FireStream: Sensor Stream Processing for Monitoring Fire Spread

Venkatesh Raghavan[1], Elke Rundensteiner[1], John Woycheese,[2] Abhishek Mukherji[1]
[1]Department of Computer Science, Worcester Polytechnic Institute, Worcester, MA 01609
[2]Department of Fire Protection Engineering, Worcester Polytechnic Institute, Worcester, MA 01609
{venky, rundenst, jpw, mukherab} @wpi.edu

Abstract

This demonstration presents FireStream, a sensor stream processing system which provides services for run-time detection, monitoring and visualization of fire spread in intelligent buildings that can be of great benefit to first responders. Our system can effectively handle large heterogeneous sensor streams using shared window execution and dynamic participant handling to yield a high-ary MJoin solution.

1. Introduction

Increasingly, modern buildings are equipped with building control systems and fire alarm control panels that monitor sensors for the safety of the occupants. The sheer number of sensors to be analyzed results in copious data streams that must be evaluated continuously to provide information such as fire location, growth, and spread.

In this demonstration, we present FireStream, a collaborative effort between Computer Science and Fire Protection Engineering. Our system employs a fire spread monitoring solution that integrates data-centric declarative queries with clustering alternatives. Our research focus is to detect fire events and track the spread of the fire over time and integrate heterogeneous sensor types for analysis. Sensors placed at key location are assumed to have minimum processing capabilities and the data streams generated by them are processed outside the sensor-network by our engine.

FireStream provides a rich set of queries of interest in the study of fire behavior. This, along with the established data library of over 200 actual fire experiments conducted at laboratories such as NIST/BFRL and the Department of Fire Protection Engineering at WPI, establishes the first benchmark for stream processing and monitoring of fires.

2. System Architecture

Our proposed framework, as illustrated in Figure 1, has, in essence, three core components: the *Execution Engine*, *Data Libraries* and the *Graphical User Interface*. Execution Engine makes use of *Phenomenon Matching Logic* and *Monitoring and Tracking Algorithms* to detect events and monitor the fire spread (Section 3). We maintain three data libraries: the *Spatial Store*, to record structural elements and sensor locations required for spatial analysis, the *Sensor Store*, a collection of metadata pertinent to sensors such as thresholds and calibrations, and the *Phenomenon Repository*, a set of patterns representing different classes of fire extracted from analysis of real fire dataset. The *Graphical User Interface* provides the users a tool to submit queries and visualize results.

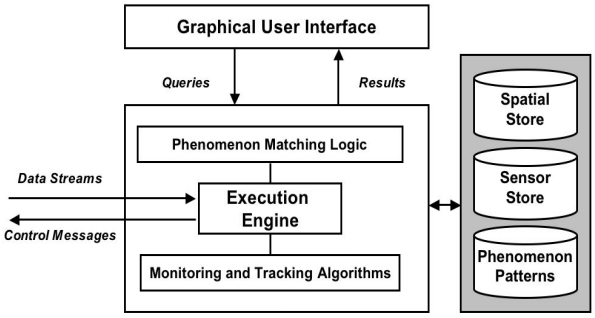

Figure 1. FireStream Framework Architecture

3. Salient Features

Location-Aware Stream Processing. Our system is supplemented with meta-knowledge about the placement of the sensors within the environment. This allows our higher-order query operators to obtain relative distances between sensors via spatial query support, such as nearest-neighbor and spatial-overlap queries. This information is vital to incorporate environmental information for direct fire paths that are influenced by factors such as corridors and walls.

Phenomenon Matching Logic. Phenomenon matching for tracking events or trends requires comparison of a multitude of data streams to identify patterns; detection of coupled, discrete events; or combinations of both. Various real test datasets [1] are analyzed to extract patterns and form

1-4244-0803-2/07/$20.00 ©2007 IEEE

a rich repository of different phenomena and their characteristics. The patterns are stored as a two-level inverted index representing the firetypes by a representative sequence of patterns, which are further divided into n-snippets. N-snippets are our numerical equivalent to n-grams for text-based pattern matching. An incremental strategy gathering statistics about sequence of matched patterns enables identification of phenomena. A feedback mechanism supplements the match operator to select future potential matches. Finally, the matching logic is scaled to multiple sensors to enhance trend tracking.

Dynamic Participants Handling. The sheer number of input streams makes tracking queries focus on scalability. Tracking queries involve cluster evaluations of moving events such as fire or smoke clusters. Stream participants that are involved in the query result change dynamically, as sensors are included/removed with the movement of the cluster. Recent research has focused on how to efficiently evaluate continuous queries with predetermined stream participants, for instance, multi-joins [5, 3]. *FireStream* reactively decides which input data streams must be involved in the join operation, based on membership criterion specified by the query (e.g., spatial proximity, sensor types, etc.), contrary to [2] which makes use of a probabilistic model that identifies sensors that record identical discrete phenomenons at a similar frequency to determine participant.

Interactive Query Plan Architecture. FireStream supports two means of communication between operators, namely using data streams and control streams. A data stream passes sensor data or partially computed results. It may be interleaved with punctuations [4] that define a data pattern for the particular slice of the stream. A control stream carries control messages between operators such as altering sampling frequencies, changing local parameters, and activating or de-activating operators. By default, every operator that is connected by a data stream is also connected by a bi-directional control stream. Operators also have the ability to broadcast control messages to all the operators involved in the query through the *Execution Engine*. This feature is critical in effectively tuning the sampling rate of the sensors and to control the number of participants of the join operation.

4. Demonstration Scenarios

FireStream makes use of the digital library called Experimental Data for Fire Science (EDaFS), developed at WPI [1], that maintains a rich collection of about 200 actual burn experiments conducted at BFRL/NIST, some of which are used to demonstrate our system. We use VRML (Virtual Reality Modeling Language) to generate the 3-D representations of the buildings and sensors (Figure 2).

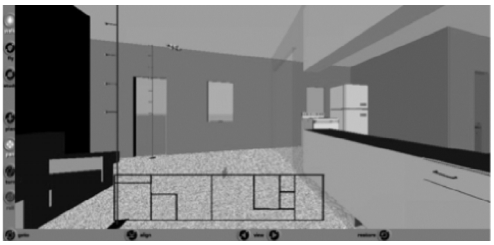

Figure 2. VRML Modeling of the Test Arena

Ambient Condition Assessment

- Visually represent the user queries as an algebra tree.
- Display results such as moving averages during stand-by mode, when there are no fire or smoke events.
- Report resources utilization parameters such as data arrival rates, CPU and memory consumption fluctuations.

Sensor Event Detection

- Detect and categorize fire events by matching windows of live sensor readings against patterns stored in the repository.
- Visualize in-alarm sensors and their spatially adjacent sensors to avoid false alarms and ascertain the accuracy of the match result.
- Represent the match result using graphical tools.

Fire Spread Monitoring

- Demonstrate fire scenarios such as arson, where several fires are ignited, or small and large scale fires.
- Visualize various clusters such as smoke and heat that aid in monitoring fire growth and spread.

References

[1] Experimental Database for Fire Science, available at: http://edafs2.wpi.edu:8050/edafs/.

[2] M. H. Ali, W. G. Aref, R. Bose, A. K. Elmagarmid, A. Helal, I. Kamel, and M. F. Mokbel. Nile-pdt: A phenomenon detection and tracking framework for data stream management systems. In *VLDB*, pages 1295–1298, 2005.

[3] E. A. Rundensteiner, L. Ding, T. Sutherland, Y. Zhu, B. Pielech, and N. Mehta. Cape: Continuous query engine with heterogeneous-grained adaptivity. In *VLDB Demo*, pages 1353–1356, 2004.

[4] P. A. Tucker, D. Maier, T. Sheard, and L. Fegaras. Exploiting punctuation semantics in continuous data streams. *TKDE*, 15(3):555–568, 2003.

[5] S. Wang, E. A. Rundensteiner, S. Ganguly, and S. Bhatnagar. State-slice: New paradigm of multi-query optimization of window-based stream queries. In *VLDB*, pages 619–630, 2006.