

Predictive Filtering: A Learning-Based Approach to Data Stream Filtering

Vibhore Kumar, Brian F Cooper, Shamkant B Navathe

College of Computing, Georgia Institute of Technology
801 Atlantic Drive, Atlanta, GA 30332-0280
{vibhore, cooperb, sham}@cc.gatech.edu

Abstract

Recent years have witnessed an increasing interest in filtering of distributed data streams, such as those produced by networked sensors. The focus is to conserve bandwidth and sensor battery power by limiting the number of updates sent from the source while maintaining an acceptable approximation of the value at the sink. We propose a novel technique called Predictive Filtering. We use matching predictors at the source and the sink simultaneously to predict the next update. The update is streamed only when the difference between the actual and the predicted value at the source increases beyond a threshold. Different predictors can be plugged into our framework, and we present a comparison of the effectiveness of various predictors. Through experiments performed on a bee-motion tracking log we demonstrate the effectiveness of our algorithm in limiting the number of updates while maintaining a good approximation of the streamed data at the sink.

1. Introduction

Advances in networking and sensor technology have made it possible to access sensor data as it is gathered, and this in turn has fueled the development of applications that use a continuous stream of data. To manage this data, several data stream management systems have been developed, including STREAM [4], NiagaraCQ [6], TelegraphCQ [7] and Aurora [5]. We consider distributed environments in which remote data sources continuously stream updates to a stream processing installation. These environments incur a significant communication overhead

in the presence of rapid update streams. Limiting the number of updates can significantly reduce this overhead.

We present a novel approach to limiting stream updates, called Predictive Filtering. In many scenarios like motion tracking and network monitoring, approximate data values can be tolerated by the stream applications [2]. When the data values do not change randomly, prediction algorithms can be used to approximate the next update when it occurs without actually streaming the data. In our approach the sink requesting data from a stream source specifies to the source a certain precision constraint that needs to be satisfied. Predictors are then deployed both at the source and the sink that adapt to evolving data patterns in the stream. An update is streamed only when the difference between the actual and the predicted value at the source increases beyond the threshold or the precision constraint; otherwise the sink uses the predicted update. In the case of streams with a known update rate, the sink knows when the next update should be predicted; otherwise an update-beacon (described later) is used to signal the occurrence of a new update at the source.

As with all previous prediction techniques our approach is also limited to data streams that show some pattern in the updates because predictions are based on the previous updates and patterns. Our approach is well suited for streaming sensor data, since many sensors track phenomena with an inherent pattern, such as temperature, motion, and so on. We next give an example of one particular application of our approach.

1.1 Example Application: Location Tracking and Collision Prevention

Consider a scenario where a number of fast moving objects are being tracked to maintain location information. Tracking the location of fast moving objects incurs a large amount of communication overhead because of the large number of updates required per unit time to track an object. Our predictive filtering approach takes advantage of the fact that motion does not tend to be random. For example, airplanes follow air routes and bees

Copyright 2004, held by the author(s)

Proceedings of the First Workshop on Data Management for Sensor Networks (DMSN 2004), Toronto, Canada, August 30th, 2004.

<http://db.cs.pitt.edu/dmsn04/>

move in a way that communicates information to other bees. As long as the objects being tracked stay on the predicted course, few updates will have to be streamed. Moreover, our filters can allow us to look beyond the current update and predict with some probability the future positions of the tracked objects. This can aid in applications that may need to react early to certain conditions, such as two airplanes passing too close to each other.

1.2 Related Work

An approach to data stream filtering was suggested by Olston, Jiang and Widom [1], which makes use of adaptive filters for processing continuous queries with precision guarantees. However, the approach makes no attempt to predict the next update and is thus similar to our approach with the predictor always predicting the next update to be equal to the last update. We present results comparing our techniques to their approach in Section 4. Quantitative guarantees regarding the precision of approximate answers is dealt with in [3]. One possible application for monitoring of environmental conditions using wireless sensors is discussed in [8, 9, 10].

1.3 Roadmap of the paper

The rest of this paper is divided into 4 sections; Section 2 gives a brief description of various prediction techniques explored in this paper. Section 3 contains the details to incorporate the prediction techniques into our stream framework. Section 4 contains an evaluation of our approach and its performance when compared to existing data stream filtering algorithms. In Section 5 we conclude by discussing some possible future directions and challenges in the domain.

2. Overview of Algorithm & Prediction Techniques

The choice of a prediction technique is highly dependent on the nature of the data stream under consideration. Using linear extrapolation one can easily approximate a

linear data stream that monotonically increases, decreases or remains constant for sufficiently large intervals of time; such a scenario happens when tracking fast moving objects that tend to stay on course. In situations where linear extrapolation is not appropriate because of the rapidly fluctuating or more complex patterns of behavior, enhanced prediction techniques like double-exponential smoothing can be used. In some cases like streaming stock market data, statistically modeling a system for predicting such updates might not be possible. In such scenarios neural network-based time series prediction methods can serve the purpose. However, besides the nature of data stream, the update rate may also affect the choice of the prediction technique. We can tolerate computational delays for streams with slow update rates; but for streams with faster update rates, techniques with less computational overhead have to be used to ensure delivery of all updates. We now briefly describe each of these approaches.

The basic components of our approach are shown in Figure 1. We maintain two predictors; one at the source, and other at the sink, that are exact copies of each other. The predictors contain three components; the ‘Predict’ component that is responsible for predicting the update based on past updates; ‘Learn’, which performs the learning in case of an incorrectly predicted update, and finally the ‘Update Trigger’ that causes periodic generation of an update in case of regular streams or causes the generation of an update on arrival of an update-beacon. The update-beacon is a small message that occurs in lieu of the actual update to signal the sink that an update has occurred. Update-beacons are important for streams with irregular update rates in which the occurrence of next update cannot be determined until it actually occurs. An update-beacon is not required for regular streams, but for irregular streams we must tolerate some communication overhead imposed by update-beacons (which is less than actually propagating the updates) in order to know when updates should be predicted. It may be possible to extend our predictors to predict the update rate as well, although we have not yet

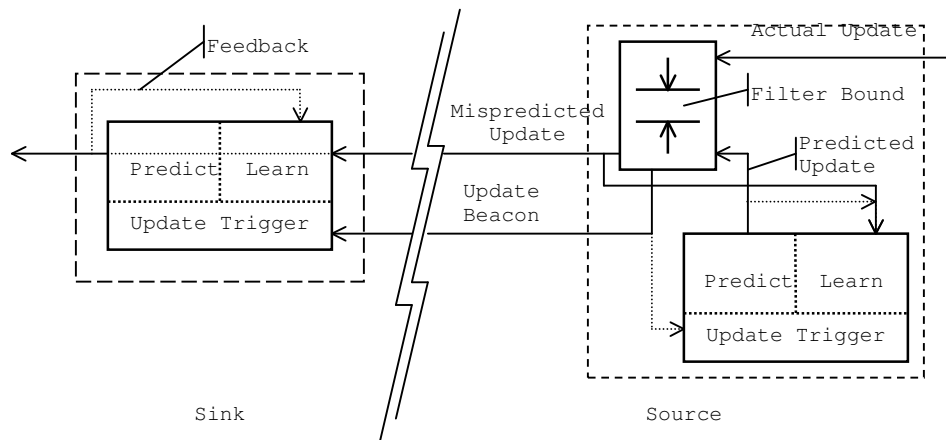


Figure 1. Showing the components in Predictive Filtering

```

/* source side component is invoked for each update */

THRESHOLD threshold;      // user defined
BOOLEAN regular_stream;   // true if stream is periodic

source_update(UPDATE actual_update)
{
    UPDATE predicted_update;
    predicted_update = predict();
    if(difference(actual_update, predicted_update)>threshold){
        stream(actual_update);
        learn(actual_update);
    }
    else{
        if(!regular_stream){
            signal_update_beacon();
        }
    }
}

```

Figure 2. Source side update component

```

/* sink side component is invoked by trigger update with NULL *
 * (either a timer or update-beacon) or by an arriving update */

sink_update(UPDATE actual_update)
{
    If(actual_update == NULL){ // no update arrived
        actual_update = predict(); // need to predict the update
    }
    else{
        learn(actual_update);
    }
}

```

Figure 3. Sink side update component

examined this possibility. The filter-bound at the source is a user specified parameter that encapsulates the degree of approximation that is tolerable.

The procedures at the source and the sink for handling updates / update-beacons are shown in Figure 2 and Figure 3 respectively. The `learn()` and `predict()` procedures are specific to the type of prediction technique being used. The following sub-sections contain details about these procedures.

2.1 Linear Extrapolation

Linear Extrapolation provides a technique that imposes a very low overhead but performs sufficiently well for predicting a wide range of data streams. Given two updates at time t_n and t_{n+1} the update at time t_{n+2} is given by the following expression:

$$u(t_{n+2}) = \frac{u(t_{n+1}) - u(t_n)}{t_{n+1} - t_n} t_{n+2} + \left\{ u(t_n) - \frac{u(t_{n+1}) - u(t_n)}{t_{n+1} - t_n} t_n \right\}$$

Thus, the next update is predicted to be on a straight line connecting the previous two updates. When the stream is

not changing rapidly we can predict more than the next update i.e. the d^{th} update into the future is calculated using the following expression:

$$u(t_{n+d}) = m \times t_{n+d} + c$$

$$\text{where } m = \frac{u(t_{n+1}) - u(t_n)}{t_{n+1} - t_n} \text{ and } c = \{u(t_n) - m \times t_n\}$$

For linear extrapolation, learning consists only of tracking the previous two updates.

2.2 Double Exponential Smoothing

Double exponential smoothing-based prediction (DESP) [11] models a given time series using a simple linear regression equation where the y-intercept c and slope m are varying slowly over time. An unequal weighting is placed on these parameters that decays exponentially through time so newer observations get a higher weighting than older ones. The degree of exponential decay is determined by the parameter $\alpha \in [0:1]$. The method makes use of two smoothing statistics:

$$S(u(t_n)) = \alpha u(t_n) + (1 - \alpha)S(u(t_{n-1}))$$

$$S'(u(t_n)) = \alpha S(u(t_n)) + (1 - \alpha)S'(u(t_{n-1}))$$

Using these smoothing statistics c and m can be estimated as c' and m' by applying the following equations

$$m'(t_n) = \frac{\alpha}{1 - \alpha} (S(u(t_n)) - S'(u(t_n)))$$

$$c'(t_n) = 2S(u(t_n)) - S'(u(t_n)) - t_n m'(t_n)$$

Given these estimates and with some algebraic manipulation the next update is predicted time d into the future with

$$u(t_{n+d}) = \left(2 + \frac{\alpha d}{1 - \alpha}\right) S(u(t_n)) - \left(1 + \frac{\alpha d}{1 - \alpha}\right) S'(u(t_n))$$

In this case, the predictor learns by refining c and m over time based on updates.

2.3 Artificial Neural Network based Predictors

Another popular technique used for prediction is Neural networks [12, 13, 14]. Compared to the previous two techniques, they tend to be more adaptable and flexible, since they can effectively model complex non-linear mappings and a broad class of problems due to their non-parametric nature. Their topology and weights are adaptable; therefore they are able to learn, which makes neural networks well suited for applications like prediction, system identification, and classification in many problem domains. One of the drawbacks of neural networks is that they need a sufficiently large data set to

```

/* source side component is invoked for each update */

THRESHOLD threshold;    // user defined
BOOLEAN regular_stream; // true if stream is periodic

source_update(UPDATE actual_update)
{
    UPDATE predicted_update;
    predicted_update = predict();
    if(difference(actual_update, predicted_update)>threshold){
        stream(actual_update);
    }
    else{
        if(!regular_stream){
            signal_update_beacon();
        }
    }
}

```

Figure 4. Modified source side update component

```

/* sink side component is invoked by trigger update with NULL *
 * (e.g. a timer or update-beacon) or by an arriving update */

sink_update(UPDATE actual_update)
{
    If(actual_update == NULL){ // no update arrived
        actual_update = predict(); // need to predict the update
    }
    else{
        learn(actual_update);
        update_source_predictor();
    }
}

```

Figure 5. Modified sink side update component

train the network, but this is what makes them even more suitable for stream-based applications that present enormous amounts of data. Another advantage of using neural networks is that they can be used to predict categorical data streams, which may have certain elements that cannot be mathematically approximated. However, the inherent mathematical complexity involved in training neural networks and then in the prediction may limit the usefulness of this approach.

Since we do not want to overload the stream data-source with computations for predicting the next update, we use a modified algorithm in this scenario. The learning component in case of neural network based predictors is present only at the sink and the predictor at the source is periodically updated to ensure similar predictions as the sink. The modified source and sink procedures are shown in Figure 4 and Figure 5 respectively. Further modifications to the algorithm include updating the sink predictor with batched updates after a threshold number of mispredictions. This allows us to limit the number of

```

/* E-Code Equivalent of source side linear predictor */

{
    int predicted_val;
    predicted_val = filter_data.m * filter_data.x + filter_data.c;
    if((predicted_val - input.val)>=filter_data.threshold ||
        (input.val - predicted_val)>=filter_data.threshold)){
        //update the filter_data
        filter_data.m = (filter_data.last_val - input.val);
        filter_data.c = input.val - filter_data.m * filter_data.x;
        filter_data.x = filter_data.x + 1;
        filter_data.last_val = input.val;
        return 1;
    }
    filter_data.x = filter_data.x + 1;
    filter_data.last_val = predicted_val;
    return 0;
}

```

Figure 6. E-Code representation of source side linear predictor

source predictor updates and produces better on-line training of the neural network. We have so far experimented with simple three-layer feed-forward neural networks and made use of the error back-propagation method to train the network; units with sigmoid function were used to construct the network. We are gathering results for neural-network based predictors as part of our ongoing work.

3. Implementation Details

In this section we deal with the issues concerning the deployment of predictive filters. We have implemented a distributed stream management framework using the ECho publish-subscribe middleware [18, 19] developed at Georgia Tech. The predictive-filtering algorithm discussed in this paper was implemented as a part of our stream management framework. The ECho middleware supports channels that facilitate the flow of data between the source and the sink. One of the important features of the ECho-Channel framework is its ability to dynamically compile and deploy filters written in E-Code, a highly portable subset of C, at remote sites to process data at the source. We have used this ability of the framework to enable deployment of predictors at the source. More details about ECho and the E-Code Language can be found in [18].

The E-Code equivalent of a source-side predictor using linear extrapolation is shown in Figure 6. The variables *input* and *filter_data* are implicitly available to the function. The *input* variable contains the update. The *filter_data* variable contains configuration and state information, including the slope, y-intercept, last update and the update iterator. A return value of 1 causes the update to be transmitted to the sink while a 0 results in non-transmission of the update.

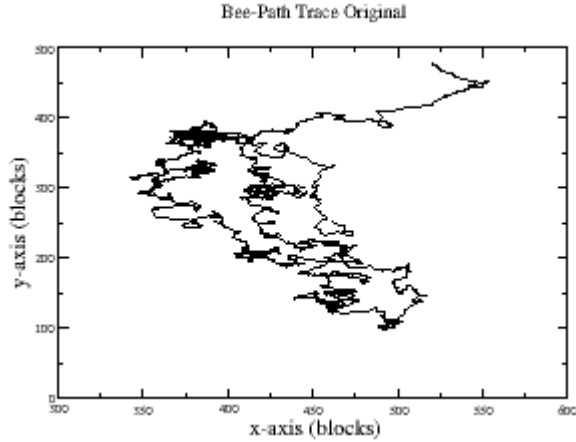


Figure 7. The original path followed by the Bee.

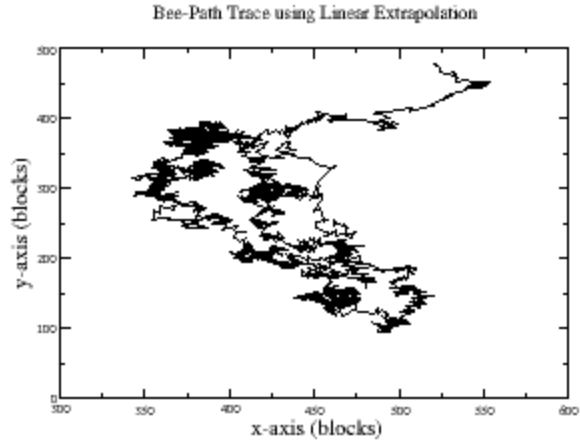


Figure 9. Bee-Path Trace using Linear Extrapolation

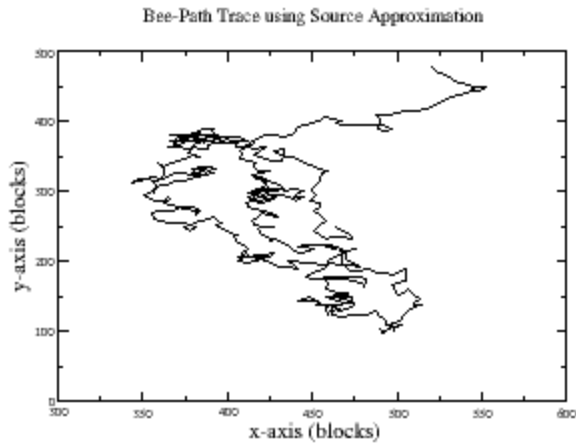


Figure 8. Bee-Path Trace using Source Approximation

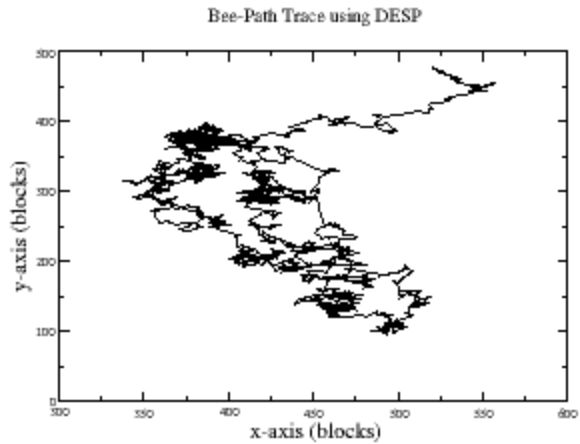


Figure 10. Bee-Path Trace using Double Exponential Smoothing

Table 1. Comparison of various predictive filtering techniques for constant Filter-Bound = 5

Filtering Technique	Number of Updates Propagated	RMS Error
No Filtering	7846	-
Source Approximation[1]	542	3.98
Linear Extrapolation	672	2.62
Double Exponential Smoothing	454	2.07

Another advantage of using ECho channels is that the sink can remotely access the *filter_data* configuration information available to the source-predictor. This facility helps in remote maintenance of this structure and proves helpful in implementation of neural network based predictors in which the source predictor needs to be updated by the sink.

4. Experimental Results

We evaluated the performance of our technique and its applicability by designing a motion tracking system that

records a log of various fast-moving objects. The temporal data feed was visual data collected by the BioTracking group at Georgia Tech [20]. The group video recorded activity of bees around a beehive for 10 days, 12 hours per day; and then processed the video using an image-processing algorithm to track the individual bees. We chose this data stream because the high rate of change in a bee's trajectory allows us to examine our techniques for predicting very complex data streams.

The original motion log of a single bee is shown in Figure 7, which depicts 7846 updates from the bee-path

Table 2. Comparison of various predictive filtering techniques for RMS-Error ~ 4

Filtering Technique	Number of Updates Propagated	Filter-Bound
No Filtering	7846	-
Source Approximation [1]	542	5.00
Linear Extrapolation	464	6.30
Double Exponential Smoothing	359	7.80

trace. In our experiment, we limited the number of updates for tracking the bee by using various prediction techniques discussed above. Figure 8 shows the path traced by the bee using the source approximation technique discussed in [1]. Note that the path is very sparse and misses some details. Figure 9 shows the results obtained by using a linear extrapolation predictor. The figure shows more of the zig-zag nature of the original curve as the filter tries to predict and approximate the correct position of the bee. The results of using a double exponential smoothing based predictor are shown in Figure 10 and smooth edges and even more detail are shown in the figure.

Table 1 shows the actual number of updates propagated from the source to approximate the position log for the various prediction techniques when filter-bound is kept constant. It also contains the root-mean-squared (RMS) difference (error) between the updates predicted by or received at the source and the actual updates. The RMS measures the quality of the approximated data; lower error is better. The number of updates required for linear extrapolation is more than that required for source approximation but the corresponding RMS error is considerably lower. However, the double exponential smoothing technique is the best both in terms of the number of propagated updates and the RMS error. Table 2 shows the actual number of updates required by each technique to approximately deliver the same quality in terms of RMS error at the sink. The table clearly shows the advantage of using prediction-based methods for filtering the updates. It may be noted that better filtering techniques allow for relaxed filter-bound to achieve the same quality of sink updates.

Experiments with neural-network based predictors are being conducted as part of our ongoing work.

5 Conclusion and Future Work

We have presented a novel approach to limiting the number of updates in streaming data environments by predicting values rather than streaming them. We have described the basic algorithm, which can be used in conjunction with a number of prediction techniques. Our initial experiments suggest that our approach can produce high quality data at the sink while effectively limiting the number of updates that must be sent. Our approach is

applicable to a large number of streaming data applications typically present with sensor networks that deal with regular data: e.g. network monitoring data, traffic data and stock market data, and so on. We are currently exploring the possibility of predicting categorical data streams using neural network based predictors. We understand that the techniques are limited to data with numeric values; if the data is from a domain with discrete strings or categories as values, a modification of the proposed techniques will be needed. This is left as part of our future work.

References

- [1] C Olston, J Jiang, J Widom. Adaptive Filters for Continuous Queries over Distributed Data Streams. In proceedings of the ACM SIGMOD International Conference on Management of Data, 2003.
- [2] R Min, M Bharadwaj, S Cho, A Sinha, E Shih, A Wang and A Chandrakasan. Low-power wireless sensor networks. In proceedings of the Fourteenth International Conference on VLSI Design, India, January 2001.
- [3] H Yu and A Vahdat. Efficient numerical error bounding for replicated network services. In proceedings of the Twenty Sixth International Conference on Very Large Databases, Cairo, Egypt, September 2000.
- [4] S Babu, J Widom (2001) Continuous Queries over Data Streams. SIGMOD Record 30(3):109-120
- [5] D Carney, U Cetintemel, M Cherniack, C Convey, S Lee, G Seidman, M Stonebraker, N Tatbul, S Zdonik. Monitoring Streams: A new class of data management applications. In proceedings of the twenty seventh International Conference on Very Large Databases, Hong Kong, August 2002.
- [6] J Chen, D DeWitt, F Tian, Y Wang. NiagaraCQ: A scalable continuous query system for internet databases. In proceedings of the ACM SIGMOD International Conference on Management of Data, Dallas, May 2000.
- [7] S Chandrasekaran, M Franklin. Streaming Queries over Streaming Data. In proceedings of the twenty seventh International Conference on Very Large Databases, Hong Kong, August 2002.
- [8] J M Kahn, R H Katz, K S J Pister. Next century challenges: Mobile Networking for "smart dust". In the proceedings of the ACM/IEEE International Conference

on Mobile Computing and Network Monitoring (MobiComm-99), Seattle, Washington, August 1999.

[9] S Madden, M J Franklin. Fjording the stream: An Architecture for queries over streaming sensor data. In the proceedings of the 18th International Conference on Data Engineering, San Jose, California, February 2002.

[10] G J Pottie, W J Kaiser. Wireless integrated network sensors. *Communications of the ACM*, 43(5):551-558, May 2000

[11] Joseph J LaViola Jr. Double exponential smoothing: an alternative to Kalman filter-based predictive tracking. Proceedings of the workshop on Virtual environments, Zurich, 2003.

[12] C. Lee Giles, Steve Lawrence, A. C. Tsoi. Noisy Time Series Prediction using a Recurrent Neural Network and Grammatical Inference. *Machine Learning Journal*, volume 44, 2001.

[13] Amalia Foka. Time Series Prediction Using Evolving Polynomial Neural Networks. MS Dissertation.

[14] S. Bengio, F. Fessant, and D. Collobert. A Connectionist System for Medium-Term Horizon Time

Series Prediction. In International Workshop on Applications of Neural Networks to Telecommunications, Stockholm, Sweden, 1995.

[15] G. Cybenko. Approximation by superposition of sigmoidal functions. *Mathematics of Control, Signal and Systems*, 2:303--314, 1989.

[16] F. Fessant, S. Bengio, and D. Collobert. On the Prediction of Solar Activity Using Different Neural Network Models. *Annales Geophysicae*, 1995.

[17] F. Fessant, S. Bengio, and D. Collobert. Use of Modular Architectures for Time Series Prediction. *Neural Processing Letters*, 1995.

[18] Greg Eisenhauer, Fabian Bustamente and Karsten Schwan. A Middleware Toolkit for Client-Initiated Service Specialization. Proceedings of the PODC Middleware Symposium - July 18-20, 2000.

[19] Greg Eisenhauer. The ECho Event Delivery System. Technical Report GIT-CC-99-08, College of Computing, Georgia Institute of Technology, Atlanta.

[20] <http://borg.cc.gatech.edu/biotracking/>