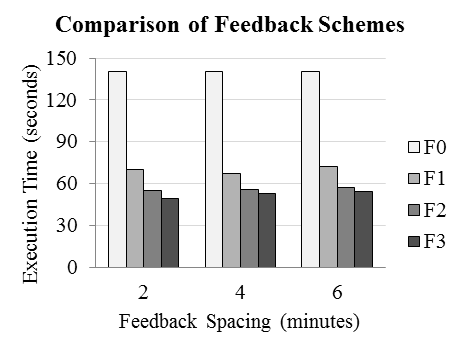
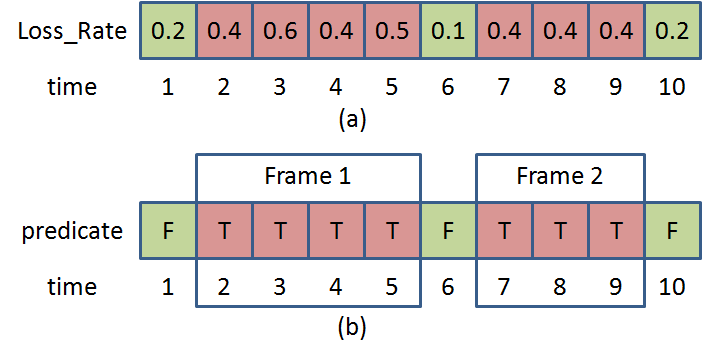
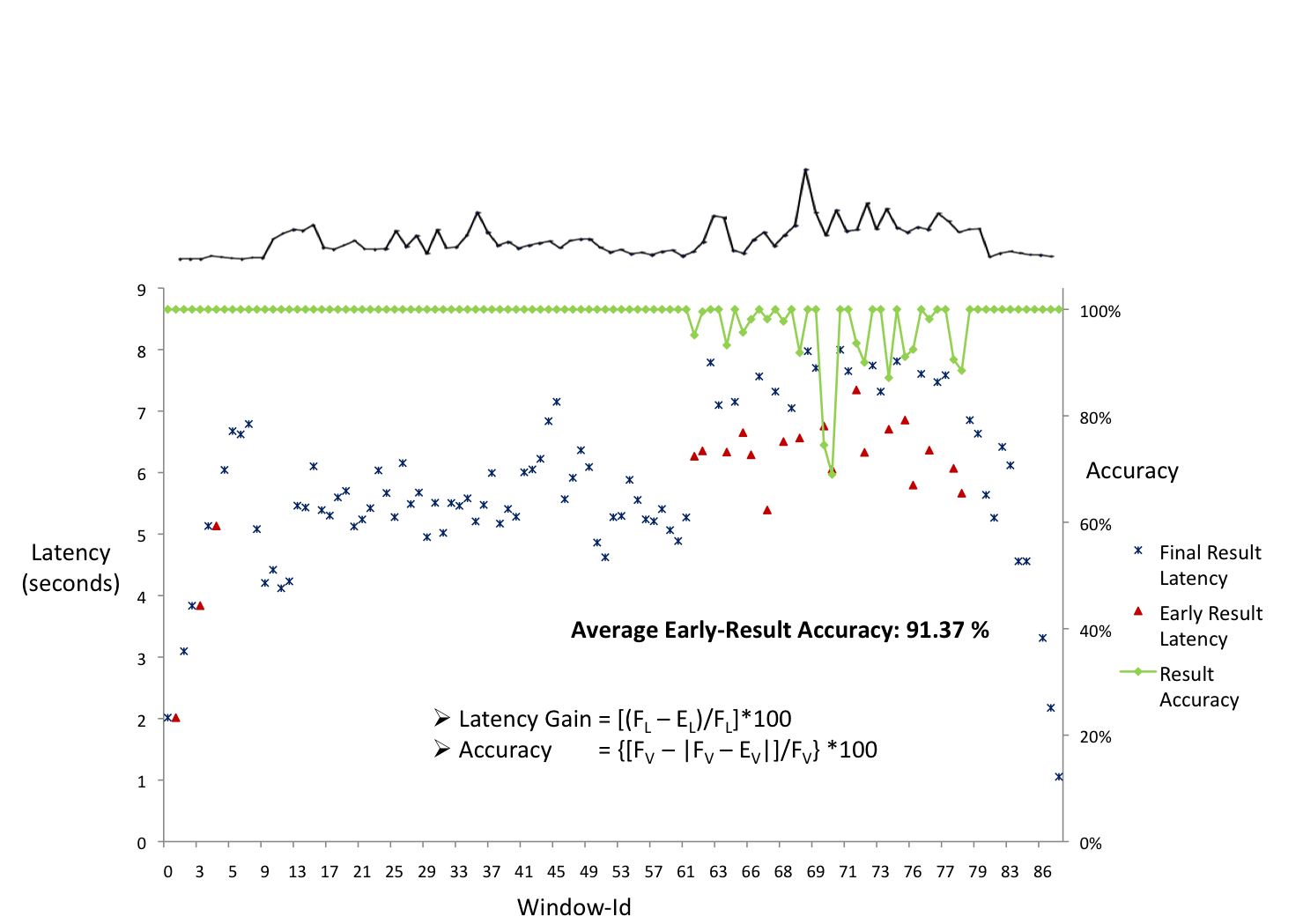
The team has developed theoretical models for contracts and assumed punctuations; techniques for ensuring the correctness and completeness of stream queries. A paper on this topic was written and presented at the International Workshop on Scalable Stream Processing Systems, 2010. This paper describes the formalization of punctuation contracts for stream operators, full-query analysis, and discusses future issues in compile-time query behavior characterization. In addition, performance tests have been completed to show the benefits of feedback punctuation. The figure below shows a comparison of query processing without feedback (F0) and various feedback schemes (F1, F2, F3) for various feedback spacings. One can see a significant improvement in execution time when feedback us used.



An implementation design and basic theoretical framework for Frames has been developed based on the observed need to identify time periods of interest in data streams. Frames are specified with a predicate and a duration and are implemented with an Apply operator, which processes the predicate, and a Frame operator which finds sequences satisfying the duration. The figure below displays frame detection. Part (a) shows data values which are converted into Boolean values, shown in part (b) by the Apply operator. Frame subsequently processes the Boolean values to detect sequences of True values satisfying the duration.



Masters student Amit Bhat completed an MS Thesis “Low-latency Estimates for Window-Aggregate Queries over Data Streams” as part of this project. He found the use of specialized stream elements called *prods* (a form of stream punctuation) can be effective in certain situations to counter latency increases due either to late tuples in a stream window, or a “density” burst in element arrival. (Density refers to the number of elements per unit of logical time. Density bursts often coincide with bursts in physical arrival rate of stream elements, but the two are not always correlated.) The figure below how prods help a windowed stream aggregate maintain relatively constant latency in result production, with marginal loss in accuracy. While not apparent in this figure, the production of early, approximate results introduced negligible delay in producing final, exact results. Bhat’s work also found that the tradeoff between latency and result accuracy using prods is dependent on data distributions and the particular aggregate being computed. For example, max and average tended to be more robust under aggressive prodding than sum and count.



*Year 2:*

*Theoretical Work on Frames:* Theoretical work on frames has focused on defining frame segmentation and maximal/minimal framing. Frames split an unbounded stream of tuples into variable length sub-streams for which a given predicate holds. The exact way of how frames segment the stream depends on the where the next frame starts in respect to where the previous frame ended. The figure below lists possible cases.



In case (a), the frames **partition** the stream. Every tuple of the stream belongs to **exactly one** frame and *endi-1 = starti*. Note that this segmentation of the stream is similar to tumbling windows. However, frames are still variable in length, whereas windows tend to be of fixed length.

In case (b), the frames **cover** the stream. Every tuple belongs to **at least one** frame and *endi-1 >= starti*. Note that this segmentation of the stream is similar to sliding windows. However, neither the slide offset nor the frame length is fixed.

In case (c), the frames produce **disjoint** stream segments. Every tuple belongs to **at most one** frame and *endi-1 <= starti*.

In case (d), the frames produce a **crazy** segmentation of the stream. Every tuple belongs to an arbitrary number of frames and there are no constraints over *endi-1* and *starti*.

The above argument (in the most cases) puts some constraints over the start and end points of frames. More precisely, it helps to define the earliest point where the next frame can start in respect where the previous frame ended (“Where do we start looking for the next frame?”). However, it does not contribute to controlling the length or slide offset of frames, in particular if the predicate is true for various overlapping segments of the stream. In this case, a frame specification should define exactly how frames are reported to the client. In order to approach this problem, let us consider the following cases.



Let us assume that t1 is the earliest possible start point for the frame that the system is currently computing. Further, t3 is the point where the predicate evaluates to true for the first time, yielding segment [t1, t3]. However, the predicate is also true for the segments [t2, t3] and [t2, t4]. In this setting there are four options how frames could be reported.

In case (a), the **maximal** frame [t1, t4] is reported. In case (b), the **minimal** frame [t2, t3] is reported. In case (c), the frame [t1, t3] is reported and in case (d), the frame [t2, t4] is reported. The former might make sense if new frames start where previous frames end and should be reported as soon as possible. The latter might make sense if there can be gaps between the frame and we want to report only if the predicates starts being false.

*Performance Work on Frames:*

We present a couple of results obtained from experimenting with frames over synthetic data. Using both frames and windows to summarize streams, the motivation of these experiments was to quantify the precision vs. overhead trade-off between the two approaches. Experiments were conducted over a synthetic data streams containing 1000 tuples with timestamps ranging from 0 to 999. The data streams were created using a data generator that used a random number generator and a trend variable to produce streams with different types of trends. We summarize the generated stream by aggregating values contained in both windows and frames by computing the average. As a condition for the frames, we use *max(value)-min(value) > k*. In addition to stream generation, *k* was varied in the experiments. Frames are minimal and adjacent. Windows have a fixed size and are tumbling.

Precision is measured by computing the absolute difference between the stream data value at timestamp *ts* and the summary value for the interval that contains *ts*, for all timestamps *ts* that occur in the generated stream. For each experiment, we plot a graph of the cumulative error up to timestamp *ts*. The lower the gradient of the curve, the better is the summarization of the stream. The plots also indicate how many frames and windows were used to summarize the stream. Note that this number only varies for frames in correspondence to variation of the parameter *k*.

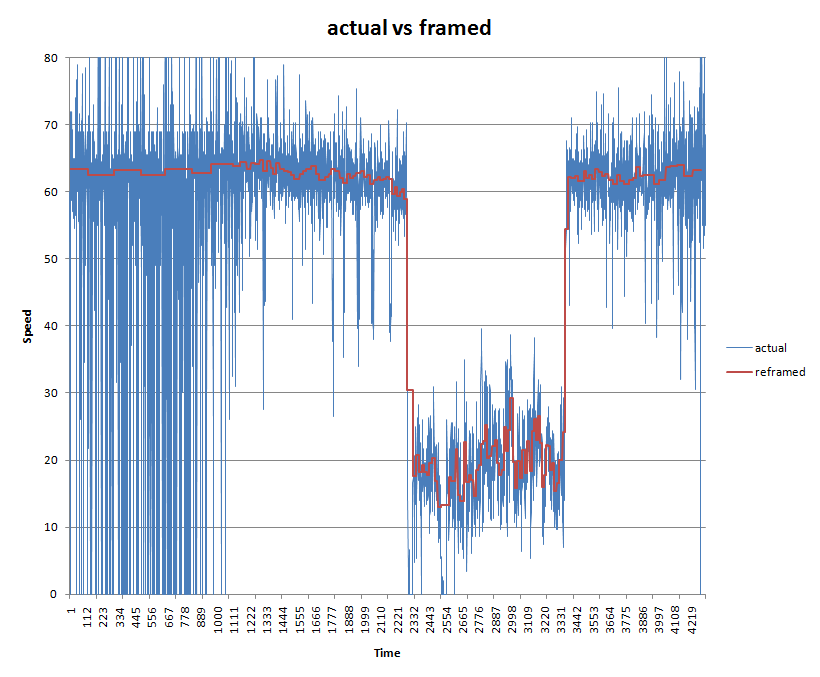
 

In the left plot, the frame parameter has been set to *k = 11.33*. This setting generates a total of 129 frames and yields a precision of 26.39%. The best window-based approach uses 200 windows and yields a comparable precision of 26.20%. In the right graph, the frame parameter has been set to *k = 7.75* which generates a total of 198 frames and yields a precision of 30.88%. Thus, using similar numbers of windows and frames, frames provide better precision.

The following graph plots the data values in the three streams introduced above and show their segmentation using windows and frames. For frames, we plot the setting of *k* that corresponds to the left plots above, i.e. the setting that yields comparable performance to the finest granularity of windows.



In addition to the experiments on synthetic data, we performed experiments using framing of traffic speed data. The figure below shows actual speed data in blue, versus the representation of the streams using frames, based on traffic volume, shown in red. One can see that frames provide a resonable representation of the data.



The next figure below shows the representation of the stream using frames, vs the representation using windows. One can see that the frame representation has less jitter, while still capturing the quick changes in reported speed. Windows, in contrast, provide a more noisy representation that uses more result values. We believe the frame representation is more useful to the end user and provides more information using less result space.

