IEEE

action [12]. Further discussion of RL is outside the scope of this paper, which will focus on the dissemination of data to model-generating ML algorithms.

However, these algorithms are themselves compute-intensive operations, and are difficult to scale to very large data sets [11, 24]. They are intrinsically iterative, preventing a parallel distributed solution across our many nodes as with Adaboost, or have a fundamentally complex algorithm in their core, as with the quadratic programming problem within SVMs. This is problematic since we would like to compute these models and policies as rapidly as possible, to keep up with the current state of the system.

These scale-resistant algorithms are generally $O(n^2)$ or worse, where n is the number of training examples for ML. Additionally, direct implementations of some ML algorithms, e.g., Support Vector Machines, will have $O(n^2)$ space complexity as well, although more sophisticated implementations avoid this at the cost of slower operation [23, 6, 4, 11]. In the case of constructing models for monitoring or management of very large distributed systems, we may have millions of nodes and a frequently-changing dynamic system, leading to values of n that are intractable for these algorithms, regardless of computer power or memory capacity. Note that simply evaluating an existing model, by giving it a set of observed or hypothetical values for the independent variables and obtaining a result (e.g., to make predictions from new sensor data), may itself be a non-trivial operation, but is generally of a lower order of complexity than the ML algorithms which actually construct the models.

Since the computational complexities of these algorithms are too high, one approach to making the problem of automatic model generation tractable is to reduce the size of the data set. For instance, while the problem of computing a single global model is difficult to parallelize, we could instead generate many smaller local models in parallel. We would impose some distance metric on the nodes, and then ignore any data coming from a source more distant than some short horizon h_∞, ensuring that no node has more data sources closer than h_∞ than it can process within the desired model generation time. For example, in the case of monitoring electrical distribution infrastructure, a circuit breaker might feed its models with data from directly connected cables (at distance 1), from transformers and joints connected to those (at distance 2), and ignore all data from anywhere else.

However, this approach makes components blind to any signals coming from outside their immediate neighborhood. If our components are to deal intelligently with non-local effects, they will require information on the state of the system as a whole.

1.2. Level of Detail

In order to enable the distributed generation of broadly-based, locally focused system models, we introduce an aggregation-dependent approach which we call *Level of Detail*, as it is somewhat analogous to the computer graphics technique of the same name [5]. This approach is built on three main ideas: An *influence metric* describes the importance of a data source to a particular node, *aggregation functions* reduce one or more streams to a single stream with a bounded data rate, while preserving information, and a *dissemination mechanism* moves all data where it needs to go, despite limited bandwidth in the underlying network. This paper focuses on the influence metric and aggregation functions, and briefly describes the data dissemination mechanism.

An aggregation function may be something as simple as returning the min or max value seen each time quantum, or as complex as fitting a regression curve to the multiple streams. Aggregations are characterized by their *aggressiveness*, which corresponds with their degree of data reduction. An aggregation that reduced a thousand high-speed data streams to one low-speed stream would be very aggressive, while one that down-sampled a single stream to 80% of its previous rate would not be considered very aggressive.

In the Level of Detail approach, the data streams delivered to a particular node are aggregated inversely proportional to their influence. Thus for a given node N, data streams from nodes with low influence on N will be strongly aggregated (i.e., subject to a large amount of data reduction), while data from nodes with high influence will be lightly aggregated if at all.

We hypothesize that this approach of presenting local model-generating algorithms with a mixture of data streams derived from much or all of the full system will give useful results, close enough to those obtained from actually analyzing the entire system, but within acceptable time bounds. Our initial investigations seem to support these hypotheses.

The idea of Level of Detail is illustrated below. Instead of simply taking all data with influence greater than some threshold and nothing else, as in Figure 1, we take the most influential data, along with information that comes from a much wider span of the system, aggregated more and more aggressively as it is estimated to be less and less influential (Figure 2).

This entire framework, consisting of influence metrics, aggregation functions, and distribution mechanism feeding limited data streams to scale-resistant modeling algorithms, enables the introduction of smart components that can take broader system issues into account when making local decisions.

We present results showing that when the Level of Detail technique is applied to actual data sets from the infras-

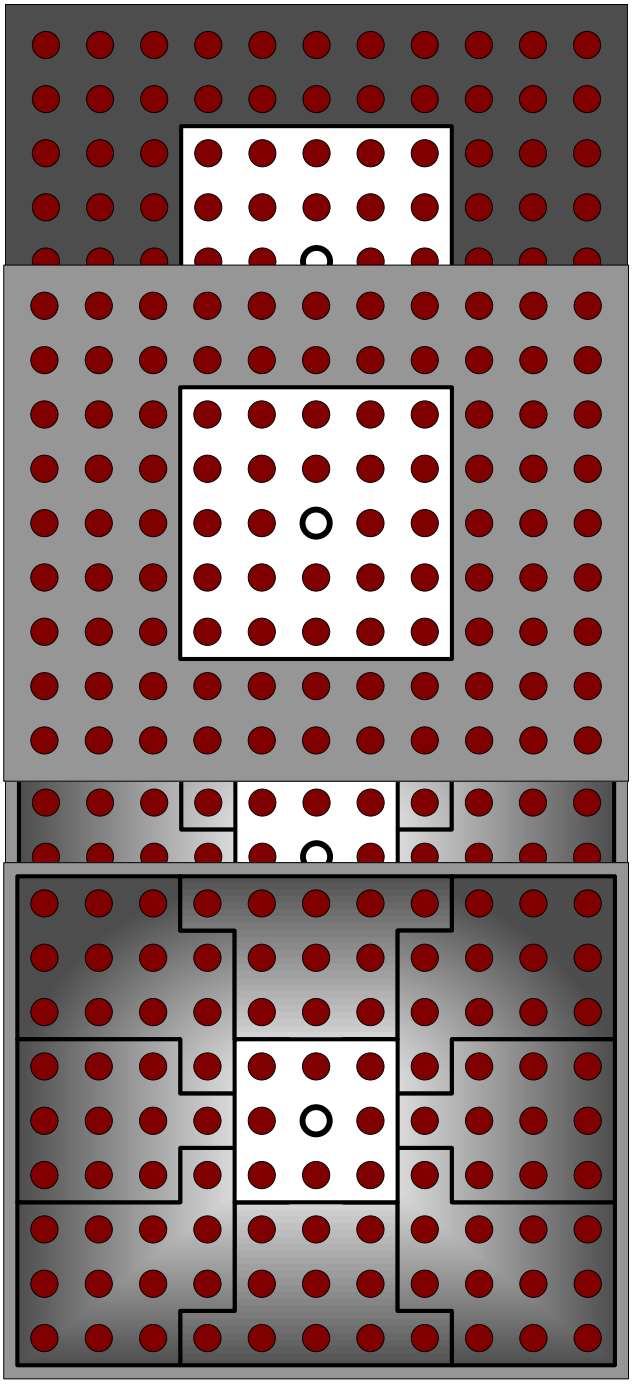

Figure 2. The Level of Detail approach is to accept all data from a smaller neighborhood, and use our remaining model capacity on aggregated data from a larger set of nodes.

tructure of a large electricity-distribution utility, not only can intensive machine learning calculations be made more tractable via a significant reduction in input size, but in some cases, results actually showed improvement compared with learning on all raw data, despite a general computational learning theory guideline that more training examples give better results [23].

This paper is organized as follows: Section 2 describes the design of our solution. Current results in the context of ongoing joint research between Columbia and Consolidated Edison Company of New York (ConEdison) are described in section 3. Section 4 discusses past and current related work as it relates to this problem. Finally, conclusions and future work are in section 5.

2. Approach

To achieve a Level-of-Detail architecture we must overcome two problems:

1. **Algorithmic Complexity:** The services of interest are intractable when applied to a full system model. We will need to make them tractable on a given node, using a reduced model of the system, while still producing useful results.

2. **Dynamic System:** The systems of interest are not static—nodes are continually being added, failing, and having their interconnections rearranged. Our solution must be able to adapt to changes in the system structure, at a speed comparable to the rate at which significant changes occur.

The Level of Detail model is based on *aggregation operations*, varying *aggressiveness* of these aggregations, *influence metrics*, and a *data dissemination mechanism*, which are described further below.

2.1 Aggregation

We define an *aggregation* as a function that takes one or more data streams as input and outputs a single stream with a bounded rate. The worst-case output rate will be less than or equal to the worst-case sum of the input rates, although for typical aggregations, as we shall see, the output rate will be far lower.

Some examples of aggregations under this definition would be sampling, SQL-style aggregations like MAX or AVG, or using a bounded buffer and dropping incoming data after the quota for a time quantum has been reached. Different learning algorithms may work better with different types of aggregation functions.

For scalability, aggregation functions are required to be incrementally calculable. This enables distributing the computation of the aggregation, and avoids overwhelming a single node with data to aggregate. For example, the maximum over many readings can be found by repeatedly taking the maximum over small subsets. Similarly, arithmetic means can be found by accumulating the sum and count of each subset, and then dividing at the very last step. The statistical median of a set of data, on the other hand, cannot be calculated from the results of local operations on subsets, so it would not be suitable as an aggregation function.

Note that under this definition, aggregations are not necessarily lossy, as this definition considers only the raw data rate, not the information content. As a trivial example, duplicate input streams could be replaced with a single output stream with no loss of information.

When generating models for use in monitoring and management, we need to balance the desire for information from the wider system with an extremely limited data budget. Aggregations provide the mechanism for simplifying and shrinking the huge amounts of system-wide data to a size the ML algorithm can handle.

2.2 Aggressiveness

We characterize the data reduction achieved by a particular aggregation as its *aggressiveness*. We indicate aggressiveness with a parameter $\alpha \in [0, 1]$ representing the reduction in data rate. Specifically, the value of α for an aggregation function f operating on input streams $s_1, \ldots, s_n$ with data rates $r_1, \ldots, r_n$ and producing an output stream s_{out} with rate r_{out} is

$$\alpha(f, \{s_i\}, s_{out}) = \frac{r_{out}}{\sum_{i=1}^{n} r_i}$$

Thus a value of $\alpha = 0$ means that all input data is simply discarded, while $\alpha = 1$ means that no aggregation is occurring.

Stochastic, non-deterministic aggregators may take α as a parameter, for instance letting a particular data item pass with probability α, and otherwise dropping it.

For deterministic aggregators, α is inversely proportional to the amount of input. If our aggregator is taking average values across its set of input streams each time quantum, giving it five input streams is less aggressive than giving it twenty input streams, assuming equal output rates.

2.3 Influence

When a particular node is planning to run a scale-resistant service, e.g., machine learning for failure prediction, the service will require system data from which to construct a model. The node running the service will have a particular input budget based on how much data the service can consume while still producing results within a reasonable time (with respect to the rate of change of the system). We decide how to spend our input data budget based on a metric we call *influence*. Influence is a function on node pairs, $Infl(N_s, N_t) \rightarrow [0, 1]$, that should be efficient to compute (or look up), and have a positive correlation with the amount of impact that data from N_s will have on the outcome of a computation by the scale-resistant service at our target node N_t. The output value of this function can be used directly as a value for α when determining aggregation aggressiveness. If $Infl(N_s, N_t) = 0.5$, then a stochastic aggregator could drop a randomly selected half of N_s's data on the way to N_t.

Influence is how a particular learning algorithm running on a particular node gives hints to the Level of Detail framework about what data will likely be of greatest utility. Some algorithms may work with outliers, others with averages, some will give optimal results with raw, individual data streams, while others need as wide and balanced a picture of the whole system as possible. Influence gives the framework guidance on how to allocate that component's data budget over the available data streams.

2.4 Influence Classes

While we could calculate all individual pairwise influence values, we can gain efficiencies by organizing the world as seen from the perspective of a particular node into $k + 2$ *influence classes*, where I_0 is the class of nodes with the most influence on us, I_∞ is the class of nodes with no influence, and $I_1, I_2, \ldots, I_k$ are classes of nodes with intermediate influence. Thus the number of distance pairs among n nodes drops from $O(n^2)$ to $O(k^2)$, with $k \ll n$. The notation of I_0 for the most influential class and I_∞ for the least is intended to suggest Cartesian "distance" from the target node, but the concept is more general, e.g., one could use a pairwise measurement of component similarity for influence, based on age, manufacturer, size, etc.

When describing computations from the perspective of a scale-resistant service at a node, we use the notation $Infl(N_s, self)$ to represent the influence of node N_s's data on that service's computation. Nodes N_0 in class I_0 are defined to have $Infl(N_0, self) = 1$, indicating that no aggregation should be performed on the data from these nodes. At the other extreme, nodes N_∞ in class I_∞ have $Infl(N_\infty, self) = 0$, indicating that data from these nodes should simply be discarded.

Each influence class may use different aggregation functions, and a given influence class may be *flat*, with all nodes in the class having the same influence on the target node, or it may be a *gradient*, with influence varying among nodes in the class. For instance, there may be other components designed to pick up the load if a node fails, which might all

be in its class I_1. Further, there may be a weight associated with each of these support components, indicating what percentage of the load it is expected to cover. We could then make I_1 a gradient class, with the pickup parameter assigning relative influence among members of the class.

In all cases, the invariant holds for all nodes $N_x \in I_x, N_y \in I_y$,

$$x < y \Rightarrow Infl(N_x, self) > Infl(N_y, self)$$

In English, all nodes in class I_2, say, will have greater influence than any node in class I_3.

2.5 Data Dissemination Mechanism

The data dissemination mechanism needs to deliver the appropriately-aggregated data to nodes running scale-resistant services, once relationships under the influence metric have been determined. The complexity of calculating all of the data dissemination paths, the continually changing system, and the need for individual items of aggregated data to be sent to many clients all indicate that a publish-subscribe architecture would be most appropriate. In order to support the potentially large numbers of participants listening or sending to a single source, and potentially high rates of data generation, we will need a data dissemination architecture more sophisticated than a simple *reflector-based* architecture, where a single node is responsible for relaying all messages on a particular channel. A number of very scalable, adaptable, overlay event systems have been developed in the last several years, such as Scribe [19], Bayeux [26], NICE [2], and OMNI [3] which can be used for this purpose.

However, the data dissemination mechanism also needs to funnel data from many sources into relatively few aggregation nodes, aggregating it incrementally along the way. In fact, such a data aggregation network looks like a typical publish-subscribe network, but inverted, with many sources leading to a single destination. The Level of Detail approach requires that the data dissemination system scale as well for aggregation as it does for dissemination. We refer to the combination of data funneling into an aggregation node, and then out to many destination nodes as the "hourglass architecture," illustrated in Figure 3.

Although the problems of data aggregation and dissemination are basically symmetric, we are not aware of any modern overlay publish-subscribe system designed to handle aggregation with the same efficiency as dissemination.

An additional requirement is that subscribers and aggregators need to find publishers in order to receive data. Again, scalable, robust, distributed indicies are a well-studied problem, and we are investigating existing Peer-to-Peer (P2P) systems such as Chord [22], the Content Addressable Network (CAN) [16] or the recent work in using Distributed Hash Tables as SIP resource locators [21] for this purpose.

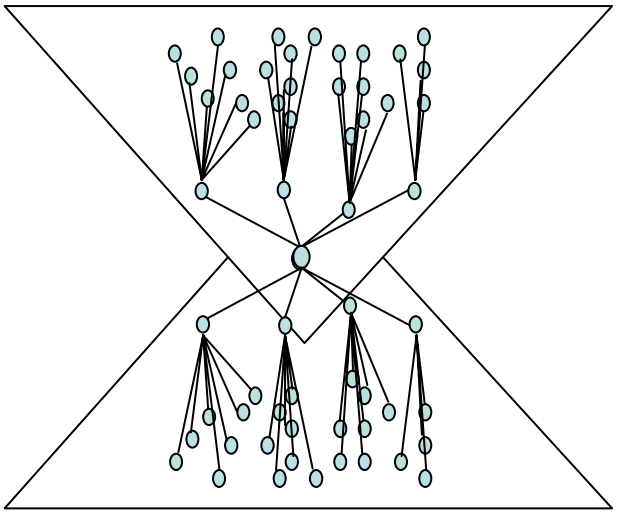

Figure 3. Hourglass Architecture

Our current work involves off-line simulation of the full system, so for our current experiments we are able to statically compute the needed data connections. For the future live deployment experiments, we are developing a system that enables automatic configuration of a scalable aggregation network, as well as handling the more typical dissemination case.

3. Experimental Results

3.1 Context

Much of this work occurred in the context of joint research between Columbia University and Consolidated Edison Company of New York (ConEdison) to develop facilities for managing the electrical distribution system in the future. Electrical infrastructure has four main parts:

1. *Generation*: a prime mover, typically the force of water, steam, or hot gasses on a turbine, spins an electromagnet, generating large amounts of electrical current at a generating station.

2. *Transmission*: the current is sent at very high voltage (hundreds of thousands of volts) from the generating station to substations closer to the customers.

3. *Primary Distribution*: electricity is sent at mid-level voltage (tens of thousands of volts) from substations to local *transformers*, over cables called *feeders*, usually 10-20 km long, and with a few tens of transformers per

feeder. Feeders are composed of many *feeder sections* connected by *joints* and *splices*.

4. *Secondary Distribution*: sends electricity at normal household voltages from local transformers to individual customers

The distribution grid of New York City is organized into *networks*, each composed of a substation, its attached primary feeders, and a secondary grid. The networks are largely electrically isolated from each other, to limit the cascading of problems.

These components are illustrated below.

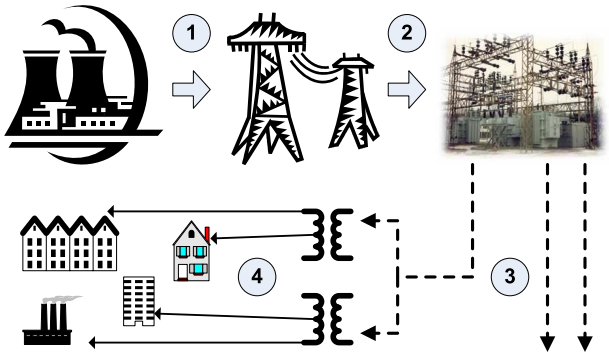

Figure 4. The Electrical Distribution Grid

Our work with ConEdison has focused on the feeders of the primary grid of New York City, as they are system critical and have a significant failure rate. Thus much of the daily work of the ConEdison field workforce involves the monitoring and maintenance of primary feeders, as well as their speedy repair on failure. Note that the system is designed as a 2-connected or 3-connected graph, and so a single feeder failure almost never causes an actual loss of power to a customer. However, the system is at greater risk until it is fixed.

3.2 Susceptibility ranking

A continually-updated estimate of the health of each feeder is therefore an important tool for system management and maintenance, so one of our first goals was to see if machine learning techniques could accurately estimate each feeder's relative likelihood of failure, or "susceptibility index." This is a ranking problem, as opposed to the more typical classification problem, but many of the same algorithms can be used. We trained the machine learning algorithms on data sets that use each feeder as one example. The feeders are described with several hundred attributes, some of which are derived from static data, such as cable length and installation date, while others are derived from dynamic data streams, such as power and voltage measurements. Finally, the response variable is the number times that feeder has had unscheduled, spontaneous outages. The predicted number of future outages is used to rank the feeders.

The machine learning algorithms attempt to create generalizable models based on this training data, such that if given new, previously unseen data on a set of feeders, they will be able to rank them in order of their susceptibility to failure. For our initial experiments, we chose the well-known Support Vector Machines (SVMs) algorithm [4], and a new algorithm called Martingale Boosting (Martiboost) [13] specifically developed (by others involved in the project, not the authors of this paper) for the ConEd susceptibility ranking problem.

3.3 AUC Scoring

In order to evaluate different approaches to the ranking problem, we need some mechanism to compare the generated rankings, to determine if one is "better" than another. We evaluate based on a *test set* of data, disjoint from our *training set*. We put the data from the test set into each model we would like to compare, and from each we get a ranking of all feeders, as if each had processed a new set of sensor readings and was making its predictions about likelihood of future failure. We now have each model's predictions, and we also know what the actual outcomes were in the test period. If all actual feeder failures in the test period are at the top of a test ranking, we consider it to be the best possible ranking. If all are at the bottom, it's the worst possible. But given two rankings that seem to have similarly distributed the failures, we would like to determine if one is doing a better job than another.

To obtain a single score for ranking comparison, we use a well-established ML scoring technique, based on a graph known as a **R**eceiver **O**perating **C**haracteristic (ROC) curve [9]. On an ROC graph, the Y-axis is the number of actual failures. The X-axis is our ranked test data. At each step along the X-axis, the Y value is the number of actual failures accounted for up to that point. Thus the curve always starts at (0,0) (no failures yet), ends in the upper right hand corner (by the bottom of the ranking, all failures have been accounted for), and increases monotonically (the number of accounted-for failures increases as we move down the ranking).

In the best-possible ranking described above, with all failures at the top of the ranking, the curve would rise sharply to the maximum value, and then continue horizontally. If our predictions are completely random, we would expect the actual failures to be uniformly distributed throughout the ranking, giving a straight diagonal line. A "good" ranking would have an ROC curve mostly above the diagonal. In the worst-case scenario, with all failures

at the bottom of the ranking, the curve would move along the X-axis (zero failures accounted for) until the last possible point, then rise sharply to the upper-right corner. These cases are illustrated in Figure 5 assuming 30 feeders with five actual outages.

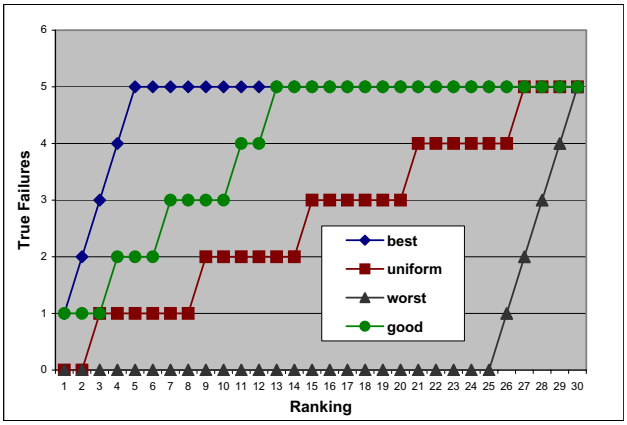

Figure 5. ROC Chart Comparison of Example Rankings

Note that if we normalize the area covered by this graph to one, the area under these four shapes are: nearly one, 0.5, somewhat greater than 0.5, and nearly zero, respectively. Thus the **A**rea **U**nder the ROC **C**urve (AUC) can function as our scoring metric, with anything above 0.5 indicating better-than-random performance (and with complex real-world systems, slightly better than random may be the best that can be achieved).

3.4 Aggregation by network results

The set of experiments described here were performed offline on recorded data. The goal of our experiments was to test the hypothesis that influence-based aggregation of data would not have a substantially negative effect on the quality of the Machine Learning results relative to models developed with full system data, while substantially increasing scalability by enabling parallel evaluation of quickly-computable small models. ConEdison is primarily interested in susceptibility rankings for the summer, which is when the system sees its highest loads. Demand for electricity is much lower in the winter, and feeder failures are much rarer. The training data included historical data from June and July 2005, a period during which there were around 300 feeder failures. We compared the AUCs of the generated rankings based on the count of actual failures in first three weeks of August 2005. There were about 70 failures in this period.

We used ConEdison's existing administrative hierarchy of networks to define the influence classes, both because the electrical distribution system had been physically constructed around this hierarchy, and because it corresponds to a planned future broadband-over-powerline sensor data transmission network.

We tried applying two very simple aggregation functions, averaging and root-mean-square, over the values from each network, as earlier investigations had shown that the algorithms seemed to give better results with averages as opposed to outliers, e.g., min or max. We ran the machine learning algorithms (both Martiboost and SVMs) independently on each network, training them on a two-influence-class system, with the raw data from that network combined with aggregated data from other networks. We also trained a model on the unaggregated, full data set, and looked at that model's performance in each network. We compared the AUC results from per-network training on aggregated data to the per-network results from training on all (unaggregated) data. The results are summarized in Tables 1 and 2 below.

Each row shows the various AUC scores for a particular network (networks are identified by a number and district, where district $\in \{B, Q, M, X\}$, for Brooklyn, Queens, Manhattan, and the Bronx). Each pair of columns shows the results for Martiboost and SVMs under a particular data reduction technique. We can see that the aggregations do not seem to impair the learning.

We have a number of ideas about why the experimental results show a possible improvement. One hypothesis is that some feeders in distant boroughs randomly resemble feeders in the network being modeled, and are incorrectly given extra weight in the training, and then prove to be misleading examples. The aggregation process eliminates these spurious matches, while still giving valid training examples.

The key benefit we were looking for was improved speed of computation as a consequence of smaller input set size, about one-twentieth of the full set, and that is what we observed. Running on a 3GHz Pentium4 system with 4GB of RAM, computing a global model took an average of 118s, while the localized models never took more than 6s, and averaged 1.4s on the same machine.

4. Related Work

4.1 Level of Detail in Graphics

As long ago as 1976, J. Clark suggested rendering objects with varying degrees of detail based on their apparent size to the viewer, allowing the display of many more objects than would otherwise be possible [5]. This technique, known as Level of Detail, has been refined over the years by computer graphics researchers to support automatic generation of appropriately simplified models based on distance

Table 1. Comparison of Level of Detail approach to training on full unaggregated data set using Martingale Boosting

Net	Average	RMS	Unaggr.
1Q	0.738	0.667	0.464
1X	0.818	0.833	0.455
2B	0.467	0.600	0.356
2M	0.491	0.667	0.565
2X	0.571	0.500	0.357
3B	0.545	0.518	0.607
3M	0.179	0.143	0.571
3Q	0.733	0.533	0.067
3X	0.881	1.000	0.524
4B	0.311	0.511	0.583
4M	0.870	0.870	0.478
5B	0.364	0.455	0.318
5M	0.400	0.689	0.289
5Q	0.546	0.554	0.565
5X	0.636	0.400	0.400
6B	0.721	0.721	0.784
6M	0.435	0.391	0.261
6Q	0.464	0.464	0.500
7B	0.909	0.818	0.818
7M	0.491	0.491	0.534
7Q	0.511	0.576	0.630
9B	1.000	1.000	0.889
9Q	0.333	0.556	0.222
10M	0.933	1.000	0.800
11M	0.608	0.765	0.441
13M	0.692	0.296	0.462
16M	0.217	0.217	0.087
18M	0.375	0.250	0.375
19M	1.000	1.000	0.818
20M	0.636	0.750	0.273
24M	0.500	0.409	0.909
26M	0.636	0.636	0.364
27M	0.957	0.761	0.891
28M	0.364	0.455	0.667
34M	0.500	0.500	0.545
Avg	**0.583**	**0.607**	**0.489**
Stdev	0.216	0.203	0.195

Table 2. Comparison of Level of Detail approach to training on full unaggregated data set using Support Vector Machines

Net	Average	RMS	Unaggr.
1Q	0.492	0.595	0.571
1X	0.773	0.818	0.682
2B	0.467	0.489	0.356
2M	0.509	0.537	0.519
2X	0.393	0.071	0.714
3B	0.476	0.500	0.518
3M	0.179	0.000	0.536
3Q	0.667	0.067	0.667
3X	0.667	0.810	0.571
4B	0.267	0.467	0.289
4M	0.667	0.623	0.652
5B	0.455	0.500	0.545
5M	0.689	0.578	0.733
5Q	0.583	0.596	0.648
5X	0.600	0.300	0.400
6B	0.662	0.725	0.804
6M	0.435	0.565	0.565
6Q	0.464	0.464	0.857
7B	0.727	0.818	1.000
7M	0.497	0.578	0.416
7Q	0.500	0.638	0.449
9B	0.765	0.765	0.824
9Q	0.778	0.889	0.444
10M	1.000	0.933	0.733
11M	0.559	0.804	0.686
13M	0.615	0.731	0.731
16M	0.522	0.783	0.826
18M	0.000	0.429	0.143
19M	0.909	0.909	1.000
20M	0.545	0.273	0.818
24M	0.391	0.727	0.545
26M	0.000	0.364	0.167
27M	0.917	0.729	0.792
28M	0.091	0.091	0.364
34M	0.545	0.545	0.500
Avg	**0.553**	**0.539**	**0.598**
Stdev	0.158	0.240	0.174

from the viewer [10, 14].

4.2 Distributed Monitoring

Numerous distributed monitoring systems have been developed, some of them supporting some form of aggregation. The topic is of keen interest to multiple industries, including utilities, power, oil and gas, telecom, industrial, water and public utilities, agriculture and facilities management [17]. We mention only a few examples.

Ganglia [15] is a hierarchical distributed monitoring system designed for monitoring federations of parallel computing clusters. Its focus is on extremely low latency and overhead for basic logging and monitoring, and it is in use at

hundreds of sites. In contrast, the Level of Detail framework is oriented towards intelligent data aggregation for the consumption of advanced learning algorithms, at the expense of higher latency and overhead.

Astrolabe [18] is a distributed information management system using administratively-defined hierarchical aggregation to make vast amounts of data tractable. Latency is higher than with systems using event distribution trees, as it uses a gossip protocol for data dissemination. Nonetheless, it is highly scalable, robust, and secure. SDIMS [25] is also a distributed information management system inspired by Astrolabe, but replacing its underlying gossip protocol data dissemination mechanism with a sophisticated hierarchical Distributed Hash Table system. The dissemination strategy for aggregated information is based on popularity and the ratio of reads to writes. It might be possible to adapt either Astrolabe or SDIMS to support Level of Detail's data dissemination, replacing their aggregation criteria with our influence classes.

5. Future Work and Conclusions

5.1 Future Work

A near term goal is to apply the techniques described here to much larger data sets to better evaluate their scalability. We plan to look at feeders again, but now broken down into their components. A single feeder may be composed of dozens of distribution transformers and hundreds of cable sections and connecting joints. Analyzing feeders at this level will increase the problem size by two orders of magnitude, giving more insight into scaling issues.

Two important areas for future work are identifying optimal aggregation functions for a particular application, and investigating the relationship between various aggregation functions and different machine learning algorithms. There are many other machine learning algorithms beside the two we used in these experiments, such as the popular AdaBoost [8] Ideally, we would like the system to autonomically determine the optimal functions, chosen from some predefined set, and self-configure.

Another interesting area for future research is in the interplay between Level of Detail and the scale-resistant services. The system in this paper is designed to bend the data to the needs of the algorithm, but conceivably there are algorithms that could be made "Level of Detail-Friendly".

For instance, the machine learning technique of Boosting takes a large number of "weak learners", classifiers that do barely better than random, and iteratively combines them into an effective learning tool. Some recent work has investigated using similar techniques for combining many pairwise rankings (including many contradictory ones) into an overall ranking, such as the RankBoost algorithm [7].

Aggregations based on a partition of a system cannot be combined in this way — there's no overlap to guide the algorithm. However, we could use several different influence metrics (including random assignment) to induce multiple, overlapping partitions of a system. Each model trained on some small portion of the system, as defined by one of the metrics, would be a weak learner. Conceivably, the resulting models could be combined to give an excellent model of the entire system.

5.2 Conclusion

Advanced machine learning algorithms hold much promise for automatic model generation for distributed system monitoring and management. Bringing these computationally complex algorithms to bear on large scale systems will require advanced data aggregation and dissemination techniques. Using the Level of Detail approach described in this paper, it is possible to deliver a relevant, customized data stream, including information from a wide portion of the entire system, to nodes running scale-resistant services. Our preliminary investigation has shown that machine learning applied to these "locally influential" data sets gives results comparable to applying the algorithms to the system as a whole, with an order-of-magnitude speed increase.

6. Acknowledgments

The Programming Systems Laboratory is funded in part by National Science Foundation grants CNS-0426623, CCR-0203876 and EIA-0202063 and in part by Consolidated Edison Company of New York.

We gratefully acknowledge the contributions of the faculty, staff, students, and many ConEdison employees who have contributed to this research, including Arthur Kressner, Matthew Koenig, Mark Mastrocinque, and Serena Lee of ConEdison, and David Waltz, Roger Anderson, Albert Boulanger, Marta Arias, and Hila Becker of Columbia University.

References

[1] C. I. Almanac. PCs In-Use Surpassed 820M in 2004, 2005. http://www.c-i-a.com/pr0305.htm.

[2] S. Banerjee, B. Bhattacharjee, and C. Kommareddy. Scalable Application Layer Multicast . In *Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, Pittsburgh, Pennsylvania, 2002.

[3] S. Banerjee, C. Kommareddy, K. Kar, B. Bhattacharjee, and S. Khuller. Construction of an Efficient Overlay Multicast Infrastructure for Real-time Applications. In *IEEE Infocom*, San Francisco, 2003.

[4] C. J. C. Burges. A Tutorial on Support Vector Machines for Pattern Recognition. *Data Min. Knowl. Discov.*, 2(2):121–167, 1998.

[5] J. H. Clark. Hierarchical Geometric Models for Visible Surface Algorithms. *Commun. ACM*, 19(10):547–554, 1976.

[6] C.-N. Fiechter. Efficient Reinforcement Learning, 1994.

[7] Y. Freund, R. Iyer, R. Schapire, and Y. Singer. An Efficient Boosting Algorithm for Combining Preferences. *Journal of Machine Learning Research*, 4, 2003.

[8] Y. Freund and R. Schapire. A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting. *Journal of Computer and System Sciences*, 55(1), 1997.

[9] J. Hanley and B. McNeil. The Meaning and Use of the Area Under a Receiver Operating Characteristic (ROC) Curve. *Radiology*, 143:29–36, 1982.

[10] H. Hoppe. Progressive Meshes. In *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*. ACM Press, 1996.

[11] T. Joachims. Making Large-Scale Support Vector Machine Learning Practical. In *Advances in Kernel Methods: Support Vector Learning*, pages 169–184. MIT Press, 1999.

[12] L. P. Kaelbling, M. L. Littman, and A. P. Moore. Reinforcement Learning: A Survey. *Journal of Artificial Intelligence Research*, 4, 1996.

[13] P. Long and R. A. Servedio. Martingale Boosting. In *18th Annual Conference on Learning Theory*, Bertinoro, Italy, 2005.

[14] D. Luebke and C. Erikson. View-Dependent Simplification of Arbitrary Polygonal Environments. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*. ACM Press/Addison-Wesley Publishing Co., 1997.

[15] M. L. Massie, B. N. Chun, and D. E. Culler. The Ganglia Distributed Monitoring System: Design, Implementation, and Experience. *Parallel Computing*, 30, 2004.

[16] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Schenker. A Scalable Content-addressable Network. In *Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, pages 161–172, San Diego, CA, 2001.

[17] Remote Site & Equipment Management Magazine. web site, 2006. http://www.remotemagazine.com.

[18] R. V. Renesse, K. P. Birman, and W. Vogels. Astrolabe: A Robust and Scalable Technology for Distributed System Monitoring, Management, and Data Mining. *ACM Trans. Comput. Syst.*, 21(2):164–206, 2003.

[19] A. Rowstron, A.-M. Kermarrec, M. Castro, and P. Druschel. SCRIBE: The Design of a Large-Scale Event Notification Infrastructure. In *NGC2001*, London, 2001.

[20] R. E. Schapire. The Boosting Approach to Machine Learning: An Overview. In *MSRI Workshop on Nonlinear Estimation and Classification*, Berkeley, CA, 2001.

[21] K. Singh and H. Schulzrinne. Using an External DHT as a SIP Location Service. Technical Report CUCS-007-06, Columbia University, New York, NY, Feb 2006.

[22] I. Stoica, R. Morris, D. Karger, M. Kaashoek, and H. Balakrishnan. Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications. In *ACM SIGCOMM*, pages 149–160, San Diego, CA, 2001.

[23] L. G. Valiant. A Theory of the Learnable. *Commun. ACM*, 27(11):1134–1142, 1984.

[24] X. Xu, H.-g. He, and D. Hu. Efficient Reinforcement Learning Using Recursive Least-Squares Methods. *Journal of Artificial Intelligence Research*, 16:259–292, 2002.

[25] P. Yalagandula and M. Dahlin. A Scalable Distributed Information Management System. *Proceedings of the 2004 conference on Applications, technologies, architectures, and protocols for computer communications*, 2004.

[26] S. Q. Zhuang, B. Y. Zhao, A. D. Joseph, R. H. Katz, and J. Kubiatowicz. Bayeux: An Architecture for Scalable and Fault-tolerant WideArea Data Dissemination. In *Eleventh International Workshop on Network and Operating System Support for Digital Audio and Video*, 2001.