SYNCHRONIZATION OF DATA STREAMS IN DISTRIBUTED REALTIME MULTIMODAL SIGNAL PROCESSING ENVIRONMENTS USING COMMODITY HARDWARE

L. Diduch, A. Fillinger, I. Hamchi, M. Hoarau, V. Stanford

{ldiduch, fantoine, ihamchi, mhoarau, vstanford}@nist.gov

ABSTRACT

We describe an API built on top of the NIST Data Flow System II, a sensor-net middleware, which allows to synchronize high volume data streams in real-time, using commodity hardware in distributed computation environments. Experiences with synchronization issues arising from working with multiple data streams in applications such as real-time multi-sensor data fusion and ad hoc video processing are addressed.

Index Terms— Multimedia Signal Processing, Data Transport Middle-ware, Distributed Sensor Networks, Multimodal Interfaces, Smart-spaces, Ambient Assisted Living

1. INTRODUCTION

High volume data generated in sensor networks, e.g. microphone array or multiple high-definition video feeds used in applications like industrial monitoring, biological or naval research, medical image processing, smart environments, surveillance applications, ambient assisted living environments or other areas requiring constant monitoring, becomes difficult to handle with rising network and processing loads. New technology standards like multi-core processors help to distribute the load of those applications. High bandwidth interfaces like firewire allow capture of data from a broad set of hardware. Using one machine and operating system however, these technologies only offer limited computational power and provide limited sets of data acquisition nodes. Distributed stream processing, which refers here to processing of streamed data on multiple computer systems applying a high bandwidth network, can help to distribute the processing load and extend the number of data acquisition nodes.

We describe applications of our sensor-net middleware, the NIST Data Flow System II (NDFS-II) [1] developed at the Smartspace Lab of the Information Access Division at the US National Institute of Standards and Technology (NIST). On top of this framework we recently developed an API assisting in real-time high volume signal processing, which will be main point of discussion. We describe our experience with high volume streaming data and synchronization of time-stamped datastreams using different data- and framerates. Two experimental results are presented, as well as two of our research applications to better explain the described features.

2. PLATFORMS FOR DISTRIBUTED SIGNAL PROCESSING

Many recently developed Distributed Stream Management Systems (DSMS) focus on continuous query database-like event stream processing. SQL like operations are used to process continuous sensor data streams. Platforms like TelegraphCQ [2], PIPES [3] and Borealis [4] fall into this category. While being very powerful in processing and recording high volume event streams (e.g. RFID data feeds from thousands of sensors), those are not designed to handle raw high-volume multi-modal data streams like a set of encoded HD video combined with multi channel audio feeds as used in multi sensor data-fusion applications. Platforms which focus on raw data transport are rare, and mostly custom coding solutions to one problem set instead of a generic framework. Most of those do not operate in distributed environments.

3. NIST DATA FLOW SYSTEM II

3.1. Overview

Development of the first version of the NDFS began in 1997 and was applied in distributed data acquisition tasks producing large multi-modal corpora (10 TByte) used in context sensitive environment research. International programs like AMI [5] or CHIL [6]) combining audio and video processing with focus on scene analysis use these corpora for evaluation purposes. Hundreds of sensors including multichannel microphone arrays, stand alone microphones and a set of HD video cameras, producing data at an aggregate rate of over three gigabytes per minute, form the Rich Transcription 2007 Meeting Recognition Evaluation (RT-07) [7] referred to above.

Requirements for operating system independence, better integration of platform specific sensors (OS specific drivers) and more dynamicity in ad hoc subscription to computing nodes drove a redesign of the NDFS culminating in the NDFS-II. *It is a decentralized, lightweight framework for distributed computing and data transport. The system developed in C++ also provides a Java API, runs on mainstream platforms like Windows, Mac OSX, Linux and Java VM and its source code is in the public domain. Strengths of the system are focus on raw data transport, generic application and*

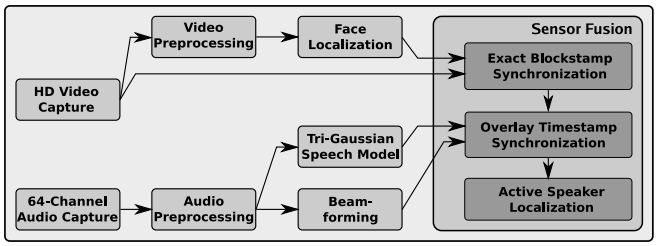

Fig. 1. A distributed multi-modal sensor fusion application estimating location of active speakers, based on block- and timestamp synchronization.

design simplicity. We use the NDFS-II not only for multi-modal data acquisition, but also for research on smartspaces and real-time multi-sensor data fusion and processing. Complex system research applications like distributed simulations of ant colony optimization algorithms (ACO) have been included recently into the spectrum of applications as well.

3.2. Advanced Features and Synchronization

Features besides raw data transport are implemented in *Flows* [8], e.g. on the fly video source coding using selectable codecs or endianess conversion. Advanced features can be implemented ontop of the NDFS-II API. This strategy gives users the possibility to refuse or accept new experimental extensions and allows our team to research more complex extensions, without modifying underlying system features. The framework comes with a set of Flows derived from our research needs. Flows are generally crafted by users to transport specific data acquired from their sensors.

Different kinds of sensors, however, like microphones and cameras, might produce varying sampling- and data-rates. Sensors capturing at the same frequency on different hosts might have varying data rates due to hardware setup differences, local clock offsets and drifts, uninterruptable kernel time, scheduler delays or unpredictable disk operations. Those stream imperfections occur using non real-time operating systems and commodity hardware. Of course, by using hardware solutions like *Genlock*, some of these sensors can be synchronized *a priori*. There are scenarios however, where new sensors do not have hardware synchronization solutions, or commodity hardware is used to capture and process data because of pecuniary boundaries. While we successfully synchronized sources of different type captured with commodity hardware *post hoc* [9], the need of *ad hoc* synchronization becomes more apparent in real-time multi-sensor data fusion and distributed signal processing.

To address this issue we developed multiple simple strategies of *ad hoc* software synchronization. Experiments are conducted on non real-time operating systems, using software generated block- and timestamps, referred to as *augmented* timestamps, using the local clock synchronized with a central NTP server. It is possible however, to use timestamps generated by hardware solutions in the synchronization algorithms, which will likely provide a better timing accuracy.

To illustrate the data flow in a distributed environment let us take a look at two fundamental queuing policies of the NDFS-II. Upon Flow initialization the user might set the *blocking* or *non blocking* data transport policy. Blocking policy simply blocks the application should the user want to retrieve data when none is in the queue. The non blocking policy never blocks the application and allows a Flow to drop data in case its queue is full, usually due to lacking processing power. While the blocking policy is useful in research scenarios where no data loss may to occur, non-blocking policies are used preferably in real-time scenarios with live signal capture or heavy processing tasks.

In a first scenario as presented in Fig. 1 we suppose no blockstamp and timestamp synchronization (see 'Sensor Fusion' box). Two main effects occur which determine basic rules of the processing pipeline: First, video and audio signal sources capture live with *different sampling and data rates*. Second each branch introduces its own *processing time-delay* due to varying processing time. Simply fusing those live multi-modal signals at the end of the pipeline, without dropping any data (by applying the *blocking* policy), will demand the user to synchronize the signals by hand and due to varying data rates knowledge about sampling rates of the sources has to be used. Even without overloaded processor nodes and neglecting branch specific processing delays *the fusion results in signals drifting away from each other* in this case.

In a second scenario we solve problems of different sampling rates and processor overload using the *non blocking* policy: The user now acquires data asynchronously. This still does not resolve processing time delay introduced by different processing branches, but allows a satisfactory way of fusing data together. Fused signals do not drift away from each other anymore, but a *shift* with a constant delay can be experienced. That is, audio signal can be heard before the corresponding video sequence because the video pipeline works slower, so streams are fused but still not synchronized.

Finally let us look at an application of the synchronization API developed on top of the described basic queuing policies: We augment a *combined* block- and timestamp (BTStamp) to the signal in the pipelines capture nodes (left in the figure). The BTStamp is generated by an incremental counter (blockstamp) and the local clock (timestamp), assuming the systems time has been synchronized with a global NTP server in advance. A timestamp allows to mix sensor signals of different data rates while a blockstamp allows data originating from one source, split across multiple computation nodes to be fused back in synchronicity (e.g. on multiple hosts or multi-core processors). Moreover the blockstamp is used to keep track of dropped frames. It might find future application in statistical ad hoc regression algorithms such as Widrow or Kalman filters estimating inter-frame timing. At data fu-

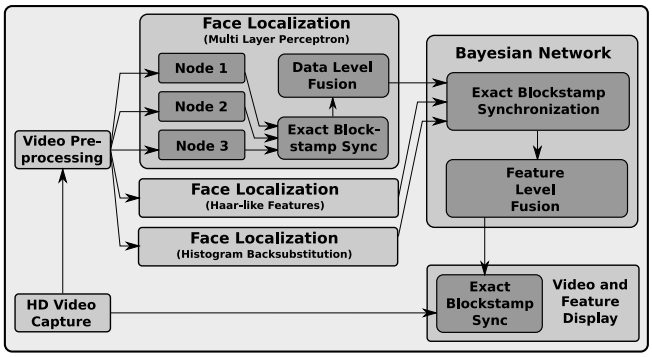

Fig. 2. A distributed, real-time, face localization task used in computer vision research. Only blockstamp based synchronization is applied since data is emerging from a single source.

sion side (right in the figure), we just feed the instance of our Synchronizer before the fusion step with incoming frames and request synchronized frames from it. The Synchronizer also provides useful information like detection of data loss or synchronization statistics, as well as strategies used to recover from data loss and additional optimization of memory consumption due to buffering. Mainly three synchronization strategies have been implemented, tested and applied so far:

Exact Blockstamp Match: applied in scenarios of single source dataflow split among multiple processing pipelines subject to stream and processing imperfections (e.g. processors of different speed). Used to detect dropped frames.

Tolerant Timestamp Match: simplest form of timestamp synchronization, similar to blockstamp match without detection of dropped frames. Ascending timestamps matching within a tolerance interval are assumed to be synchronized. This policy is only useful for sources with equal rates.

Overlay Timestamp Match: most applied form of timestamp synchronization. Timestamp *intervals* of several dataflows with different sampling rates are grouped in case of an interval overlay. The output rate of synchronized data might be higher than the rate of the fastest stream. Combined with blockstamp information about the sequence integrity of the stream this algorithm is a very powerful way to synchronize even streams with a large difference in frequency.

Applying the same example scenario but using the Overlay Timestamp Match algorithm (the timestamp and blockstamp synchronization of Fig. 1 boxes are used now) relieves the user of dealing with different data-rates. The shift due to processing delays is also eliminated because only frames which *overlap in time* are returned grouped. A last inevitable effect is a delay to all frames which occurs due to the largest processing time-delay because of the slowest pipeline branch.

For more details about the synchronization algorithms, application cases and limitations please refer to the API documentation.

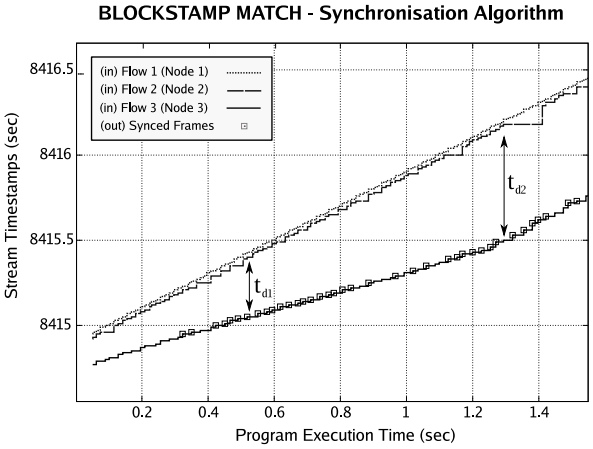

Fig. 3. Exact Blockstamp Match synchronization of three processing nodes with variable computation frequencies. Increasing processing time delay t_d for Node 3. Due to dropped frames, synchronization occurs rarely.

4. EXPERIMENTAL RESULTS

To demonstrate the difference between Exact Blockstamp Match and Overlay Timestamp Match algorithms we introduce a *test scenario* similar to the Multi Layer Perceptron (MLP) face localization box from Fig. 2. We split the computational needs by three to simulate distributed processing. A video signal originating from a single source is processed with different speeds by three computation nodes (Node 1, 2, 3) and collected in the 'Exact Blockstamp Sync' box. All Nodes operate in non-blocking mode and while Node 1 and 2 have similar processing speeds, *Node 3 computes slower* as can be seen in the update frequency of the lowest graph in Figures 3 and 4. The nodes input flows have a history buffer to compensate for stream imperfections. If this buffer overflows (e.g. because a node cannot consume its data in time) the node drops incoming data. This effect can be seen for Node 3 in a processing time delay drift ($t_{d1} < t_{d2}$) (buffering) between Flow 3 and the other two Flows before simulation time of 1.3 seconds. The Flows graphs become parallel after 1.3 seconds, indicating that Node 3 starts dropping data constantly, introducing a *constant processing time delay* t_d of approx. one and a half seconds. The Exact Blockstamp Match algorithm only groups frames with equal blockstamps and for those frames dropped (more or less randomly) by Node 3 no grouping of all three Flows can occur. As it can be seen in Fig. 3, the density of synchronized frames is small.

In case we replace the Exact Blockstamp Match with the Overlay Timestamp Match algorithm in the 'Blockstamp Sync' box inside the MLP box of Fig. 2, a better synchronization of all three sources can occur (high density of synced frames in Fig. 4). In this new case it does not matter if frames have been dropped or the Nodes computational frequencies differ because timestamp information is used.

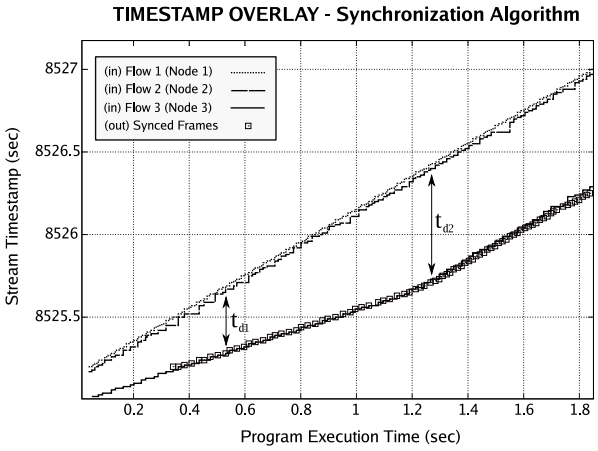

Fig. 4. Overlay Timestamp Match synchronization demonstrates better synchronization in a scenario of dropped frames and different computation frequencies.

5. APPLICATION EXAMPLES

The multi-modal sensor fusion shown in Fig. 1 is one of our test applications and performs a speaker localization based on sensor fusion of audio and video data. The setup consists of an HD-video and an audio feed generated from one of our NIST Mk-III 64-channel 44-kHz 24bit microphone arrays. The video branch of the pipeline performs image preprocessing for the second stage of face localization. The face localizer returns a feed with a region of interest (ROI) representing the coordinates of the localized face. The audio processing branch applies FIR filtering on 64-channels as a first step. The filtered data is fed into a tri-gaussian speech model processor for thresholding of active speech segments and into a beam-former to find the angle of a sound source. The sensor fusion box evaluates the correlation between the ROI and the angle based on a Overlay Timestamp synchronization due to the very varying sampling and computation rates.

Another application example, real-time processing with a high throughput (computer vision research), can be seen in Fig. 2. Using one video source, three *different* face localization algorithms are run in parallel in order to enhance the localization rate. The resulting ROIs are fed into a Bayesian network which uses *a priori* information about each localizers credibility. Sensor fusion occurs here in two flavors: The MLP part of the processing branch is split into three smaller computation nodes for faster processing. It uses blockstamp synchronization and data level fusion. In the Bayesian network box three different algorithms are compared using feature level fusion. Blockstamp based synchronization is used to avoid different frames being grouped in case a frame has been dropped.

6. CONCLUSION

By using a distributed processing system for real-time multisensor data-fusion applications, the ability to synchronize various data streams *ad hoc* has a crucial role. Our newly developed API built on top of the NDFS-II allows to synchronize such streams on commodity hardware using augmented time- and blockstamps. Synchronization of hardware generated timestamps is theoretically possible as well, however not yet verified by experiments.

Disclaimer: Specific commercial products, or open source software projects referred to by name are offered for the information of the reader. There is no endorsement by the US National Institute of Standards and Technology expressed or implied by such references.

7. REFERENCES

[1] V. Stanford, J. Garofolo, O. Galibert, M. Michel, and C. Laprun, "The nist smart space and meeting room projects: Signals, acquisition, annotation and metrics," *Proceedings of ICASSP*, 2003.

[2] Chandrasekaran, Sirish, O. Cooper, A. Deshpande, M. Franklin, J. Hellerstein, W. Hong, S. Krishnamurthy, S. ¡adden, V. Ramman, F. Reiss, and M. Shah, "Telegraphcq: Continous dataflow processing for an uncertain world," January 2005.

[3] C. Heinz, J. Kraemer, A. Markowetz, and B. Seeger, "Pipes: A multi-threaded publish-subscribe architecture for continuous queries over streaming data sources," July 2003.

[4] Abadi and J. Daniel et al., "The design of the borealis stream processing engine.," January 2005.

[5] T. Hain, L. Burget, M. Karafiat, J. Dines, D. van Leeuwen, G. Garau, M. Lincoln, and V. Wan, "The AMI Meeting Transcription System: Progress and performance," 2007, pp. 419–431.

[6] "Chil: Computers in the human interaction loop," http://chil.server.de.

[7] J. Fiscus, J. Ajot, M. Michel, and J. Garofolo, "The rich transcription 2007 meeting recognition evaluation.," 2007.

[8] V. Stanford, M. Michel, and O. Galibert, "Network transfer of control data: an application of the nist smart data flow," *J. of Systemics, Cybernetics and Informatics*, vol. 2, January 2005.

[9] M. Michel and V. Stanford, "Synchronizing multimodal data streams acquired using commodity hardware," VSSN, 2006.