Scalable Spatio-Temporal Analysis on Distributed Camera Networks

Kirak Hong, Beate Ottenwälder, and Umakishore Ramachandran

Abstract Technological advances and the low cost of sensors enable the deployment of large-scale camera networks in airports and metropolises. A well-known technique, called spatio-temporal analysis, enables detecting anomalies such as an individual entering into a restricted area without permission. Spatio-temporal analysis requires a large amount of system resources to infer locations of occupants in real-time. In particular, state update becomes a bottleneck due to computation and communication overhead to update possibly large application state. In this paper we propose a system design and mechanisms for scalable spatio-temporal analysis. We present a distributed system architecture including smart cameras and distributed worker nodes in the cloud to enable real-time spatio-temporal analysis on large-scale camera networks. Furthermore we propose and implement a couple of selective update mechanisms to further improve scalability of our system by reducing the communication cost for state update.

1 Introduction

As sensors for recognizing humans, such as cameras and voice recognition sensors are becoming more capable and widely deployed, new application scenarios arise, requiring an automated processing of the continuous stream data to identify and track human beings in real-time. Airports, as an example, currently have more than 1,000 cameras in place and plan to increase

Kirak Hong and Umakishore Ramachandran
College of Computing, Georgia Institute of Technology, Atlanta, Georgia, USA e-mail: <khong9,rama>@cc.gatech.edu

Beate Ottenwälder
Institute of Parallel and Distributed Systems, University of Stuttgart, Stuttgart, Germany, e-mail: beate.ottenwaelder@ipvs.uni-stuttgart.de

Published in Zavoral, Filip and Jung, Jason J. and Badica, Costin Studies in Computational Intelligence, Volume 511, pp. 131-140, 2014.

that number [1]. Video streams can be automatically analyzed by *situation awareness applications* [6] to detect security violations, such as unauthorized personnel entering a restricted area.

Situation awareness applications often rely on a technique called *spatio-temporal analysis* to answer queries on occupants such as "When did person O leave zone Z?". Recently, Menon et al. [5] showed the feasibility of spatio-temporal analysis using a global *application state*, which comprises the information for each occupant that she is with a certain probability in a zone. Sensor-streams, stemming from cameras or other modalities (such as audio and biometrics), detect occupants in the observed system and the application state is updated regarding these observations.

However, keeping a global application state at a central server in large-scale camera networks imposes a significant performance bottleneck. Thousands of cameras constantly send updates to that server, drastically increasing the communication costs, and the server has to potentially perform a vast number of computations to process all the updates.

Our contribution in this paper are therefore, i) the distributed design of a system for spatio-temporal analysis, ii) identifying the performance bottleneck of spatio-temporal analysis on large-scale camera networks, and iii) scalable mechanisms to maintain a distributed application state.

The paper is structured as follows: Section 2 explains details of spatio-temporal analysis on camera networks and identifies the sources of performance bottlenecks in large scale camera networks. Section 3 describes the key problem in terms of system scalability. Section 4 provides our solution, namely, scalable state update to overcome the performance bottleneck. Section 5 presents our evaluation results. Section 6 presents other work related to our project. Section 7 presents concluding remarks and future work.

2 Spatio-temporal Analysis

In this section, we discuss the common steps in spatio-temporal analysis and propose a distributed system model for large-scale spatio-temporal analysis.

Spatio-temporal analysis is an inference technique that generates probabilistic locations of occupants at a given time using sensor streams. Figure 1 shows an example of spatio-temporal analysis. In the example, the face of a person (*signature*) is captured by a camera, from which an *event* is generated. The event is a vector of probabilities representing how closely this signature matches identities pre-registered in the system. The event causes a transition of an application's global state to represent up-to-date location of individual occupants. Before making a transition, the previous state is recorded with a timestamp to answering time-dependent questions in the future.

The probabilistic locations captured by a global state is a key to answering various queries referring to location-, occupancy-, and time-dependent

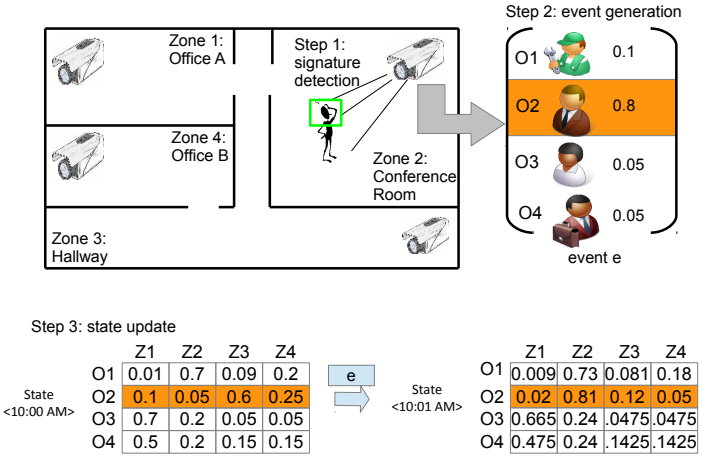


Fig. 1 Example of spatio-temporal analysis on camera networks

questions. Each row in the global state, called an *occupant state*, represents probabilistic location of a specific occupant that allows to answer a *location query* such as "where is occupant O1?". Each column is called a *zone state*, representing the probabilistic occupancy of a zone that allows to answer *occupancy query* such as "who are presently in zone z1?". A sequence of global states tagged by timestamp allows *temporal query* such as "When did person B leave zone X?". By combining the location, occupancy, and temporal queries, application can infer various situations:

"Where did person A and B meet for the last time?"
"How many times did person A move from zone X to zone Y?"
"How many people access zone X today?"

Maintaining the global state involves three steps of processing. The first step, signature detection, involves video analytics to detect signatures such as faces. For example, when a person enters Zone 2 in Figure 1, a face detection algorithm reports the person's face by analyzing video frames from a camera observing the zone. The second step, event generation, generates a vector of probabilistic estimates about the identity of the detected signature, called an *event*. Depending on the application, different algorithms can be used in this step to generate the vector. For example, various face recognition algorithms [10] or human gait recognition [8] may be used to generate an event, which includes similarities between the detected signature and known signatures. The final step, state update, uses the generated event to update the global state of an application. This modifies the location information of occupants represented in the event. For example, Figure 1 shows that an occupant $O2$ was in zone $Z2$ with probability 0.05 before the state update,

but the probability is increased to 0.8 after making a transition based on an event from $Z2$ with a high probability for $O2$ being in that zone. [1]

3 Problem Description

In large-scale spatio-temporal analysis, each processing step involves significant amount of computation and communication that require distributed computing resources to provide real-time situation awareness. The workload of signature detection and event generation are massively parallel, since each signature and the associated event can both be independently computed on distributed nodes including smart cameras and cloud computing resources. This makes the two steps linearly scalable with the amount of distributed computing resources. However, unlike the previous two steps, state update requires sequential processing of events due to the inherent nature of maintaining a single global application state. Due to the probabilistic nature of the global application state, an event update potentially affects the probabilities of every occupant in all the zones. Therefore, to allow the temporal evolution of the global application state, each event has to be applied sequentially in temporal order to the current global state. In a large-scale situation awareness application, a centralized approach that maintains the global state at a single node will be overloaded due to the computation and communication overhead of state update. If many events are generated at the same time, a single node cannot receive and process all events in real-time and therefore the latency for situation awareness will increase. In the following sections, we describe our scalable state update solution to solve this performance bottleneck.

4 Scalable State Update

In order to ensure the scalability of spatio-temporal analysis, both computation and communication overhead of state update should be addressed. To address computation overhead, we propose a distributed state update using partitioned application state across multiple state worker nodes where state update is performed on the partial application states at each node. Figure 2 shows our distributed state update using multiple occupant state workers (OW) and zone state workers (ZW). Each state worker maintains a set of occupant states and zone states to answer queries regarding specific occupants and zones. For example, occupant state worker OW1 maintains the occupant state of O1 to answer location queries on the occupant, while a zone state

[1] The concrete formulas to calculate the probability are presented in [5].

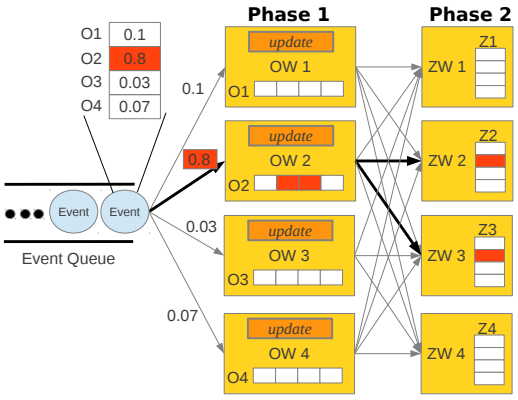


Fig. 2 Distributed state update: For each event, occupant states are updated in parallel at distributed occupant state workers (Phase 1). Once all occupant states are updated, occupant state workers transmit the elements of occupant states to zone state workers (Phase 2). Thin gray lines indicates the communication path for naive state update while thick black lines show communication path for selective update.

worker ZW1 maintains the zone state of Z1 to answer occupancy queries for the zone.

When a new event is generated, each probability for different occupants in the event are delivered to different occupant state workers commensurate with the occupant states that each is responsible for. Upon the arrival of each event element, state update is performed on each occupant state at different occupant state workers (phase 1). Once the occupant states are updated, each occupant state worker transmits elements of occupant states to zone state workers (phase 2). As shown in the figure, the real work of computing the new probabilities for each occupant in every zone is carried out by the occupant state workers in phase 1. Phase 2 is a simple data copy of the computed probabilities in phase 1 and involves no new computation. Since no computation happens at zone state workers, phase 2 may seem to be optional. However, the zone state workers are crucial to handle occupancy queries efficiently because a user has to broadcast a zone-related occupancy query to all occupant state workers if there are no zone state workers.

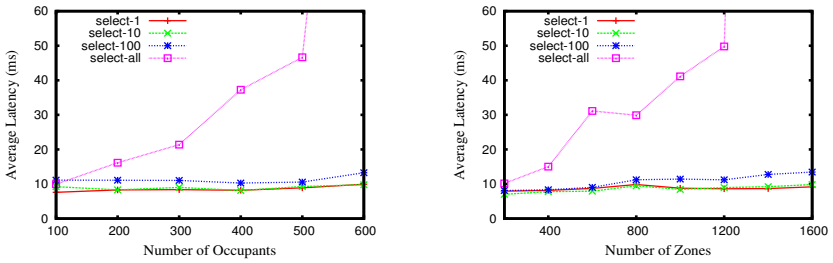
Although computation overhead is distributed over multiple nodes, communication overhead of state update is still a significant bottleneck. As Figure 2 indicates, each element of an event has to be transmitted to all occupant state workers, which increases the number of messages when more occupant state workers are used. Furthermore, each occupant state worker has to communicate to all zone state workers to update the zone states in phase 2. The total communication cost in terms of bytes transferred linearly depends on the number of occupants and zones in a system:

$$Cost_{comm} = (N_{occupants} + N_{occupants} \times N_{zones}) \times sizeof(double) \qquad (1)$$

For example, assuming thousand occupants and thousand zones, and 8 bytes double-precision floating-point representation for event probabilities, each event represents a communication payload of eight megabytes for state update. Assuming hundred signatures are captured from distributed cameras every second, state update would incur a communication cost of eight hundred megabytes per second.

To solve the communication overhead, we propose selective update mechanisms for state update. In a realistic application scenario, the event generation algorithm (e.g., face recognition) may generate an event that has only a few significant probabilities. If an event generation algorithm is highly accurate, i.e., giving high probability for the ground truth identity and very small probability for other identities, a threshold can be applied to use only meaningful event elements for updating specific occupant states. Similarly, another threshold can be applied to occupant states to allow occupant state workers to transmit only significant changes in occupant states to zone state workers. Highlighted elements and arrows in Figure 2 show an example of selective state update. In the example, our system selects only one event element for O2 with significant probability, which is used for state update. After updating an occupant state of O2, only two significant changes for zone Z2 and Z3 are selected and transmitted to corresponding zone state workers. As shown in the figure, the communication cost of state update depends on selected occupants in each event and selected zones from each occupant state rather than the total number of occupants and zones in the system.

To allow such selective state update, our system provides two parameters to users: *occupant selectivity* and *zone selectivity*. Occupant selectivity allows a user to specify the number of occupants to be selected from each event, while zone selectivity specifies the number of zones to be selected from each occupant state. When an event is generated, our system finds occupants with top N probabilities pertaining to the occupant selectivity. Similarly, when an occupant state is updated, our system calculates the difference between the current state and previous state to selects zones with top N changes. Our system supports automatic tuning of an occupant selectivity or a zone selectivity, which makes sure that the total communication cost is bounded by a user-provided threshold. To use the automatic tuning, a user specifies one selectivity (either occupant or zone selectivity) and the maximum communication cost. Using Equation 1, our system automatically infers the right value for an unspecified selectivity to make sure the total communication cost stays below the given maximum communication cost. While event rate changes over time, our system adaptively changes the unspecified selectivity to help system running in real-time in the presence of highly varying event rates from the real world.



(a) Different number of occupants **(b)** Different number of zones

Fig. 3 Average latency for state update with different number of occupants and zones. (a) 1000 zones are used where different number of occupants are selected (b) 425 occupants are used where different number of zones are selected

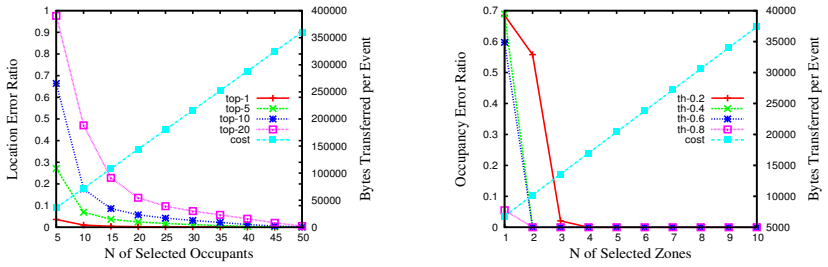
5 Evaluation

5.1 Scalability of State Update

To measure the performance based on a realistic workload, we performed the state update algorithm reported by Menon et al. [5] on eight distributed worker nodes in Amazon Elastic Compute Cloud (EC2) where each node is an *m1.medium* class.

Figure 3(a) shows average latency of state update per event with different selectivity of occupants, while varying the scale of the system in terms of the number of occupants in the system. For example, *select-1* indicates selecting only a single occupant from each event while probabilities for all other occupants are ignored. Similarly, *select-all* indicates that probabilities for all occupants are used for state update. As shown in the figure, the naive state update selecting all event elements (select-all) scales poorly, as the system is overloaded when there are more than 500 occupants in the system. Until 500 occupants, latency for state update increases depends on the total number of occupants in the system. Other selective mechanisms show good scalability, where the average latency depends on the number of selected occupants rather than the total number of occupants in the system.

Similarly, Figure 3(b) shows the poor scalability of naive state update and improved scalability and latencies with selective zones for state update. With more than 1200 zones, the naive state update becomes overloaded and cannot handle incoming events in real-time. Other selective state update mechanisms scale well, while the latency for state update depends on the number of zones selected from occupant states. Because we used virtual machines in EC2, available bandwidth and communication latency between distributed state workers vary over time, which results in slightly nonlinear latencies in figures.



(a) Selective Occupants **(b)** Selective Zones

Fig. 4 Accuracy of query results and communication cost when selective state update is used

5.2 Impact of Selective Update on Spatio-temporal queries

In this experiment, we show the impact of selective state update on spatio-temporal queries using simulated events that are generated from simulation of moving occupants in camera network [2]. Since selective update ignores small probabilities in events and negligible changes in occupant states, resulting application state can differ from the application state computed by the naive state update. Although the naive state update does not always guarantee correct answers due to its inherently probabilistic nature, we use the naive state update as a baseline to compare with our approximated application state resulting from selective state update.

In this experiment, we used two different types of queries. First type of query, called *location query*, asks whereabouts of a particular occupant. For instance, an application may ask top three most likely places for an occupant in order to select potential video streams to track the occupant. Another type of query, called *occupancy query* asks the occupants who are likely (with higher than 0.5 probability) to be in a specific zone. Using the two types of queries, we compare an approximate application state calculated by selective state update to an application state calculated by the naive state update. For each state update, we issue location queries and occupancy queries for all occupants and zones on the original application state and the approximate application state. If results for the same query differ between the original and approximate states, we count it as an error. Finally, we calculate the ratio of errors over all the query results.

[2] We simulated randomly moving occupants in a camera network with a grid topology while events pertain higher probability for a ground truth occupant and lower random probabilities for others.

Figure 4(a) shows the impact of selective update on the result of location queries. This experiment includes four different types of location queries asking different number of probable locations, which can be used for different application scenarios. For instance, *top-1* asks the most likely location for an occupant while *top-10* asks ten probable locations for an occupant to increase the chance to find the occupant. Over all types of location queries, the error ratio reduces when more occupants are selected for state update. However, the communication cost also increases due to the increased number of messages transmitted from an event queue to the occupant state workers. When more probable locations are asked, the error ratio is higher since there is a higher chance of disparity between query results from an approximate state and the original state.

Figure 4(b) shows the error ratio for occupancy queries that ask probable occupants in a specific zone. For occupancy queries, we use a different threshold to answer probable occupancy, ranging from 0.2 to 0.8. Similar to the previous experiment, error ratio reduces when more number of zones are selected for state update while communication cost linearly increases.

Our experimental results shown in Figure 4 indicate that using a small number of occupants and zones for selective state update can significantly reduce the communication overhead without significantly affecting the accuracy as measured by the error rates for the approximate state compared to the original state. For instance, if an application is interested in only the most probable location for each occupant, using only ten occupants for the selective update is sufficient to achieve the same accuracy as compared to the naive state update

6 Related Work

There are many distributed systems that help developing large-scale applications for camera networks. IBM S3 [2] provides a middleware for video-based smart surveillance system including video analytics and storage modules. Target Container [3] provides a parallel programming model and runtime system to help domain experts to develop large-scale surveillance application on distributed smart cameras. Above systems focused on processing raw video streams on distributed nodes, which is complementary to our system since we focus on collective inference on events that are generated from video streams.

Moving object databases [9] allow for spatio-temporal analysis by keeping track of mobile devices, like vehicles, and their location. Like our system, they allow for queries on uncertain data [4, 7], typically the GPS location. However, unlike situational information drawn from video data, the uncertainty is locally confined and thus does not require to update a global state which causes performance bottlenecks.

7 Conclusion

In this paper we addressed the scalability problems of situation-awareness applications using spatio-temporal analysis on large-scale camera networks. Due to the probabilistic nature of event generation and state update, an application has to keep a global application state that keeps track of known occupants in the observed area. We identify state update as the real performance bottleneck of spatio-temporal analysis, and presented solutions to address the bottleneck. To address the computation overhead of state update, we have presented a distributed state udpate with a partitioned application state over distributed nodes. To reduce the comomunication overhead of state update, we have proposed selective state update mechanisms. Our experimental results show that we can effectively remove the performance bottleneck of spatio-temporal analysis by applying selective state update while maintaining similar level of accuracy with the original application states.

References

1. Schiphol Airport leverages technology to drive efficiency. URL "http://www.securitysystemsnewseurope.com/?p=article&id=se200906VF2Isw"
2. Feris, R., Hampapur, A., Zhai, Y., Bobbitt, R., Brown, L., Vaquero, D., Tian, Y., Liu, H., Sun, M.T.: Case-Study: IBM smart surveillance system. In: Y. Ma, G. Qian (eds.) Intelligent Video Surveillance: Systems and Technologies. Taylor & Francis, CRC Press (2009)
3. Hong, K., Smaldone, S., Shin, J., Lillethun, D., Iftode, L., Ramachandran, U.: Target container: A target-centric parallel programming abstraction for video-based surveillance. In: Distributed Smart Cameras (ICDSC), 2011 Fifth ACM/IEEE International Conference on, pp. 1 –8 (2011)
4. Kalashnikov, D.V., Ma, Y., Mehrotra, S., Hariharan, R.: Index for fast retrieval of uncertain spatial point data. In: Proc. of Int'l Symposium on Advances in Geographic Information Systems (ACM GIS 2006). Arlington, VA, USA (2006)
5. Menon, V., Jayaraman, B., Govindaraju, V.: The Three Rs of Cyberphysical Spaces. Computer **44**, 73–79 (2011)
6. Ramachandran, U., Hong, K., Iftode, L., Jain, R., Kumar, R., Rothermel, K., Shin, J., Sivakumar, R.: Large-scale situation awareness with camera networks and multimodal sensing. Proceedings of the IEEE **100**(4), 878 –892 (2012)
7. Trajcevski, G., Ding, H., Scheuermann, P., Cruz, I.: Bora: Routing and aggregation for distributed processing of spatio-temporal range queries. In: Mobile Data Management, 2007 International Conference on, pp. 36 –43 (2007)
8. Wang, L., Tan, T., Ning, H., Hu, W.: Silhouette analysis-based gait recognition for human identification. Pattern Analysis and Machine Intelligence, IEEE Transactions on **25**(12), 1505–1518 (2003)
9. Wolfson, O., Xu, B., Chamberlain, S., Jiang, L.: Moving objects databases: issues and solutions. In: Scientific and Statistical Database Management, 1998. Proceedings. Tenth International Conference on, pp. 111–122 (1998)
10. Zhao, W., Chellappa, R., Phillips, P.J., Rosenfeld, A.: Face recognition: A literature survey. ACM Comput. Surv. **35**(4), 399–458 (2003)