Target Container: A Target-Centric Parallel Programming Abstraction for Video-based Surveillance

Kirak Hong*, Stephen Smaldone†, Junsuk Shin*, David Lillethun*, Liviu Iftode†, and Umakishore Ramachandran*
*Georgia Institute of Technology
{khong9, jshin, davel, rama}@cc.gatech.edu
†Rutgers University
{smaldone, iftode}@cs.rutgers.edu

***Abstract*— We introduce a novel abstraction, the target container (TC), which serves as a parallel programming model and execution framework for developing complex applications for tracking multiple targets in a large-scale camera network. The key insight is to allow the domain expert (e.g., a vision researcher) to focus on the algorithmic details of target tracking and let the system deal with providing the computational resources (cameras, networking, and processing) to enable target tracking. Each TC has a one-to-one correspondence with a target, possibly tracked from multiple cameras. The domain expert provides the code modules for target tracking (such as detectors and trackers) as handlers to the TC system. The handlers are invoked dynamically by the TC system to discover new targets (detector) and to follow existing targets (tracker). The TC system also provides an interface for merging TCs whenever they are determined to be corresponding to the same target.**

This paper presents the design of the TC system, details of an experimental prototype, and an example application to demonstrate the simplicity of using the TC programming model.

I. Introduction

Sensors of various modalities and capabilities have been ubiquitously deployed throughout many urban environments. Fear of terrorism, among other criminal activities, have driven cities such as London and New York to employ a broad-based approach to surveillance using closed-circuit video cameras, among other sensor modalities, to monitor the majority of intra-city locations. The overall goal of surveillance is to detect suspicious activities and to track the individuals who perform them.

The conventional approach to surveillance has required direct human involvement, either at the time of video capture or to periodically review archived video recordings. Recent advances in computer vision techniques are now spawning a class of automated surveillance systems, requiring little to no human involvement to monitor targets of interest, detect threatening actions, and raise alerts. With the advanced vision techniques and large number of cameras, it should be possible to build a large-scale surveillance application that can detect and track threatening objects simultaneously.

The IBM Smart Surveillance project [1] represents the state-of-the-art in smart surveillance systems. Quite a bit of fundamental research in computer vision technologies form the cornerstone for IBMs smart surveillance solution. It is fair to say that IBM S3 transformed the video-based surveillance system from a pure data acquisition endeavor (i.e., recording the video streams on DVRs for post-mortem analysis) to an intelligent realtime online video analysis engine that converts raw data into actionable knowledge. Our work, which focuses on the programming model for large-scale situation awareness, is complementary to the state-of-the-art established by the IBM S3 research.

In the past decades, many programming models have been proposed to support different application domains. However, the existing programming models do not provide the right level of abstraction for developing complex streaming applications involving thousands of cameras. Thread based programming models require extra programming effort including communication and synchronization. Stream-oriented programming models do not support dynamic changes to stream graph, which makes it hard to use for surveillance applications that are highly dynamic. The problem we are trying to address in this paper can be stated as follows: *How can we facilitate the development of complex applications for video-based surveillance by domain experts that span 1000s of cameras and other sensors?*

In this paper, we introduce a novel abstraction, the *target container* (TC), which serves as a parallel programming model and execution framework for developing complex applications to track multiple targets in a large-scale camera network. The key insight is to allow the domain expert (e.g., a vision researcher) to focus on the algorithmic details of target tracking and let the system deal with providing the computational resources (cameras, networking, and processing) to enable target tracking. The TC corresponds to a target, possibly tracked from multiple cameras. The domain expert provides the code modules for target tracking (such as detectors and trackers) as handlers to the TC system. The handlers are invoked dynamically by the TC system to discover new targets (detector) and to follow existing targets (tracker).

This paper makes the following three contributions:

- It introduces the *target container* (TC) programming model, which simplifies the development of surveillance systems, freeing domain experts from the complex systems concerns of synchronization, scheduling, and resource management.
- It explains challenges in developing large-scale surveillance system using existing distributed/parallel programming models and the necessity for a new high-level parallel programming model.
- It presents the design of a TC system, details of an experimental prototype and an example application implemented based on the TC programming model.

Section II explains the limitations of existing programming models in developing large-scale surveillance applications. Section III describes the principles underlying the TC system. Section IV describes the TC prototype implementation and a target tracking application using the TC system. Finally, Section V positions TC within the broader context of related work in the area and Section VI concludes the paper.

II. Programming Large-scale Surveillance Applications

In this section, we motivate our approach by presenting limitations in developing surveillance applications based on existing programming models. We consider a couple of different approaches (namely, *thread based* and *stream oriented*) and identify the shortcomings of those approaches for large-scale surveillance applications before presenting our TC programming model.

A. Application Model

Let us first understand the application model of surveillance systems. In a surveillance application, there are two key functions: detection and tracking. Detection primarily focuses on finding an event that may be of interest to a survelliance application. For example, in an airport, there are control rooms that people are not allowed to access. If an unauthorized person is trying to access a control room, it should be captured by the automated surveillance system among thousands of normal activities in the airport. Once an event is detected, the automated surveillance system should keep track of the target that triggered the event. While tracking the target across multiple cameras, the surveillance system provides all relevant information of the target including location and multiple views captured by different cameras, helping a security team react to the threatening target. For clarity, we will use the term $detector$ and $tracker$ to indicate the independent instances of application logic.

The application model represents the inherent parallel/distributed nature of surveillance applications. Each detector is a per camera computation that exhibits a massive data-parallelism since there is no data dependency among the detectors working on different camera streams. Similarly, each tracker is a per target computation that can run simultaneously on each target. However, there exist complex data sharing and communication patterns among the different instances of detectors and trackers. For example, it is necessary to compare up-to-date target data generated by trackers and object detection results generated by a detector to find new targets within a camera stream. If a detected object has similar features (e.g., location, color, etc.) with an existing target, the two objects may be the same. Computer vision researchers have designed several spatio-temporal [2] and probablistic analysis [3] techniques to track an object across multiple cameras; essentially they compare different appearance models of targets to eliminate duplicates. Similarly, if a target simultaneously appears in the field of view (FOV) of multiple cameras, the trackers following the target on each of the different camera streams need to work together to build a composite model of the target. Building a composite model of the target is in the domain of computer vision and there is much research work in that space [1], [2], [3]. The focus of our research is in providing the right programming level hooks in a distributed setting for the domain expert to build such composite knowledge.

B. Existing Programming Models

Thread based Programming Model

The lowest-level approach to building surveillance systems is to have the application developer handle all aspects of the system, including traditional systems aspects, such as resource management, and more application-specific aspects, such as mapping targets to cameras. Under this model, a developer wishing to exploit the natural parallelism of the problem has to manage the concurrently executing threads over large number of computing nodes. This approach allows the developer to optimize the computational resources most effectively since he/she has complete control over the system support and the application logic.

However, carefully managing computational resources for multiple targets and cameras is a daunting responsibility for surveillance application programmer. For example, the shared data structure between detectors and trackers ensuring target uniqueness should be carefully synchronized to achieve the most efficient parallel implementation. Multiple trackers operating on different video streams may also need to share data structures when they are monitoring the same target. These complex patterns of data communication and synchronization place an unnecessary burden on an application developer, which is exacerbated by the need to scale the system to hundreds or even thousands of cameras and targets in a large-scale deployment (e.g., airports, cities).

Stream oriented Programming Model

Another approach is to use a stream-oriented programming model [4], [5], [6], [7] as a high-level abstraction for developing surveillance applications. Under this model, the programmer does not need to deal with low-level system issues such as communication and synchronization. Rather, she can focus on writing an application as a stream graph consisting of computation vertices and communication edges. Once a programmer provides necessary information including a stream graph, the underlying stream processing system

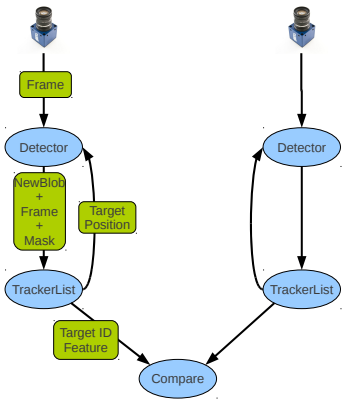

Fig. 1. Target Tracking based on Stream-oriented Models

manages the computational resources to execute the stream graph over multiple nodes. Various optimizations are applied at the system level, shielding the programmers from having to consider performance issues.

Figure 1 illustrates our attempt to implement a target tracking application using IBM System S [4], one of the representative off-the-shelf stream processing engines. In the application, a detector processes each frame from a camera, and produces a data item containing three different information: newly detected blobs, an original camera frame, and a foreground mask. A second stream stage, trackerlist, maintains a list of trackers following different targets within a camera stream. It internally creates a new tracker if newly detected blobs are received by a detector. Each tracker in a trackerlist uses an original camera frame and a foreground mask to update each target's blob position. The updated blob position will be sent to a detector, to prevent redundant detection of the target.

We implemented a lab-scale video surveillance system using IBM System S to see if the stream-oriented programming model is suitable for building large-scale surveillance systems. However, it has several critical drawbacks that makes it hard to use for large-scale surveillance application development. First, a complete stream graph should be provided by a programmer. In a large-scale scenario, providing a stream graph with huge number of stream stages and connections among them considering camera proximities is a very tedious task. Even if it is a static camera network where the number of cameras and their locations do not change, building such a stream graph for each application can be a non-trivial task, especially when it comes to thousands of cameras. Second, the stream-oriented approach is not well-suited for exploiting the inherent parallelism of target tracking. For example, when a new target is detected, a new instance of the tracker should be created to track the target. There is an opportunity to execute the new tracker concurrently if there is hardware parallelism available in the infrastructure. To exploit such target tracking parallelism, it is necessary to create a new stream stage to track the new target. However, dynamically creating a new stream stage is not supported by system S and therefore a single stream stage (the stage labeled trackerlist in Figure 1), should execute multiple trackers internally. This makes a significant load imbalance of different trackerlists, as well as low target tracking performance due to the sequential execution of trackers. Lastly, stream stages can only communicate through stream channel, which prohibits arbitrary real-time data sharing among different computation modules. As shown in Figure 1, a programmer has to explicitly connect stream stages through stream channels and deal with communication latency under conditions of infrastructure overload.

III. TC Programming Model

Based on the limitations of existing programming models described in the previous section, we present the design of our new programming model, Target Container. TC programming model is designed for domain experts who want to rapidly develop large-scale surveillance applications. In principle, the model generalizes to dealing with heterogeneous sensors (cameras, RFID readers, microphones, etc.). However, for the sake of clarity of the exposition, we adhere to cameras as the only sensors in this paper. We assume that the physical deployment topology of the camera network is known to the execution framework of the TC system. With the TC programming model, the connections between the cameras and the cluster nodes, as well as local communication among nearby smart cameras are orchestrated under the covers by the TC abstraction. This design decision has the downside that it precludes an application developer from directly configuring inter-camera communication. However, we believe that domain experts (e.g., vision researchers) would much rather delegate the orchestration of such communication chores to the system, especially when it comes to thousands of distributed cameras. Consequently, an application developer only needs to deal with the algorithmic aspect of target tracking using the TC paradigm. In the remainder of this section, we will further describe details of the TC programming model including application handlers and API provided by TC system.

A. TC Handlers and API

The intuition behind the TC programming model is quite simple and straightforward. Figure 2 shows the conceptual picture of how a surveillance application will be structured using the new programming model and Table I summarizes APIs provided by the TC system. The application is written as a collection of handlers. There is a *detector* handler associated with each camera stream. The role of the detector handler is to analyze each camera image it receives to detect any new target that is not already known to the surveillance system. The detector creates a *target container* for each new target it identifies in a camera frame by calling *TC_create_target* with initial tracker and TC data.

In the simple case, where a target is observed in only one camera, the target container contains a single *tracker* handler, which receives images from the camera and updates

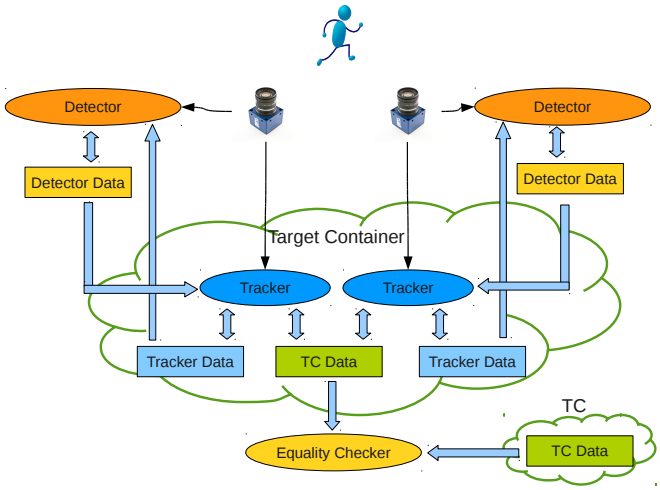

Fig. 2. Surveillance Application using TC Model

the target information on every frame arrival[1]. However, due to overlapping fields of view, a target may appear in multiple cameras. Thus, in the general case, a target container may contain multiple trackers following a target observed by multiple cameras. A tracker can call *TC_stop_track* to notify the TC system that this tracker need not be scheduled anymore; it would do that upon realizing that the target it is tracking is leaving the camera's field of view.

In addition to the detectors (one for each sensor stream), and the trackers (one per target per sensor stream associated with this target), the application must provide additional handlers to the TC system for the purposes of merging TCs as explained below. Upon detecting a new target in its field of view, a detector would create a new target container. However, it is possible that this is not a new target but simply an already identified target that happened to move into the field of view of this camera. To address this situation, the application would also provide a handler for *equality checking* of two targets. Upon establishing the equality of two targets[2], the associated containers will be merged to encompass the two trackers (see Target Container in Figure 2). The application would provide a *merger* handler to accomplish this merging of two targets by combining two application-specific target data structure (TC data) into one. Incidentally, the application may also choose to merge two distinct targets into a single one (for example, consider a potential threat situation when two cohorts join together and walk in unison in an airport).

As shown in Figure 2, there are three categories of data with different sharing properties and life cycles. Detector data is the result of processing the per-stream input that is associated with a detector. The data can be used to maintain detector context such as detection history and average motion level in the camera's field of view, which are potentially useful for surveillance applications using per camera information. The detector data is potentially shared by the detector and the trackers spawned thereof. The trackers spawned by the detector as a result of blob detection may need to inspect this detector data. The tracker data maintains the tracking context for each tracker. The detector may inspect this data to ensure target uniqueness. TC data represents a target. It is the composite of the tracking results of all the trackers within a single TC. The equality checking handler inspects the TC data to see if two TCs pertain to the same target, and if so calls the merger handler to merge the two TCs and create one composite TC data. Building such a composite data structure is in the purview of the domain expert.

The TC programming model allows dynamic data sharing between cameras and server nodes. This flexibility means that programmers do not have to statically set up the data communication among the camera nodes at application development time. Dynamically, the required communication topology among the camera nodes and the cluster nodes can be set up depending on the current needs of the application. Further, as described above, different handlers need access to the different categories of shared data at different points of time in their respective execution. Thus, providing access to shared data is a basic requirement handled in the TC system.

While all three categories of data are shared, the locality and degree of sharing for these three categories can be vastly different. For example, the tracker data is unique to a specific tracker and at most shared with the detector that spawned it. On the other hand, the TC data may be shared by multiple trackers potentially spanning multiple computational nodes if an object is in the FOV of several cameras. The detector data is also shared among all the trackers that are working off a specific stream and the detector associated with that stream. This is the reason our API (see Table I) includes six different access calls for these three categories of shared data.

When programming a target tracking application, the developer has to be aware of the fact that the handlers may be executed concurrently. Therefore, the handlers should be written as sequential codes with no side effects to shared data structures to avoid explicit application-level synchronization. TC programming model does not sandbox handlers to allow application developer to use optimized handlers written in low-level programming languages such as C and C++. Data sharing between different handlers are only allowed through TC API calls (shown in Table I), which subsume data access with synchronization guarantees.

B. TC Merge Model

To seamlessly merge two TCs into one while tracking the targets in real time, the TC system periodically calls equality checker on candidates for merge operation. To avoid side effects while merging, the TC system ensures that none of

[1]Since the physical deployment topology of the camera network is available to the execution framework of the TC system, the specific camera stream is implicitly delivered to the newly spawned tracker by the TC system.

[2]We use a manual approach to decide the set of targets that are potential candidates for equality checking based on the known physical topology of the camera network. However, it is important to exploit spatio-temporal knowledge about the targets for a scalable implementation of equality checking. This is part of our future work.

API	Description
TC_create_target()	It creates a TC and associates it with the new target. This is called by a detector. It also associates a tracker within this TC for this new target, which analyzes the same camera stream as the detector.
TC_stop_track()	When a target disappears from a camera's FOV, tracker makes this interface call to prevent further execution of itself.
TC_get_priority()	Get a priority of a TC.
TC_set_priority()	Set a priority of a TC.
TC_update_detector_data()	This will be used by detector for updates to per detector data structures.
TC_read_detector_data()	This will be used by detector/tracker for read access to per detector data structures.
TC_update_tracker_data()	This will be used by tracker for updates to per tracker data structures.
TC_read_tracker_data()	This will be used by detector/tracker for read access to per tracker data structures.
TC_update_TC_data()	This will be used by tracker for updates to per TC data structures.
TC_read_TC_data()	This will be used by detector/tracker for read access to per TC data structures.

TABLE I
TARGET CONTAINER API

the trackers in the two TCs are running. After merge, one of the two TCs is eliminated, while the other TC becomes the union of the two previous TCs.

Execution of the equality checker on different pairs of TCs can be done in parallel since it does not update any TC data. Similarly, merger operations can go on in parallel so long as the TCs involved in the parallel merges are all distinct.

TC system may use camera topology information for efficient merge operations. For example, if many targets are being tracked in a large scale camera network, only those targets in nearby cameras should be compared and merged to reduce the performance overhead of real-time surveillance application. Although discovery of camera connectivity in large scale is very important issue, it is outside the scope of this paper.

IV. IMPLEMENTATION

A. TC System

Our current implementation of TC system is in C++ and uses the Boost thread library and OpenMPI library. The TC runtime system implements each detector handler as a dedicated thread running on a cluster node. The TC system creates a pool of worker threads in each cluster node for scheduling the execution of the tracker handlers. The number of worker threads are carefully selected to maximize CPU utilization while minimizing context switching overhead.

The TC paradigm addresses parallelism at a different level in comparison to potentially parallel implementations of the OpenCV primitives. For example, OpenCV provides options to use the Intel TBB library and the CUDA programming primitives to exploit data-parallelism and speed up specific vision tasks. The TC programming model deals with parallelism at a coarser level, namely, multiple cameras and multiple targets. This is why TC uses OpenMPI, which supports multi-core/multi-node environments. Using the TC paradigm does not preclude the use of local optimizations for vision tasks *a la* OpenCV. It is perfectly reasonable for the application-specific handlers (trackers, detectors) to use such optimizations for their respective vision tasks.

In the prototype TC system, each cluster node has a TC scheduler that executes trackers running on a camera stream. The TC scheduler uses conventional round robin scheduling for ensuring fairness of TCs in each cluster node. TC data sharing between trackers is achieved via MPI communication. Handler migration across different computing nodes in the case of node failure or load imbalance is our future work, since the programming model gives a great degree of control to the underlying system.

Since the prototype TC system is designed for real-time tracking, it does not maintain a queue for past camera frames; it overwrites previous frame in a frame buffer if a new frame arrives. To avoid any side effects from overwriting frames, each tracker has its own duplicated frame for use by the tracking code. This ensures trackers always work with the most up-to-date camera frame. In this design, however, trackers can skip several frames if the system is overloaded. This is a trade-off between accuracy and latency: maintaining a frame queue will ensure that trackers process all the camera frames but such a design choice will introduce high latency for event detection under overloaded conditions.

B. Multi Camera Target Tracking Application

Using the TC Programming model, a surveillance application can be simply developed by writing four different handlers. In our prototype implementation, the detector handler uses blob entrance detection algorithm as shown in Figure 3. The detector discovers a new object within a camera's FOV by comparing the new_blob_list (obtained from the blob entrance detection algorithm) to the old_blob_list (from existing trackers running on the camera). The old_blob_list represents up-to-date position of existing targets within a single camera's FOV since each tracker is asynchronously updating its position. If the detector finds a new blob that does not overlap with any other existing targets, it creates a new TC by calling *TC_create_target* with initial data associated with the new tracker and TC.

Once created, a tracker tracks a target within a single camera's FOV using a color-based tracking algorithm as described in Figure 4[3]. In the tracker, *color_track* function computes the new position of the target using a blob-tracking algorithm from the OpenCV library. If the target is no longer in the image as determined by the *is_out_of_FOV* function (i.e., the target has left the FOV of the camera), the tracker requests

[3]The suggested implementation of the tracker and detector are for illustration purposes on the ease of development of a complex application using the TC model. As such, the choice of the algorithms for tracking, detection, equality checking, and merging is in the purview of the domain expert.

```
void Detector(CAM cam, IMAGE img)
{
  DetectorData dd = TC_read_detector_data(cam);
  list<Tracker> tracker_list = dd.tracker_list;
  TrackerData td;
  bool is_new;

  List<Blob> new_blob_list;
  List<Blob> old_blob_list;

  for_each(tracker in tracker_list)
  {
    td = TC_read_tracker_data(tracker);
    old_blob_list.add( td.blob );
  }

  new_blob_list = detect_blobs(img);

  for_each(nB in new_blob_list)
  {
    is_new = true;

    for_each(oB in old_blob_list
      If( blob_overlap(nB, oB) == TRUE)
        is_new = false;

    if(is_new == TRUE)
    {
      TrackerData new_td;
      new_td.blob = nB;
      TCData new_tcd;
      tcd.hist = calc_hist(img, nB);
      Tracker tracker = TC_create_target(tcd, td);
      dd.tracker_list.insert(tracker);
    }
  }
}
```

Fig. 3. Example Detector

```
void Tracker(Tracker tracker, TC tc, CAM cam, IMAGE
    img)
{
  TrackerData td = TC_read_tracker_data(tracker);
  TCData tcd = TC_read_tc_data(tc);
  Histogram hist;
  int threat_level;
  Blob new_blob;

  new_blob = color_track(img, td.blob);

  If( is_out_of_FOV( new_blob ) )
    TC_stop_track(tracker, cam);

  data.blob = new_blob;
  TC_update_tracker_data(tracker, data);

  threat_level = calc_threat_level(img, new_blob);
  TC_set_priority(tc, threat_level);

  hist = calc_hist(img, new_blob);

  if( compare_hist( hist, tcd.hist) < CHANGE_THRES )
  {
    tcd.hist = hist;
    TC_update_tc_data(tc, tcd);
  }
}
```

Fig. 4. Example Tracker

```
bool Equality_checker(TCData src1, TCData src2)
{
  if( compare_hist( src1.hist, src2.hist) >
      EQUAL_THRES )
    return TRUE;

  return FALSE;
}

void Merger(TCData src1, TCData src2, TCData dst)
{
  dst.hist = average_hist(tcd1.hist, tcd2.hist);
}
```

Fig. 5. Example Equality Checker and Merger

the system to stop scheduling itself by calling *TC_stop_track*. While tracking, application level target priority may change over time, depending on a target's behavior or location. Using *TC_get_priority* and *TC_set_priority*, an application can notify a target's priority to the TC system. This is vital for priority-aware resource management of the TC system. Tracker in this application also updates TC data if it finds color histogram of the target has been changed more than a threshold. Updating TC data has performance implications due to data sharing among multiple nodes, although the actual cost depends on the mapping of handlers to physical nodes. Because of this, updating TC data sparingly is a good idea.

Figure 5 illustrates examples of an equality checker and a merger. An equality checker compares two color histograms and returns TRUE if the similarity metric exceeds a certain threshold. Details such as setting the threshold value are in the purview of the domain expert and will depend on a number of environmental conditions (e.g., level of illumination). Such details are outside the scope of the TC programming model. The *compare_hist* function is implemented based on histogram comparing function from OpenCV. Merger simply averages two color histograms and assigns the result histogram to a newly merged TC data. The above examples are overly simplified illustration of the application logic for demonstrating the use of the TC system API. A sophisticated application may contain much more information than a blob position and a color histogram to represent the target data structure. For example, the target data structure may contain a set of color histograms and trajectories for different camera views of the same target. The equality checker and merger handler will be correspondingly more sophisticated, effectively comparing and merging two sets of color histograms and trajectories.

V. Related Work

TC shares with large-scale stream processing engines [4], [5] the concept of providing a high-level abstraction for large-scale stream analytics. However, TC is specifically designed for real-time surveillance applications with special support based on the notion of target.

The IBM Smart Surveillance project [8] is one of the few research projects in smart surveillance systems that turned into a product, which has been recently used to augment Chicagos video surveillance network [9]. IBM S3 product includes several novel technologies [2] including multi-scale

video acquisition and analysis, salient motion detection, 2-D multi-object tracking, 3-D stereo object tracking, video-tracking based object classification, object structure analysis, face categorization following face detection to prevent "tail-gating", etc. Backed by the IBM DB2 product, IBM S3 is a powerful engine in the hands of security personnel for online querying of live and historical data.

Other projects that are related to TC include ASAP [10]; the activity topology design based surveillance middleware [11]; service-oriented architectures for sensor networks such as OASIS [12]; and the high level abstractions for sensor network programming [13], [14]. ASAP [10] provides scalable resource management by using application-specific prioritization cues. Hengel et al. [11] approach scalability by partitioning the system according to an activity topology describing the observed (past) behavior of target objects in the network. OASIS provides a service-oriented programming framework for composing an application as a graph of modular services. EnviroSuite [13] and the work by Liu et al. [14] provide programming abstractions to shield the programmer from the chores of the distributed system (monitoring, leader selection, etc.) that are complementary to the concerns in the TC system.

Through extensive research in the last two decades, the state of the art in automated visual surveillance has advanced quite a bit for many tasks including: detecting humans in a given scene [15], [16]; tracking targets within a given scene from a single or multiple cameras [17], [18]; following targets in a wide field of view given overlapping sensors; classification of targets into people, vehicles, animals, etc.; collecting biometric information such as face [19] and gait signatures [20]; and understanding human motion and activities [21], [22].

The TC programming model with its focus on enabling the prototyping of large-scale video-based surveillance applications complements these advances. We have been working with computer vision researchers in identifying the right level of abstractions in defining our programming model and are planning to make our system more broadly available to the vision community.

VI. Conclusion

In this paper, we have argued that building effective automated, large-scale video surveillance systems requires a new approach, with a focus on programmability and scalability. While considerable progress has been made in the area of computer vision algorithms, such advances have not translated to deployment in the large. We believe that this is due to the lack of adequate system abstractions and resource management techniques to ensure their performance. We have proposed *target container* (TC) as such an abstraction, and have presented it as the core contribution of this paper. Along with the TC abstraction, we have presented the TC API for using the TC programming model, and an execution framework. The APIs are simple and expressive and greatly simplifies surveillance application development compared to the state-of-the-art.

The TC programming model provides the following key benefits: First, programmers need not deal with low-level thread management; nor do they have to provide a complete stream graph. Building a large-scale video surveillance application boils down to writing four computer vision algorithms presented to the TC system as handlers. Second, decoupling the programming model from its execution framework makes it easier to exploit domain-specific parallelism. Spawning the detectors and trackers on an actual execution platform consisting of smart cameras and cluster nodes is the responsibility of the TC system's execution framework. Third, the TC system subsumes the buffer management and synchronization issues associated with real-time data sharing of the different classes of shared data among the different instances of detectors and trackers. Finally, the TC system allows the application programmer to specify priorities for the targets that it is currently tracking. This information is available to the TC system to orchestrate the allocation of computational resources commensurate with the priority of the targets.

Our ongoing work includes development of a cluster-based execution framework for the TC system, and working with vision researchers to develop and deploy identity tracking applications using the TC programming model.

VII. Acknowledgement

We would like to acknowledge Bogdan Branzoi, who participated in this research early on and contributed to the development of the execution framework for the TC system on an SMP.

References

[1] R. Feris, A. Hampapur, Y. Zhai, R. Bobbitt, L. Brown, D. Vaquero, Y. Tian, H. Liu, and M.-T. Sun, "Case-study: IBM smart surveillance system," in *Intelligent Video Surveillance: Systems and Technologies*, Y. Ma and G. Qian, Eds. Taylor & Francis, CRC Press, 2009.

[2] A. Hampapur, L. Brown, J. Connell, A. Ekin, N. Haas, M. Lu, H. Merkl, and S. Pankanti, "Smart video surveillance: exploring the concept of multiscale spatiotemporal tracking," *Signal Processing Magazine, IEEE*, vol. 22, no. 2, pp. 38 – 51, march 2005.

[3] V. Menon, B. Jayaraman, and V. Govindaraju, "Multimodal identification and tracking in smart environments," *Personal Ubiquitous Comput.*, vol. 14, pp. 685–694, December 2010. [Online]. Available: http://dx.doi.org/10.1007/s00779-010-0288-6

[4] B. Gedik, H. Andrade, K.-L. Wu, P. S. Yu, and M. Doo, "Spade: the system s declarative stream processing engine," in *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, ser. SIGMOD '08. New York, NY, USA: ACM, 2008, pp. 1123–1134. [Online]. Available: http://doi.acm.org/10.1145/1376616.1376729

[5] L. Neumeyer, B. Robbins, A. Nair, and A. Kesari, "S4: Distributed stream computing platform," in *Proceedings of the IEEE International Conference on Data Mining Workshops*, 2010.

[6] W. Thies, M. Karczmarek, and S. P. Amarasinghe, "Streamit: A language for streaming applications," in *Proceedings of the 11th International Conference on Compiler Construction*, ser. CC '02. London, UK: Springer-Verlag, 2002, pp. 179–196. [Online]. Available: http://portal.acm.org/citation.cfm?id=647478.727935

[7] P. S. Pillai, L. B. Mummert, S. W. Schlosser, R. Sukthankar, and C. J. Helfrich, "Slipstream: scalable low-latency interactive perception on streaming data," in *Proceedings of the 18th international workshop on Network and operating systems support for digital audio and video*, ser. NOSSDAV '09. New York, NY, USA: ACM, 2009, pp. 43–48. [Online]. Available: http://doi.acm.org/10.1145/1542245.1542256

[8] IBM smart surveillance system (S3). [Online]. Available: http://www.research.ibm.com/peoplevision/

[9] ABC7 puts video analytics to the test. [Online]. Available: "http://abclocal.go.com/wls/story?section=news/special_segments&id=7294108"

[10] J. Shin, R. Kumar, D. Mohapatra, U. Ramachandran, and M. Ammar, "ASAP: A camera sensor network for situation awareness," in *OPODIS'07: Proceedings of 11th International Conference On Principles Of Distributed Systems*, 2007.

[11] A. van den Hengel, A. Dick, and R. Hill, "Activity topology estimation for large networks of cameras," in *AVSS '06: Proceedings of the IEEE International Conference on Video and Signal Based Surveillance*. Washington, DC, USA: IEEE Computer Society, 2006, p. 44.

[12] M. Kushwaha, I. Amundson, X. Koutsoukos, S. Neema, and J. Sztipanovits, "Oasis: A programming framework for service-oriented sensor networks," in *Communication Systems Software and Middleware, 2007. COMSWARE 2007. 2nd International Conference on*, jan. 2007, pp. 1 –8.

[13] L. Luo, T. F. Abdelzaher, T. He, and J. A. Stankovic, "Envirosuite: An environmentally immersive programming framework for sensor networks," *ACM Trans. Embed. Comput. Syst.*, vol. 5, no. 3, pp. 543–576, 2006.

[14] J. Liu, M. Chu, J. Liu, J. Reich, and F. Zhao, "State-centric programming for sensor-actuator network systems," *IEEE Pervasive Computing*, vol. 2, no. 4, pp. 50–62, 2003.

[15] C. Stauffer and W. Grimson, "Adaptive background mixture models for real-time tracking," *Computer Vision and Pattern Recognition, IEEE Computer Society Conference on*, vol. 2, p. 2246, 1999.

[16] A. Elgammal, D. Harwood, and L. Davis, "Non-parametric model for background subtraction," in *FRAME-RATE Workshop, IEEE*, 2000, pp. 751–767.

[17] I. Haritaoglu, D. Harwood, and L. Davis, "W4: Who? when? where? what? a real time system for detecting and tracking people," *Automatic Face and Gesture Recognition, IEEE International Conference on*, vol. 0, p. 222, 1998.

[18] C. Wren, A. Azarbayejani, T. Darrell, and A. Pentland, "Pfinder: Real-time tracking of the human body," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 19, pp. 780–785, 1997.

[19] W. Zhao, R. Chellappa, P. J. Phillips, and A. Rosenfeld, "Face recognition: A literature survey," *ACM Comput. Surv.*, vol. 35, no. 4, pp. 399–458, 2003.

[20] J. Little and J. E. Boyd, "Recognizing people by their gait: The shape of motion," *Videre*, vol. 1, pp. 1–32, 1996.

[21] D. M. Gavrila, "The visual analysis of human movement: a survey," *Comput. Vis. Image Underst.*, vol. 73, no. 1, pp. 82–98, 1999.

[22] A. F. Bobick and J. W. Davis, "The recognition of human movement using temporal templates," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 23, no. 3, pp. 257–267, 2001.