An Event System Architecture for Scaling Scale-Resistant Services

Thesis proposal

Philip N. Gross
Department of Computer Science
Columbia University
phil@cs.columbia.edu

Advisor: Prof. Gail E. Kaiser

December 9, 2005

Abstract

Large organizations are deploying ever-increasing numbers of networked compute devices, from utilities installing smart controllers on electricity distribution cables, to the military giving PDAs to soldiers, to corporations putting PCs on the desks of employees. These computers are often far more capable than is needed to accomplish their primary task, whether it be guarding a circuit breaker, displaying a map, or running a word processor. These devices would be far more useful if they had some awareness of the world around them: a controller that resists tripping a switch, knowing that it would set off a cascade failure, a PDA that warns its owner of imminent danger, a PC that exchanges reports of suspicious network activity to its peers to identify stealthy computer crackers.

In order to provide these higher-level services, the devices need a *model* of their environment. The controller needs a model of the distribution grid, the PDA needs a model of the battlespace, and the PC needs a model of the network and of normal network and user behavior. Unfortunately, not only might models such as these require substantial computational resources, but generating and updating them is even more demanding. Model-building algorithms tend to be bad in three ways: requiring large amounts of CPU and memory to run, needing large amounts of data from the outside to stay up to date, and running so slowly that can't keep up with any fast changes in the environment that might occur.

We can solve these problems by reducing the scope of the model to the immediate locale of the device, since reducing the size of the model makes the problem of model generation much more tractable. But such models are also much less useful, having no knowledge of the wider system.

This thesis proposes a better solution to this problem called *Level of Detail*, after the computer graphics technique of the same name. Instead of simplifying the representation of distant objects, however, we simplify less-important data. Compute devices in the system receive streams of data that is a mixture of detailed data from devices that directly affect them and data summaries (*aggregated* data) from less directly influential devices. The degree to which the data is aggregated (i.e., how much it is reduced) is determined by calculating an *influence metric* between the target device and the remote device. The smart controller thus receives a continuous stream of raw data from the adjacent transformer, but only an occasional small status report summarizing all the equipment in a neighborhood in another part of the city.

This thesis describes the data distribution system, the aggregation functions, and the influence metrics that can be used to implement such a system. I also describe my current towards establishing a test environment and validating the concepts, and describe the next steps in the research plan.

Contents

1 Introduction

Organizations are deploying larger and larger numbers of networked computational units, with node counts planned to increase into the millions [2]. 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.

These compute nodes generally have a local task, but are also contributing towards a higher-level goal. A circuit-breaker is protecting a specific piece of equipment, but, in the ”smart grid” of the future, will also try to maintain the health of the whole electrical distribution system [3]. A soldier has an immediate tactical goal, but is also part of a larger strategy. A particular PC tries to protect itself from malicious outsiders, but in new Collaborative Intrusion Detection Systems, it is also helping the whole network identify broader attacks [34].

In order to contribute to these higher goals, nodes with sufficient capability might want to have some idea of the bigger picture in the form of a periodically-updated computational *model* of their wider environment. A model allows a node to answer simple “what-if” questions and predict the effect of system changes. The frequency of model updates should be of the same order as the rate of significant system changes, whether it is seconds or hours. The updates to the model come in the form of data streams from other system nodes.[1]

Nodes might want to maintain their own independent models (and not simply depend on a remotely-maintained model) for a number of reasons. In some cases, as with circuit-breakers, decisions may need to be taken within milliseconds, leaving no time to consult with a remote system. Nodes may frequently be disconnected from the wider grid, particularly in the military context, and so need to ensure that they can operate independently. Also, there may be relevant data known to a local node, but not intended to be shared with the wider network for reasons of privacy or security. For instance, a smart electric meter would inform the wider grid of actual energy use, but treat planned consumption as private information.

For low-level nodes, these models may be small and from a local perspective (e.g. a smart electric meter bidding for electricity and controlling large appliances), while large central servers may be attempting to model and optimize the entire system (e.g. smoothing electrical demand over the entire disribution system). Rather than hand-generate countless individualized models, it is much more convenient for nodes to derive their own model from domain knowledge and observation. Machine Learning algorithms such as Support Vector Machines [6] and Adaboost [28] can do this. Further, once in possession of a model, a node can try to determine optimal *policies* for future behavior, by using the model to evaluate outcomes of potential actions.

Unfortunately, deducing the model from data or searching for optimal policies are compute-intensive operations, which are difficult to scale to very large data sets [19, 33]. Also, we would like to compute these models and policies in near-real time, to keep up with the current state

[1] There will always be extremely simple nodes in the system as well, able to emit a sensor reading, and perhaps react to on-off messages, but lacking the processor or memory to run a useful model. These minimal nodes will not be discussed further, and we will only be considering nodes above a minimal threshold of CPU, memory, and network connectivity.

of the system. One trivial way to cope with such computationally-intensive tasks on resource-constrained systems is simply to ignore any data more distant than some horizon h_∞, e.g. the circuit breaker feeds its models with data from directly connected cables (at distance 1), from transformers and joints connected to those (at distance 2), and ignores all data from anywhere else.

Instead, I introduce an aggregation-dependent approach which I call *Level of Detail*, as it is somewhat analogous to the computer graphics technique of the same name. I assume the existence of an efficiently-computable metric I call *influence*, representing the expected impact of one node's data on another's calculations. I also assume the existence of *aggregations*, which reduce one or more input streams of data to a single stream with a bounded rate, while preserving some useful information.

The key idea is to aggregate data streams 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. Drawing data from a wider range, even if aggregated, may benefit our model in a number of ways. While information from our first highly influential peer may be valuable, information from the second and further may be redundant. Less influential nodes may carry information from distant parts of the system, giving us a broader picture.

This idea is illustrated below. Instead of simply taking all data from our neighborhood, we take all data from a smaller region, and join it with information that comes from a much wider span of the system, scaled to emphasize the most relevant data.

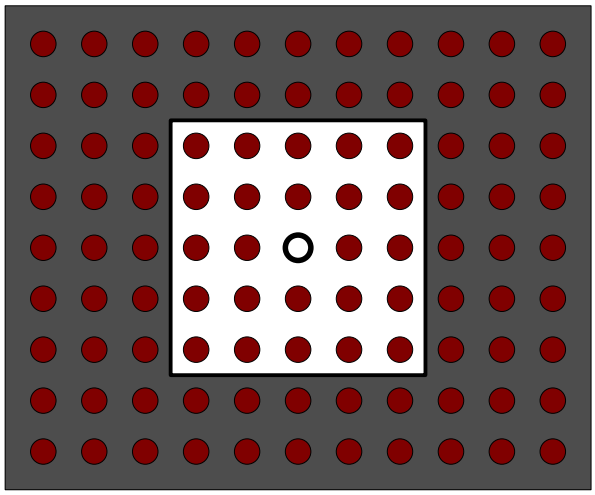


Figure 1: A simple approach to reducing model complexity is to accept all data from as many nearby nodes as possible, and ignore all other data.

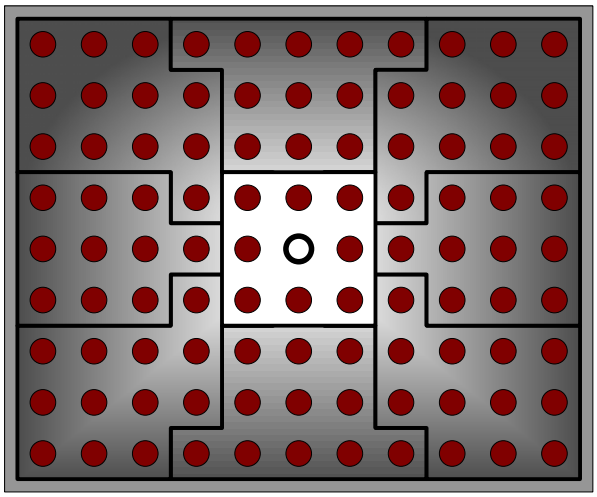


Figure 2: The Level of Detail approach is to accept all data from a smaller neighborhood, and use our remaining model capacity on aggregated data from a larger set of nodes.

However, intelligent data aggregation alone does not solve all of the problems of scaling scale-resistant services. While aggregation reduces the amount of data that needs to be delivered to any particular node, the aggregation operations may themselves be computationally expensive and are certainly data intensive. If every node required its own unique set of aggregated data,

the demands on the system would still be unacceptable. We will need a limited set of influence classes, allowing aggregations to be widely reused.

I will present results showing that when the Level of Detail technique is applied to actual data sets from Consolidated Edison of New York (ConEd), not only can intensive Machine Learning calculations be made more tractable via a factor of 20 reduction in input size, but in some cases, results actually showed improvement compared with learning on all raw data (an AUC score [defined later] of .60 compared with .51), despite a general computational learning theory guideline that more training examples give better results [32].

The ability of the Level of Detail approach to automatically scale large amounts of remote data to a quantity tractable for even fairly simple nodes could eventually lead to new forms of distributed capability. For large systems, *every* node of sufficient capability would now have the possibility of modeling and analyzing the system from its own perspective and to the best of its ability, identifying imminent problems and developing optimal policies to handle them. David Waltz has suggested the term *Distributed Awareness* for a system with such pervasive intelligence. Level of Detail data distribution could be a first step towards this vision.

This proposal is organized as follows: first, section 2 defines terms such as *event system* and *scale resistant*, and describes the requirements for an effective system for implementing Level of Detail. Section 3 describes the hypotheses and 4 describes the model for my solution. Section 5 discusses past and current related work as it relates to this problem. Current results in the context of ongoing joint research between Columbia and ConEd are described in section 6. The plan for completion of the thesis work is given in section 7. Expected Contributions, future work and conclusions are in section 8.

2 Problem, Definitions and Requirements

2.1 Definitions

These are some of the terms used in this proposal:

- **Event:** *Events* (also known as *messages* or *notifications*) are encapsulated pieces of information, usually small. The classic event representation is as an unordered set of attribute-value pairs, akin to a record from a relational database [1, 7, 29].

  A particularly important type of event describes the state of a system component at a point in time. These events can be used to derive a model of the system, and/or drive an existing system model. Over time, they create a time series data set, or data stream.

  While there may be many other event types in the system, in a large, sensor-rich system, these state-descriptive messages will usually comprise the vast bulk of event traffic.

- **Event System:** The main participants of interest in our problem domain are event stream producers, event stream consumers, and various transforms on the event streams. Systems for managing event streams are known as *Event Systems* (and also as *pub/sub systems*). Producers of event sequences are known as *publishers*, and consumers of event sequences

are *subscribers*. The primary goal of an event system is to get every published event to each of its subscribers. For a particular published event, there may be zero, one, or many subscribers.

- **Event System Architecture:** A software architecture designed to support advanced pub/sub features is an Event System Architecture. Usually they are constructed as overlay networks, leveraging an existing TCP/IP internetworking substrate.

  Some examples of advanced capabilities are:

  - **Content-Based Routing** (CBR), which matches an event to subscribers based on predicate functions (*filters*) that can examine the entire event, in constrast to the older channel-based systems, where subscribers subscribe to a channel, and receive all messages published to it [7].
  - **Transformations** of the event stream, including aggregation, unit conversion, message annotation, etc. [35]
  - Alternate **Event Types**, particularly structured events, as opposed to the typical attribute-value pair event format [12].
  - An advanced **Distribution Architecture** to support large numbers of participants on a particular channel. The system will usually construct and maintain a large, low-latency distribution tree, with limited *degree* (inputs and outputs) at any node.
  - **Autonomic Behavior**, where the system has enough self-knowledge and control to continually adjust and optimize itself [13].

- **Service:** A piece of software that does something useful. The term is used in the broad software engineering sense, as opposed to the specific operating system sense. The services of interest for this thesis will be useful throughout the system, from the global to the local level, and involve a system model, constructed from and/or driven by system-status events.

  An example service might use a survivability model to try to predict infrastructure component failures based on the current and predicted future environment. Another might use a threat assessment model to identify and rank the most pressing dangers to a military unit.

- **Scale-resistant Service:** I use the term scale-resistant to refer to model-related services that become intractable for very large models. The particular services we have encountered during the ConEd project that fall into this category are Machine Learning (ML) algorithms, which can construct a model of a system from observations, and Reinforcement Learning (RL) algorithms to find optimal policies given a model. 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, may itself be a non-trivial operation, but is generally of a lower order of complexity than ML and RL which actually construct models.

These scale-resistant algorithms are generally $O(n^2)$ or worse, where n is the number of training examples for ML and the number of model features for RL. Additionally, direct implementations of some ML algorithms, e.g., Support Vector Machines (SVMs), will have $O(n^2)$ space complexity as well, although more sophisticated implementations avoid this at the cost of slower operation [32, 10, 6, 19].

2.2 Problem Statement

We would like to implement Level of Detail-based data distribution. To achieve this we must overcome three 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, possibly on a reduced model of the system, while still producing useful results.
2. **Network Traffic:** The network will generally not have enough capacity to send all status events to a single node, which will in turn be limited by have a maximum rate at which it can accept events. We will need to limit the total traffic our data distribution requires of the system, and also limit the event arrival rate at each participating node. We will need to do this even for very large systems, while still providing sufficient data for our services to produce useful results.
3. **Dynamic System:** We cannot assume that the structure of our system is unchanging. Indeed, it is virtually impossible for a system of millions of nodes to remain 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.

Arbitrary event support is not a research goal of this thesis, but I will to support legacy and structured event types as necessary, beyond attribute-value sets. For instance, if events are being generated as processing-friendly fixed-format records, I will avoid expending computation in order to force them in and out of a larger and less-efficient format such as an attribute-value representation.

It is also worth noting that any large, distributed software system raises security issues. Event systems are particularly tempting targets for insider threats, thanks to their ability to take a single event and multiply it into a packet storm. However, a security framework for event systems is beyond the scope of this thesis.

3 Hypotheses and Proposed Approach

First I will define some additional terms, and briefly describe the Level of Detail approach. Then I will list my hypotheses, and describe the proposed model.

3.1 Aggregation

For the purposes of this proposal I will define an *aggregation* as a function that takes one or more streams of events as input and outputs a stream with a bounded event rate. The worst-case output rate will be lower than the worst-case sum of the input rates.

Some examples of aggregations under this definition are:

1. A bounded buffer model: Pass all incoming events through until the quota for this time quantum is reached, then discard all events until the next time quantum. This may result in a distorted view of the data, but is the easiest mechanism to implement.
2. For numerically-valued events, return the count, mean, and variance of all received events over the time quantum. This is useful if we are only interested in the statistical properties of the event stream.
3. Return a single stream synthesized from multiple streams by calculating the mean (or the max, or the median) across all input streams each time quantum. This is useful if we expect a number of streams to be highly correlated with each other.
4. Fit a polynomial to the incoming data points, and return its coefficients each time quantum. This is appropriate when we are interested in the streams' trend over time.
5. Use some stochastic function to reduce the data, e.g. drop each event with probability p, but still subject to a strict quota per time quantum. A random sample is a prerequisite for some statistical operations.

Note that aggregations under this definition are not necessarily lossy, as this definition considers only the raw event rate, not the information content. As a trivial example, duplicate input streams could be replaced with a single output stream.

3.2 Aggressiveness

We would like to characterize the data reduction achieved by a particular aggregation, which I term *aggressiveness*. I 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 aggregators may take α as a parameter, for instance letting events pass with probability α, and otherwise dropping them.

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 data rates.

3.2.1 Influence

When a particular node is planning to run a service e.g., machine learning, the service will require system data from which to construct a model. The node running the service may be anything from a resource-constrained embedded device to a powerful compute server. In any case, it will have a particular input budget based on how much data the service can consume while still producing results within the needed time. We decide how to spend our input data budget based on a metric I refer to as *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 some sort of correlation with the amount of impact that data from N_s will have on the outcome of a computation 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 $Infl(N_s, self) = 0.5$, then a randomly selected half of the data from N_s can be discarded.

3.3 Proposed Approach

My solution is inspired by a technique from the field of computer 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 [8]. 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 from the viewer [15, 21].

The Level of Detail technique for data dissemination is similar in structure to the graphics technique, but addresses the problem of too much complexity in our data streams instead of in our renderable objects. Instead of distance from the viewer, I use the influence metric described above. Instead of polygon and vertex reduction, I apply scalable aggregation functions to data streams, with increasingly strong aggregation as influence decreases.

In order to implement such a system, many event streams will need to be aggregated. Also, the output of aggregations will need to be disseminated to many nodes. In order to support the potentially large numbers of participants listening or sending to a single source, and potentially high rates of event generation, we will need an event distribution architecture more complex 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 [27], Bayeux [36], NICE [4], and OMNI [5].

One particular requirement for Level of Detail is that the event distribution system scale for aggregation as well as it does for dissemination. Although the two problems are symmetric, I have not been able to find a modern overlay event system designed to handle aggregation with the same efficiency as dissemination. I will describe a fairly simple protocol below which I can

use for research, but I am open to using existing systems if they can be modified for scalable aggregation.

An additional requirement is that subscribers and aggregators need to find publishers in order receive events. Again, scalable, robust, distributed indicies are a well-studied problem, and I intend to use an existing solution, either SIP [16], a Peer-to-Peer (P2P) system such as Chord [31] or the Content Addressable Network (CAN) [25], or recently developed hybrids [30].

3.4 Hypotheses

3.4.1 Network Hypotheses

1. For a large system running scale-resistant services on many nodes, dynamic data reduction techniques will allow the total amount of data moving through the system, as well as the amount of data being processed at each node, to remain at feasible levels even as the system grows very large, while still allowing the services to produce useful results.
2. An event distribution system can be effectively "inverted," and used to aggregate together many data streams with the same efficiency it uses to deliver a single data stream to many subscribers.

3.4.2 Service Hypotheses

For an important set of scale-resistant services (specifically machine learning and reinforcement learning):

1. There exist efficiently computable metrics (influence) to guide the aggressiveness of our aggregations for each service.
2. There exist simple and adjustable aggregation functions that can reduce data streams to varying degrees, while preserving essential information.
3. If these scale-resistant services use data from a broad area of the system, aggregated proportionally to its influence, they will produce useful results of higher quality than could be achieved by simply reducing the number of inputs to the service.

While I hope to produce some guidelines for finding appropriate influence metrics and aggregation functions for any scale-resistant service, a full theoretical analysis of how to derive such parameters from service characteristics is beyond the scope of this thesis.

4 Model

4.1 Distribution Architecture

A straightforward mechanism to distribute events in a channel is designate a node as the *reflector* for that channel. All published events are sent to that node, which then sends it onward to all other nodes. However, as the number of participants on a channel rises, the reflector may not

be able to either process all of its inputs or send to all of its outputs before the next event arrives. This is especially problematic when the participants are hardware or software entities, as opposed to humans, and capable of much higher event-generating and processing rates.

In order to support these high rates, one must trade latency for throughput. This generally involves introducing extra nodes into the dissemination/aggregation tree. Unlike hardware multicast, overlay distribution systems must process events sequentially, so in the case of a single incoming event that must be forwarded on to n downstream consumers costing t time each, and an event arrival rate of λ, we must limit the number of downstream consumers to $\frac{\lambda}{t}$ to ensure that we will be finished by the time the next event arrives. Thus we will need to enforce a limited *fan-out*.

In order to allow scalable multi-stream aggregation, the situation is similar. If we are, for instance, calculating some function across multiple input streams, the time to process each incoming event in order to access its value may be significant relative to the arrival rate of new events. Thus we will only be able to average across a small number of streams, and must enforce limited *fan-in*.

The following two sections describe a simple but scalable event system, similar to the Overlay Multicast Network Infrastructure (OMNI) system. I intend to use this simple architecture for the short term to carry out feasibility experiments. The goal is to use one of the more complete systems from overlay networking specialists, if one can be found that can be easily adapted for efficient aggregation as well as dissemination.

4.2 Recuitment

When building our distribution tree, the important participants in the process are nodes that are capable of *forwarding* messages. In the dissemination case, they need to have available at least twice the output rate capacity as the input stream, i.e., be capable of forwarding an incoming event to at least two other nodes. For multi-stream aggregation, the input capacity must be at least twice the output capacity, i.e. be capable of aggregating at least two streams. Nodes incapable of this, due to resource constraints such as limited bandwidth, are termed *leaf nodes*.

When a node would like to subscribe to a stream, it finds the publisher and asks to be grafted onto the distribution tree. If the publisher has available slots, the subscriber is added as a direct child, and receives every event directly as it is published. If the publisher is already at maximum fan-out, it will suggest two of its non-leaf children as candidates to become *event forwarders* for the subscriber. The subscriber picks one and repeats the subscription request, until an open slot is reached. Eventually either an available slot will be found, or a node is reached that is fully populated with leaf nodes. In the case of an all-leaf-children node, a more capable node will be inserted above one of the leaves. Ideally this will be the subscription-requester itself, but if it is a leaf itself, a new node will be *recruited* from the set of registered nodes with spare capacity.

Aggregation recruitment works in a similar fashion. An aggregator subscribes to streams containing data to be aggregated as it discovers them. When it reaches its maximum fan-in, it requests that existing sources aggregate new streams on its behalf. If they lack capacity to do so, the aggregator will recruit a new node from the available pool.

4.3 Distribution Optimizations

"The Power of Two Random Choices" is a phenomenon popularized by Michael Mitzenmacher[22]. If we have n balls and n bins, and throw each ball into a randomly chosen bin, the largest number of balls in any bin, with a high probability, will be $\frac{\log n}{\log \log n}$. If instead we put each ball into the least loaded of $d \geq 2$ bins, again chosen randomly, the most loaded bin is now likely to contain only $\frac{\log \log n}{\log d} + \theta(1)$. Thus by giving just two choices, we get a large improvement in worst-case performance, with only a constant factor of improvement for having more than two choices.

We leverage this when growing the distribution tree. If a publishing node has no further room for direct children, rather than either picking a random child or telling the subscriber to evaluate all the children to determine its next hop, it picks two randomly, and lets the subscribe choose between them. This captures most of the benefits of choice without a burdensome amount of evaluation. The primary metric for the sample implementation is latency, as that is what our staged architecture tends to add. Thus if fully-populated node A suggests child nodes B and C as candidate forwarding nodes to us, we compare latency(A, B) + latency(B, self) with latency(A, C)+latency(C, self). We then resend our subscription request to the node with the lower latency sum.

Distribution trees built in this fashion are unlikely to be optimal, and nodes may leave the system at any time. Therefore, I introduce mechanisms to try and reduce latency after tree construction. Periodically, special events are produced by publishers, allowing each forwarding node along the way to append its latency from its parent to the list. If a lower level node discovers that it has less latency to an ancestor node than that node's current child event forwarder, the lower level node can suggest switching places. The threshold for making such a such a swap is dependent on the magnitude of the improvement.

Additionally, leaf nodes are pushed to the bottom of the tree. Nodes with leaf children (or vacated slots) periodically announce this to their descendants. Should the node also have non-leaf grandchildren, they can be swapped with the leaves.

4.4 Registry and Channels

I assume the existence of a distributed, scalable registration facility that supports `put` and `get` operations. There has been extensive research in this area in recent years, and I believe the best option is to use one of the existing, well-developed systems. Peer-to-Peer (P2P) systems such as Chord [31] and CAN [25] provide a distributed repository with proven scalability properties. SIP [16] is a broadly-supported protocol for, among other things, storing and locating information about entities. Recent research has combined the two [30], using a P2P system as a substrate for SIP.

Nodes join the system (and thus enter the pool of recruitable nodes) by informing the distributed registry of their existence. Each node has associated identity value, which can be used for calculating influence metrics, an in-data-rate capacity and an out-data-rate capacity. As nodes participate in aggregation or dissemination trees which use up their capacity, the registration is

updated. Registering also automatically subscribes a node to the system notification channel. A future goal would be to look at giving nodes an attribute corresponding to CPU capability, to indicate their ability to handle CPU-intensive aggregation operations.

I also require publishers to register. Some event systems, notably Siena [7], are so-called *pure content-based-routing* systems. Such event systems typically do not require registration for event publishers. Any node can publish any sort of event at any time, and it will be evaluated against currently known subscriptions to determine routing. In contrast, *channel-based systems* require publishers to *advertise* their intentions ahead of publication, allowing the establishment of organized distribution trees. The two approaches could be seen as analogous to run-time versus compile-time evaluation.

For the scope of this thesis, I will be using a channel based system, as I am trying to set up efficient distribution structures. I use a flexible definition of channel, which is simply a template event that every published event will match. For instance, a transformer might advertise {type='xfmrReading', network='24M', source='V2077', amps, timeOfReading}. Events may include more information than the template, but never less.

Node subscriptions are against channels. A node submits its subscription request, which is in the form of a predicate to be matched against channel descriptions, and receives back a list of matching publishers. It can optionally create a persistent subscription request, and it will be notified should any matching publishers appear while the request is active. When a publisher registers, the advertisement is compared against persistent subscription requests and aggregations. If a match is found, a `NEW_SOURCE` message is sent to the subscribers and aggregators, giving them the opportunity to join the distribution tree. Note that the system design does not preclude the possibility of clients declaring custom *content-based filters* for the events on a channel, but these are not in the scope of this thesis.

Aggregations are also registered as individual entities. An aggregation may be either *universal* or *grouped*. Universal aggregations apply over all channels that match some *predicate* (a boolean-valued function). Grouped aggregations have a predicate as well as a *free field*, which functions similarly to an SQL `GROUP BY` clause. When the predicate matches, the free field is used to create a particular aggregation. For instance if the aggregation `current-by-net` has predicate `valueType = 'amps'` and our free field is `network`, channels with a valueType of '`amps`' and network of `23M` will be aggregated into a (`current-by-net`,`23M`) stream, while channels with a valueType of '`amps`' but a network of `8Q` will be aggregated into a (`current-by-net`,`8Q`) stream.

4.5 Influence Details

While in some cases we will calculate individual pairwise influence values, we can gain efficiencies by organizing the world as seen from the perspective of a particular node into "bands of influence," where I_0 is the class of nodes with the most influence on us, I_∞ is the class of nodes with the least influence, and $I_1, I_2, \ldots, I_n$ are classes of nodes with intermediate in-

fluence.[2] When describing computations from the perspective of a particular node, I will use the notation $Infl(N_s, self)$ to represent the influence of node N_s's data on "my" computation. Nodes N_0 in class I_0 often 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_∞ may have $Infl(N_\infty, self) = 0$, indicating that data from these nodes should simply be discarded.

Establishing an *influence horizon*, I_∞, has a substantial effect on the scalability of the system. Regardless of how many nodes n are in the system as a whole, the number of nodes sending data to a particular node remains bounded by k, the maximum number of nodes with a non-zero influence on a target node. With all nodes generating data each time step, the total amount of data that the network must move changes from $O(n^2)$ to $O(nk)$ and the amount of data any node receives is reduced from $O(n)$ to $O(k)$, $k << n$.

A given influence band may be *flat*, with all nodes in the band having the same influence on us, or it may be a *gradient*, with influence varying among nodes in the band. 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 band I_2, say, will have greater influence than any node in band I_3.

4.6 Influence Example

To give a concrete example in the case of our electric utility (see section 6.1 for an explanation of energy distribution terms), if we are attempting to run our machine learning model of the primary grid from the perspective of a particular feeder cable section, the data sources with the highest influence, I_0, will be those in our own feeder, as we are directly connected to them. I_0 is flat with all nodes having influence 1 (i.e. raw, unaggregated data). For the next band, I_1, network engineers have calculated a value for each feeder cable called "shifted load". If my feeder fails, other feeders in my network will have to pick up the load I'm no longer serving. Shifted load gives a breakdown of which feeders will take what amount of my load if I fail, and serves as an excellent scaling factor for the next stage of influence. Thus I_1 is a gradient band, varying from 0.6 to 0.9 as a function of the shifted load.

The feeders that are in my network, but don't directly pick up my shifted load, form I_2, $Infl(N_2, self) = 0.5$. I_3 consists of feeders in other networks in my region, $Infl(N_3, self) = 0.3$. I_4 is feeders in other regions, $Infl(N_4, self) = 0.2$. And finally, any data coming from outside my own distribution utility is outside my universe in I_∞, with zero influence. See section 6.1 for further discussion.

[2]The notation of I_0 for the most influential band and I_∞ for the least may seem counterintuitive. The concept of influence was originally termed "distance," but was deemed too suggestive of simple Cartesian distance. The band designations remained, however.

4.7 Influence Band Reuse

Consider a situation where the influence metric is based on physical Cartesian distance. Nodes cover a vast area, arranged in a rectangular grid, each one unit apart. We define three bands of influence, where I_0 is nodes within 30 units of us, I_1 is nodes between 30 and 60 units away, and I_∞ is nodes greater than 60 units away. All three are flat bands. This presents a scalability problem, because each node's $I_{0,1,\infty}$ are different from every other node's. We get no benefit from aggregation, because there's no reuse.

There are a number of ways to fix this. One is to organize the nodes into a hierarchy, possibly using pre-existing organizational boundaries. Organizational boundaries may be a good choice when available, as these may well have been defined to make influence boundaries and administrative boundaries coincide. Good organizational components (like good software components), have high internal cohesion and low external connectivity.

In the aggregation example above, we used the network boundary as a natural influence boundary, since not only is it an administrative unit, it is mostly isolated from surrounding networks.

The administrative definition of a hierarchy for dissemination of aggregated information is at the core of the Astrolabe system [26]. Aggregates only need to be calculated once per organizational unit and reuse is maximized. However, there is a cost in administrative complexity, and some organizational layers may still be too large to aggregate as a whole.

I support an alternative approach which I call *follower bias*, where one node's established bands of influence exert a slight attractive force on the attempts of later nodes to find their own boundaries, tending to pull them into the previously established bands. In the previous example of a regularly spaced grid of nodes, a cluster of nearby nodes might end up sharing the same I_1 and I_∞ definitions, trading accuracy for a substantial boost in efficiency.

To implement follower bias, we have new nodes first identify their own "local neighborhood" of I_0. They then ask up to k randomly selected nodes already present in their I_0 what the boundaries of their own local neighborhood look like. If one is similar enough to that of the new node, (e.g. 80% commonality), it subscribes to the other's existing aggregations instead of defining its own. See the illustration below.

5 Related Work

5.1 Scalable Event Systems

5.1.1 The Information Bus©

The Information Bus©was primarily described in single paper from ACM SIGOPS in 1993 [24]. The authors went on to found TIBCO (http://www.tibco.com), which has grown into the largest purveyor of commercial publish/subscribe software systems. Their client roster includes many Wall Street firms, AT&T, Intel, and many others.

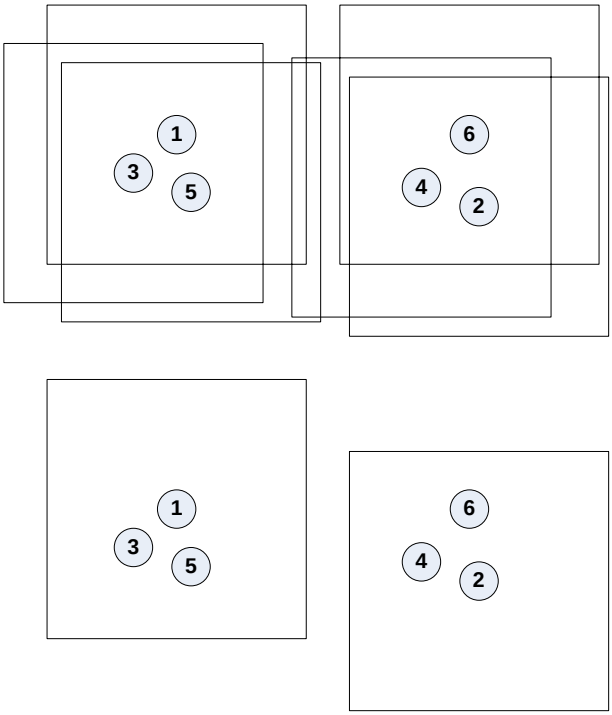


Figure 3: In the top set, each node (joining the system in numeric sequence) defines its own influence bands.
In the lower set, with follower bias, nodes 3-6 reuse the influence bands defined by 1 and 2

A primary contribution of their paper was to identify the set of problem requirements best suited to an event-based solution:

- Continuous operation: It is unacceptable to bring down the system for upgrades or maintenance.
- Dynamic system evolution: The system must be capable of adapting to changes in application architecture and in the type of information exchanged. It should also support the dynamic integration of new services and information.
- Legacy system integration: New software must be able to interact smoothly with existing software, regardless of the age of that software.
- Fault-tolerance + scalability: it must not have a single point of failure. The system must scale in terms of both hardware and data.
- Tolerance for non-Byzantine failures: Nodes and the network may fail, and it is assumed that these failures are fail-stop and not Byzantine

From the last bullet above, another contribution of the paper was the identification of pub/sub systems Achilles heel for fault tolerance: Byzantine failures could be devastating.

5.1.2 Gryphon

The Gryphon system [1] was developed by Robert Stroms group at IBM Research. Like TIB, Gryphon has been used in commercial systems. IBM has used it for sporting-event websites, so that client browsers can be updated with the latest scores by means of a Gryphon-client applet, and thus without burdening the HTTP server.

Gryphon models event streams from producer to consumer as an information flow graph, along which various transformations, aggregations, etc. may occur. The Gryphon work is among the first to define Content-Based Routing. The project was also among the first to formally define the matching problem: given a large set of predicates and a message, how can one efficiently find the subset of the predicates that is satisfied by the message.

Gryphons primary model for events and queries is what I call the database event model: events are unordered sets of attribute-value pairs, similar to a database record, and filters are variants on SQL-WHERE clauses over the event space. For efficient matching, Gryphon organizes the filters into a Trie structure, so that multiple predicates can be tested in parallel.

5.1.3 Siena

The Siena Event System, developed at the University of Colorado at Boulder, has an elegant approach to routing events under the database event model [7]. Siena routers are organized by an administrator into an acyclic tree structure. Events are unordered attribute-value sets, and queries are similar to SQL WHERE clauses.

Siena is a pure content-based routing system, allowing any participant to emit any message at any time, and determining appropriate routing based on run-time examination of published messages.

5.1.4 ECho

The ECho system from Georgia Tech [9] is a minimal but extremely fast system designed for Grid computing, where many computing elements are exchanging data, and low latency is critical. ECho is a distributed publish/subscribe events system. The main distinctive aspect is its focus on lowering latency by delivering data in its original raw binary form (native data representation) whenever possible. ECho supports user defined functions written in a C-like language operating on event streams, which would be useful for aggregation implementation. However, it uses a static, administratively configured network topology, making it more suited to small distributed systems.

5.1.5 REBECA

The REBECA system from the Darmstadt University of Technology uses covering and merging techniques similar to Siena [23]. They have introduced the idea of scopes, which group segments of the publish-subscribe network into higher-level constructs, bearing some resemblance to influence bands [11]. However, their vision of “scopes” is more of an abstract namespace,

as opposed to a aggregation-scaling mechanism. They have done work on formal correctness proofs for portions of their system.

5.1.6 YANCEES

YANCEES (Yet ANother Configurable Extensible Event Service) is a fully modular event system [12]. The system itself is focused on full configurability and pluggability, which makes it a good candidate for constructing "overlays on overlays." It can run support user-defined event types running over standard or custom event substrates, such as Siena or the Java Messaging System.

5.1.7 Astrolabe

Astrolabe [26] is a distributed information management system using administratively-defined hierarchical aggregation to make vast amounts of data tractable. The aggregation functionality is restricted to an SQL-like language, and 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.

5.2 Distributed Registries

The Level of Detail technique needs some sort of distributed registry so that aggregators can find publishers. Two major candidates are Peer to Peer systems and SIP. P2P systems such as Chord [31] and CAN [25] present a simple put/get interface to the user, while using clever functions on hashes of the item's key to locate the data on a potentially large set of participating peer systems. SIP [16] is a versatile protocol for establishing sessions between entities on the internet, including locating them despite movement and multiple user names. Recent work has applied some of the structures of P2P systems to SIP messages directly [30], blending the scalability and robustness of Distributed Hash Table systems with the capabilities of SIP.

5.3 Presence and Chat Systems

Chat and presence systems, such as Internet Relay Chat (IRC), AOL Instant Messenger, MSN Messenger, ICQ, and others, probably carry the most event-like traffic on the Internet. When a member connects to or disconnects from the system, his or her status change is published to all of that users friends. During a chat session, particularly multiway, each message sent by a user is published to the other participants. These systems have been engineered to scale to millions of simultaneous users.

In the research and open source communities, SIMPLE [18], an extension of the SIP signaling protocol, and XMPP [17], an IETF version of the Jabber protocol, have developed protocols and standards for instant-message and presence applications. SIMPLE is a sophisticated standard based on a peer-to-peer architecture. Jabber is a simple and centralized architecture which has proven popular for casual use.

The fundamental difference between the presence and chat systems described above and the problem domains discussed in this proposal is the nature of the participants (and the corresponding consequences for the architecture). Participants in presence and chat systems are humans, connected to social circles, usually numbering fewer than 100 people. Humans will usually not generate data that needs to be published at a rate of more than one or two items per second. Under these constraints, an appropriate architecture, and the one usually adopted by the above systems, is to have a single node responsible for reflecting information to participants. E.g., in a simplified view of the SIMPLE system, a Presence Agent accepts SUBSCRIBE messages and issues NOTIFY messages when appropriate to all SUBSCRIBEd presentities.

5.4 IP Multicast

IP Multicast is the subject of many RFCs and IETF working groups. Much advanced research on event dissemination occurs here, although their focus, appropriately, tends to be on embedded systems, the devices at the heart of the Internet. Among the relevant documents are the Internet Group Management Protocol RFCs (1112, 2236, and 3376) and Protocol Independent Multicast – Sparse Mode (2117 and 2362), and the IETF working groups on Multicast & Anycast Group Membership (magma), Border Gateway Multicast Protocol (bgmp), Source Specific Multicast (ssm), Mobile Ad-Hoc Networks (manet), Multicast Security (msec), and Reliable Multicast Transport (rmt).

5.5 Machine Learning

Learning algorithms themselves are not the focus of this thesis, so I will simply mention some of the major papers in the field. Valiant [32] is one of the founders of computational learning theory, the mathematical analysis of machine learning algorithms. Support Vector Machines [6] are one of the more popular machine learning techniques in use today, along with variations on Boosting [28]. The actual algorithm we are primarily using for the ConEd project is Martingale Boosting [20], developed by Phil Long and Rocco Servedio particularly for the project.

6 Feasibility

6.1 The ConEd Project for Predicting Feeder Cable Failure

Electrical infrastructure has four main parts:

1. *Generation* involves a prime mover, typically the force of water or steam on a turbine, spinning a dynamo and generating large amounts of electrical current.
2. *Transmission* sends the current at very high voltage (hundreds of thousands of volts) to substations close to the customers.
3. The *primary grid* sends electricity at high voltage (tens of thousands of volts) from substations to local *transformers*, over cables called *feeders*, usually with length less than

10km, and with a few tens of transformers per feeder. Feeders are composed of many *feeder sections* connected by *joints*.

4. The *secondary grid* takes electricity at normal household voltage from local transformers to individual customers.

These components are illustrated below.

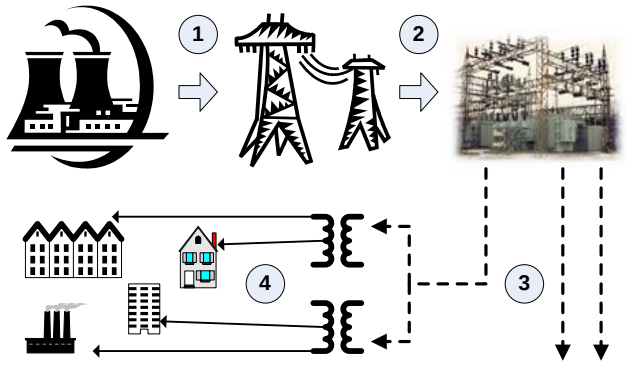


Figure 4: The Electrical Distribution Grid

The primary grid is organized into *networks*, each composed of a substation and its attached feeders. The networks are largely electrically isolated from each other, to limit the spread of major problems.

The feeders of the primary grid are system critical and have a significant failure rate, and thus much of the daily work of the electricity distribution utility involves their monitoring and maintenance, as well as their speedy repair or replacement on failure.

One of the key goals of the current ConEd-Columbia collaboration is to predict which feeders are most likely to fail using machine learning techniques. The machine learning algorithms are trained on learning sets that treat each feeder as one example. The feeders are described with several hundred attributes, some of which are derived from static data, such as age and length, and others summarized from dynamic data, such as how much current was run through the feeder or how many outages it suffered over a particular period. Finally, the response variable is the number of unscheduled, spontaneous outages experienced by that feeder.

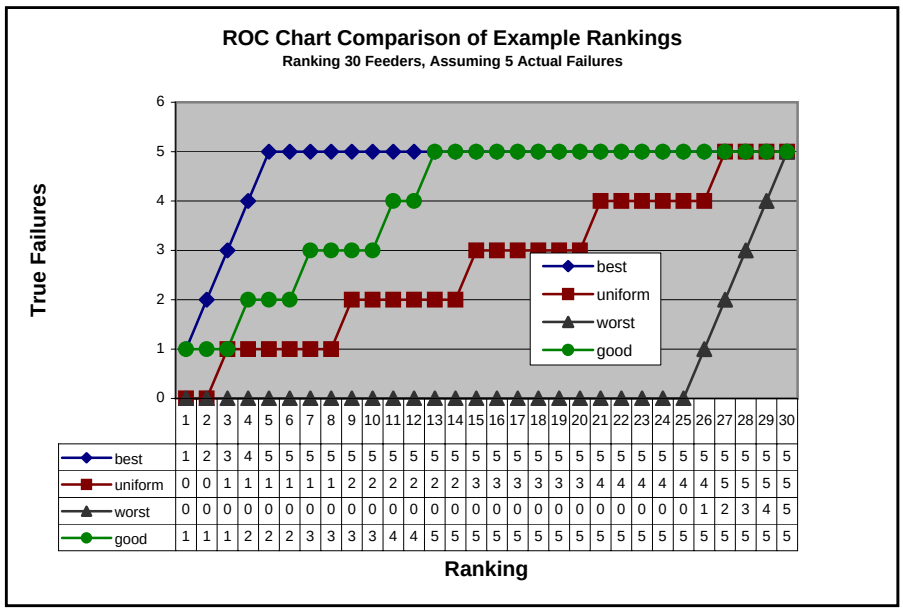
The machine learning algorithms attempt to create generalizable models based on this training set, such that if given a new, previously unseen set of data on a set of feeders, it will be able to rank them in order of their susceptibility to failure. The machine learning techniques we are currently using are Support Vector Machines (SVMs) [6], and a new system called Martingale Boosting (Martiboost) [20] specifically developed for the ConEd project.

6.2 AUC Scoring

In order to evaluate our results, we need some mechanism to compare our rankings, to determine if one is “better” than another. We will need to 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. We now have each model's predictions, and we know what the actual outcomes were in our test period. If all actual feeder failures in the test period are at the top of a test ranking, we know it's the best possible ranking. If all are at the bottom, we know it's the worst possible. But given two rankings that spread the actual failures around the ranking and look fairly similar, can we determine if one is doing a better job than another?

To obtain a single comparable score, we use a graph known as a **R**eceiver **O**perating **C**haracteristic curve (ROC) [14]. The Y-axis is the number of actual failures. The X-axis is our ranking. At each step along the X-axis, the value is the number of actual failures accounted for up to that point. Thus the curve always starts at (0,0), ends in the upper right hand corner, and increases monotonically. In our ideal ranking described above, 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, giving a straight diagonal line. In the worst-case scenario, the curve would move along the X-axis until the last possible point, then rising sharply to the upper-right corner. Note that if we normalize the area covered by our graph to one, the area under these three shapes is: nearly one, 0.5, and nearly zero, respectively (See Figure 5 below). 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.



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
best	1	2	3	4	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5
uniform	0	0	1	1	1	1	1	1	2	2	2	2	2	2	3	3	3	3	3	3	4	4	4	4	4	4	5	5	5	5
worst	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	2	3	4	5
good	1	1	1	2	2	2	3	3	3	3	4	4	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5

Figure 5: Comparison of three idealized rankings (best, uniform, and worst) and an example "good" (better than random) ranking

6.3 Aggregation by network results

Over the course of our current one-year project with ConEd, we have accumulated a large amount of data, currently around 55GB and growing by several GB per week, containing in

particular over 800 million feeder section status readings. This represents a small but non-trivial sampling of the data ConEd is reading from its network all the time. We expect to be able to continue extensive experimentation with this data, assuming cooperation between Columbia and ConEd continues. I describe some of those planned experiments in the Future Work section.

Much of our current effort has gone towards gaining access to the available data sources, storing them efficiently, reducing them to a form which is tractable for machine learning, and developing new machine learning techniques to produce useful results. This has been sufficiently successful to raise hopes of a longer-term partnership commitment which will allow much more scope for exploratory work and a thorough investigation of the concepts described in this proposal.

I have been able to do some experiments on the current data set. The goal was to test the hypothesis that reduction of data based on Level of Detail principles would not have a substantially negative effect on the Machine Learning. The training data set covered June and July 2005, a period during which there were around 300 feeder failures. The test data was the first three weeks of August 2005, with about 70 failures. I used ConEd's existing administrative hierarchy of networks to define the aggregation regions. I tried applying two very simple aggregation functions, averaging and root-mean-square, over the values from each network. I used our Machine Learning algorithms (both Martiboost and SVMs) on each network, training them on the raw data from that network combined with aggregated data from other networks. I compared the AUC results from my training to the per-network results from our current approach of training with all available data. The results are summarized in Table 1 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 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. Indeed, the improvement of RMS aggregation on Martiboost verges on the statistically significant, although given the high variance and small sample size, it is safer to simply say that based on this example, there is not much evidence of any degradation.

We have a number of ideas about why we see this possible improvement, even with vastly smaller input. My personal opinion is that some "uninfluential" 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.

It should also be noted that the time to compute a model for a single network is ten to twenty times faster than calculating the entire monolithic model. This gap will widen as the size of the model increases.

6.4 Simulation of large-scale network growth and adaptation

A large-scale simulation of the proposed networking architecture is planned for early next year, when a more detailed description of the projected future sensor and processor sets becomes

Table 1: Comparison of Level of Detail approach to training on full data set

Net	Average		RMS		Original	
	Marti	**SVM**	**Marti**	**SVM**	**Marti**	**SVM**
1Q	0.738	0.492	0.667	0.595	0.464	0.571
1X	0.818	0.773	0.833	0.818	0.455	0.682
2B	0.467	0.467	0.600	0.489	0.356	0.356
2M	0.491	0.509	0.667	0.537	0.565	0.519
2X	0.571	0.393	0.500	0.071	0.357	0.714
3B	0.545	0.476	0.518	0.500	0.607	0.518
3M	0.179	0.179	0.143	0.000	0.571	0.536
3Q	0.733	0.667	0.533	0.067	0.067	0.667
3X	0.881	0.667	1.000	0.810	0.524	0.571
4B	0.311	0.267	0.511	0.467	0.583	0.289
4M	0.870	0.667	0.870	0.623	0.478	0.652
5B	0.364	0.455	0.455	0.500	0.318	0.545
5M	0.400	0.689	0.689	0.578	0.289	0.733
5Q	0.546	0.583	0.554	0.596	0.565	0.648
5X	0.636	0.600	0.400	0.300	0.400	0.400
6B	0.721	0.662	0.721	0.725	0.784	0.804
6M	0.435	0.435	0.391	0.565	0.261	0.565
6Q	0.464	0.464	0.464	0.464	0.500	0.857
7B	0.909	0.727	0.818	0.818	0.818	1.000
7M	0.491	0.497	0.491	0.578	0.534	0.416
7Q	0.511	0.500	0.576	0.638	0.630	0.449
9B	1.000	0.765	1.000	0.765	0.889	0.824
9Q	0.333	0.778	0.556	0.889	0.222	0.444
10M	0.933	1.000	1.000	0.933	0.800	0.733
11M	0.608	0.559	0.765	0.804	0.441	0.686
13M	0.692	0.615	0.296	0.731	0.462	0.731
16M	0.217	0.522	0.217	0.783	0.087	0.826
18M	0.375	0.000	0.250	0.429	0.375	0.143
19M	1.000	0.909	1.000	0.909	0.818	1.000
20M	0.636	0.545	0.750	0.273	0.273	0.818
24M	0.500	0.391	0.409	0.727	0.909	0.545
26M	0.636	0.000	0.636	0.364	0.364	0.167
27M	0.957	0.917	0.761	0.729	0.891	0.792
28M	0.364	0.091	0.455	0.091	0.667	0.364
34M	0.500	0.545	0.500	0.545	0.545	0.500
Avg	**0.583**	**0.553**	**0.607**	**0.539**	**0.489**	**0.598**
Stdev	0.216	0.158	0.203	0.240	0.195	0.174

available. However, as ConEd moves on its own schedule, I am creating a local testbed using the Boost Graph Library to quantify the effect of two-random-choice on average path latency.

7 Research Plan and Schedule

Table 2 shows my plan for the completion of this research.

Table 2: Research plan and schedule for completion.
(✓ indicates a complete task, ◗ indicates an ongoing task, and ❑ indicates an upcoming task.)

Status	Task	Completion Date
✓	Semantic auto-discovery and transformation for event streams	2002
✓	Connectivity clique identification from network connection events	2003
✓	Investigation of modular architectures for event distribution systems	2004
✓	Data access and organization combined with efficient machine learning for near-real-time analysis of ConEd data	June, 2005
✓	Feeder connectivity model generation from "noisy" event data	Aug. 8, 2005
✓	Conduct Level of Detail feasibility experiments	Sep. 13, 2005
✓	Final end-of-year deliverables approval with ConEd for feeder susceptibility project	Oct. 5, 2005
✓	Write thesis proposal	Nov. 25, 2005
❑	Defend thesis proposal	Dec. 9, 2005
❑	Set up enhanced framework for working with ConEd data	February, 2006
❑	Set up network simulation testbed for evaluating Level of Detail performance on large networks	March, 2006
❑	Extended results on various types of aggregation and influence metrics	April, 2006
❑	characterization of effective influence metrics and aggregation algorithms for machine learning and reinforcement learning	June, 2006
❑	Thesis defense	August, 2006

8 Contributions, Future Work and Conclusion

8.1 Thesis Contributions

The expected contributions of this thesis are:

1. An **event system architecture** capable of supporting large numbers of participants on a single channel, both for **dissemination and aggregation**. While there currently exist many event systems capable of efficient event dissemination, there are none that can perform the symmetric task for event aggregation. As smart, networked systems proliferate, the goal will no longer be just to deliver information to those that need it, but to bring the chatter and traffic under control, and distill information from the data.

2. An architecture for allowing **scale-resistant services** to run on constrained nodes with high-quality results. Our smart devices have the potential to engage with the world outside

and offer new types of features. However, the algorithms are expensive and the amount of data is overwhelming. With access to intelligently scaled data, constrained devices can leverage the state of the art in machine learning.

3. **Example aggregation and influence functions** that have been shown to produce good results for particular combinations of scale-resistant services and data sets. ConEd and potentially other organizations are interested in making their systems smarter as fast as possible. A real, functioning example of machine and reinforcement learning tackling problems of the largest scale will not only help to make our electricity cheaper and more reliable, but will provide insights towards taming other scale-resistant services in the future.

8.2 Future Work

There are a number of aspects of event distribution which are not addressed by this proposal but offer many interesting questions for the future.

- **Security:** Event systems are inherently problematic from a security perspective. One of their main purposes is to take a single message and turn it into a blizzard of thousands. In the hands of a rogue insider, they are perfect for a Denial of Service attack. Event systems will not truly be welcome in corporate and government networks until we can make a system which is resistant to these sorts of threats.
- **Privacy:** While we can trivially encrypt traffic between nodes, are there workable solutions for truly private channels, where intermediate nodes cannot read our messages? The problem is daunting, given the fluid presence of both people and systems in event distribution networks. Recent research in fast, secure, group key generation is promising, however.
- **Event Types:** An ideal event system would allow users to define any sort of data type as events, and then let them supply their own predicate functions on those types for use as subscriptions. Even better would be if they could define their own predicates for use as filters, to be applied to every message. However, such a system would need to be resilient to unsafe, buggy, or malicious code.

One of the most interesting areas for future research is in the interplay between Level of Detail and the scale-resistant services. The systems in this proposal are designed to bend the data to the needs of the algorithm, but conceivably there are algorithms that could be made “Level of Detail-Friendly”.

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.

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.

And finally, there is the vision of Distributed Awareness, where model-based, predictive, proactive and cooperative systems are ubiquitous throughout a large system. From the large central servers to the small devices in remote locations, they are generating data for all who are interested, and consuming the observations and insights of others. Level of Detail would enable the most useful information to reach the most participants.

8.3 Conclusion

This proposal has presented an *event service architecture for scaling scale-resistant systems*, with a goal of allowing advanced model-generating algorithms to run on systems of limited capability, and to model systems that would otherwise be too large to be tractable. I have described both an event distribution architecture and a principled method for reducing the data needs of complex algorithms. I also showed preliminary results indicating that such scale-resistant services can still produce useful models even when working with substantially less input data. I hope that the work done for my thesis will bring new insights into the large-scale delivery of information, and also lead to much wider applicability for some of the most advanced techniques of Computer Science.

References

[1] Marcos K. Aguilera, Robert E. Strom, Daniel C. Sturman, Mark Astley, and Tushar D. Chandra. Matching events in a content-based subscription system. In *18th Annual ACM Symposium on Principles of Distributed Computing*, pages 53–61, 1999. Atlanta, Georgia, United States.

[2] Computer Industry Almanac. PCs In-Use Surpassed 820M in 2004, 2005. `http://www.c-i-a.com/pr0305.htm`.

[3] Roger Anderson and Albert Boulanger. Smart Grids and the American Way. *Power and Energy*, March 2004 2004.

[4] Suman Banerjee, Bobby Bhattacharjee, and Christopher Kommareddy. Scalable application layer multicast . In *Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, Pittsburgh, Pennsylvania, 2002.

[5] Suman Banerjee, Christopher Kommareddy, Koushik Kar, Bobby Bhattacharjee, and Samir Khuller. Construction of an Efficient Overlay Multicast Infrastructure for Real-time Applications. In *IEEE Infocom*, San Francisco, 2003.

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

[7] Antonio Carzaniga, David S. Rosenblum, and Alexander L. Wolf. Design and evaluation of a wide-area event notification service. *ACM Transactions on Computer Systems*, 19(3):332–383, 2001.

[8] James H. Clark. Hierarchical geometric models for visible surface algorithms. *Commun. ACM*, 19(10):547–554, 1976.

[9] G. Eisenhauer, F.E. Bustamante, and K. Schwan. Event services for high performance computing. In *9th International Symposium on High-Performance Distributed Computing*, pages 113–120, 2000. TY - CONF.

[10] Claude-Nicolas Fiechter. Efficient reinforcement learning, 1994.

[11] L. Fiege, M. Mezini, G. Mhl, and A. Buchmann. Engineering Event-Based Systems with Scopes. In *16th European Conference on Object-Oriented Programming (ECOOP'02)*, Mlaga, Spain, 2002.

[12] R. S. Silva Filho and D. Redmiles. Striving for Versatility in Publish/Subscribe Infrastructures. In *Fith International Workshop on Software Engineering and Middleware (SEM'2005)*, Lisbon, Portugal, 2005.

[13] A. G. Ganek and T. A. Corbi. The dawning of the autonomic computing era. *IBM Syst. J.*, 42(1):5–18, 2003.

[14] J.A. Hanley and B.J. McNeil. The meaning and use of the area under a receiver operating characteristic (ROC) curve. *Radiology*, 143:29–36, 1982.

[15] Hugues Hoppe. Progressive meshes. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*. ACM Press, 1996.

[16] IETF. RFC3261: SIP: Session Initiation Protocol, 2002.

[17] IETF. Extensible Messaging and Presence Protocol (xmpp) Charter, 2004.

[18] IETF. SIP for Instant Messaging and Presence Leveraging Extensions (simple) Charter, 2004.

[19] Thorsten Joachims. Making large-scale support vector machine learning practical. In *Advances in kernel methods: support vector learning*, pages 169–184. MIT Press, 1999.

[20] Phil Long and Rocco A. Servedio. Martingale Boosting. In *18th Annual Conference on Learning Theory*, Bertinoro, Italy, 2005.

[21] David Luebke and Carl 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.

[22] M. Mitzenmacher, A. Richa, and R. Sitaraman. The power of two random choices: A survey of the techniques and results. In P. Pardalos S. Rajasekaran and J. Rolim, editors, *Handbook of Randomized Computing*. Kluwer, 2000.

[23] Gero Mhl and Ludger Fiege. Supporting Covering and Merging in Content-Based Publish/Subscribe Systems: Beyond Name / Value Pairs. *IEEE Distributed Systems Online (DSOnline)*, 2(7), 2001.

[24] Brian Oki, Manfred Pfluegl, Alex Siegel, and Dale Skeen. The Information Bus: an architecture for extensible distributed systems. In *14th ACM symposium on Operating systems principles*, pages 58–68. ACM Press, 1993. Asheville, North Carolina, United States.

[25] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Schenker. A Scalable Content-addressable Network. In *Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, pages 161–172, San Diego, CA, 2001.

[26] Robbert Van Renesse, Kenneth P. Birman, and Werner Vogels. Astrolabe: A robust and scalable technology for distributed system monitoring, management, and data mining. *ACM Trans. Comput. Syst.*, 21(2):164–206, 2003.

[27] A. Rowstron, A-M. Kermarrec, M. Castro, and P. Druschel. SCRIBE: The design of a large-scale event notification infrastructure. In *NGC2001*, London, 2001.

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

[29] B. Segall, D. Arnold, J. Boot, M. Henderson, and T. Phelps. Content based routing with Elvin. In *Australian UNIX and Open Systems User Group*, 2000.

[30] Kundan Singh and Henning Schulzrinne. Peer-to-Peer Internet Telephony using SIP. In *NOSSDAV*, Skamania, Washington, 2005.

[31] 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.

[32] L. G. Valiant. A theory of the learnable. *Commun. ACM*, 27(11):1134–1142, 1984.

[33] Xin Xu, Han-gen He, and Dewen Hu. Efficient Reinforcement Learning Using Recursive Least-Squares Methods. *Journal of Artificial Intelligence Research*, 16:259–292, 2002.

[34] Vinod Yegneswaran, Paul Barford, and Somesh Jha. Global Intrusion Detection in the DOMINO Overlay System. In *Proceedings of Network and Distributed Security Symposium (NDSS)*, 2004.

[35] Yuanyuan Zhao and Rob Strom. Exploiting event stream interpretation in publish-subscribe systems, 2001.

[36] Shelly Q. Zhuang, Ben Y. Zhao, Anthony D. Joseph, Randy H. Katz, and John 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.