CAESAR: Context-Aware Event Stream Analytics for Urban Transportation Services

Olga Poppe*, Chuan Lei**, Elke A. Rundensteiner*, Dan Dougherty*, Goutham Deva*, Nicholas Fajardo*, James Owens*, Thomas Schweich*, MaryAnn VanValkenburg*, Sarun Paisarnsrisomsuk*, Pitchaya Wiratchotisatian*, George Gettel*, Robert Hollinger*, Devin Roberts*, and Daniel Tocco*
*Worcester Polytechnic Institute, 100 Institute Road, Worcester, MA 01609
**NEC Labs America, 10080 N Wolfe Rd, Cupertino, CA 95014
*opoppe|rundenst|dd|godeva|nafajardo|jmowens|taschweich|mevanvalkenburg|spaisarnsrisomsu|pwiratchotisatia|gtgettel|rhollinger|dtroberts|dtocco@wpi.edu, **chuan@nec-labs.com

ABSTRACT

We demonstrate the first full-fledged context-aware event processing solution, called CAESAR[1], that supports application contexts as first class citizens. CAESAR offers human-readable specification of context-aware application semantics composed of context derivation and context processing. Both classes of queries are only relevant during their respective contexts. They are suspended otherwise to save resources and to speed up the system responsiveness to the current situation. Furthermore, we demonstrate the context-driven optimization techniques including context window push-down and query workload sharing among overlapping context windows. We illustrate the usability and performance gain of our CAESAR system by a use case scenario for urban transportation services using real data sets [2, 1].

1. INTRODUCTION

Context-Aware Event Stream Analytics. Complex Event Processing (CEP) is a prominent technology for supporting time-critical applications. Traditionally, CEP systems consume an event stream and continuously evaluate the *same* query workload against the *entire* event stream. However, the semantics of many streaming applications is determined by contexts, meaning that *the system reaction to one and the same event may significantly vary depending on the context*. Therefore, most event queries are appropriate only under certain circumstances and can be safely suspended otherwise to save valuable resources and reduce the latency of currently relevant queries.

Running Demonstration Scenario. With the growing popularity of Uber and Lyft, their real-time systems face a wide range of challenges, including but not limited to extracting supply/demand sequence patterns from event streams, real-time aggregation, geospatial prediction, traffic data monitoring and alerting. These data intensive event queries continuously track the status of drivers, riders, and traffic, such as driver dispatched, rider waiting for pickup, road congestion, etc. An intelligent event processing system receives both vehicle and rider position reports and their associated messages, analyzes them, infers the current supply and demand situation in each geolocation, and reacts instantaneously to ensure that riders reach their destinations in a timely and cost-effective manner. Early detection and prompt reaction to critical situations are indispensable. They prevent time waste, reduce costs, increase riders' satisfaction and drivers' profit.

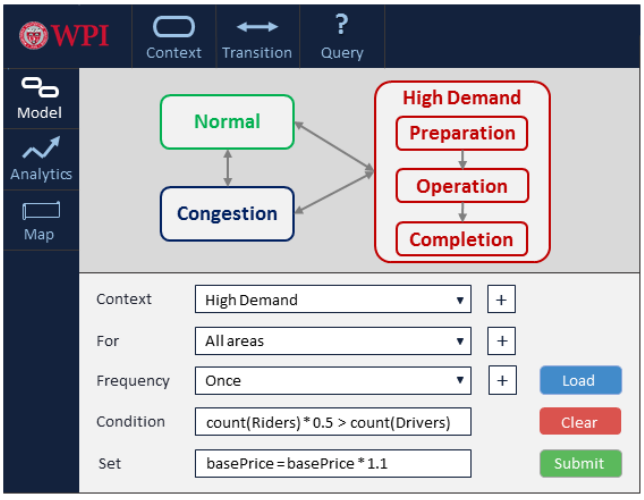

Figure 1: The CAESAR model

System reaction to a position report should be modulated depending on the *current situation on the road*. Indeed, if *HighDemand* is detected, all drivers close by are notified and a higher fee is charged in this area to attract more drivers and reduce the waiting time of riders (Figure 1). If a road segment becomes *Congested*, drivers may be alerted and alternate routes should be advised so as to smooth traffic flow. If a road segment is *Normal*, none of the above

[1]CAESAR stands for Context-Aware Event Stream Analytics in Real time.

actions should take place. Clearly, *current application contexts* must be rapidly detected and continuously maintained to determine appropriate reactions of the system at all times.

The hierarchical application logic in Figure 1 is drilled down into the *HighDemand* specification, while the interior structures of all other processes are rolled up and thus abstracted to increase readability. Three contextual stages are differentiated during the *HighDemand* context. First, *all* drivers in proximity are notified and a higher base fee is computed (*Preparation*). Afterwards, only *new* nearby drivers are notified and the high base fee is used to compute the cost of each trip (*Operation*). Lastly, the base fee is reduced once demand is satisfied (*Completion*). Appropriate event queries are associated with each context. For example, new drivers are detected during the *Operation* phase in a high demand geolocation (Figure 1).

Conditions implying an *application context* can be complex. They are specified on both the event streams and the current contexts. For example, if over 50 cars per minute move with an average speed less then 40 mph and the current context is no *Congestion* then the *context-deriving query* updates the context to *Congestion* for this geolocation. To save resources and thus to ensure prompt system responsiveness, such complex context detection should happen once. Its results must be available immediately and shared among all queries that belong to the detected context. In other words, *context-processing* queries are *dependent* on the results of *context-deriving* queries. A synchronization mechanism ensuring their correct execution must be employed.

Challenges. To enable real-time responsiveness of such applications, the following challenges must be tackled.

Context-aware specification model. Many streaming applications have context-driven semantics. Thus, they must support application contexts as first class citizens and enable linkage of the appropriate event query workloads to their respective contexts in a modular format to facilitate on-the-fly reconfiguration, easy maintenance, and avoid fatal specification mistakes.

Context-exploiting optimization techniques. To meet the demanding latency constraints of time-critical applications, this context-aware application model must be translated into an efficient physical query plan. This is complicated by the fact that the duration of a context is unknown at compile time and potentially unbounded. Furthermore, contexts are implied by complex conditions. They are interdependent and may overlap.

Context-driven execution infrastructure. An efficient runtime execution infrastructure is required to support multiple concurrent contexts. To ensure correct query execution, the inter-dependencies between context-deriving and context-processing queries must be managed effectively.

State-of-the-Art Approaches. Traditional CEP windows fail to express variable-length inter-dependent context windows. Indeed, tumbling and sliding windows [8] have fixed length, while predicate windows [6] are defined independently from each other.

Most graphical models express either only the workflow [4] or only single event queries [3]. Some models and event languages can express contexts by procedures [7] or queries [5]. However, they do not allow for the modular specification of context-driven applications – placing an unnecessary burden on the designer [9, 10]. Furthermore, optimization techniques enabled by contexts are yet to be developed.

Contributions. We demonstrate the following contributions of the CAESAR technology: (1) The easy-to-use graphical interface to illustrate the powerful CAESAR model [9, 10]. It visually captures application contexts, transitions between them, context-deriving and context-processing queries. (2) The optimization techniques enabled by contexts suspend those queries that are irrelevant to the current context and share computations between overlapping contexts. (3) The CAESAR infrastructure guarantees correct and efficient context management at runtime. (4) We illustrate the usability and performance gain of the CAESAR technology using the real-world urban transportation scenario [2, 1].

2. CAESAR SYSTEM OVERVIEW

Figure 2 provides an overview of the CAESAR system.

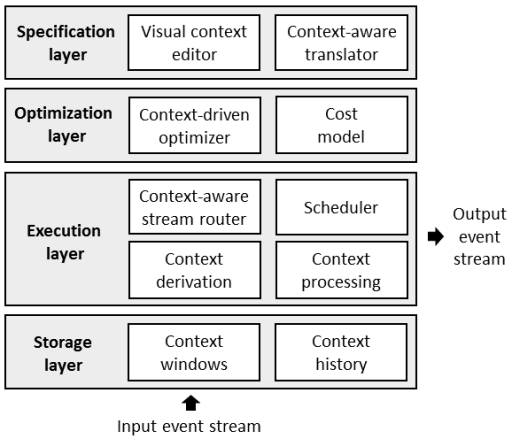

Figure 2: The CAESAR system

Specification Layer. The designer specifies the CAESAR model (Section 3.1) using the visual context editor. The model is then translated it into an algebraic query plan.

Optimization Layer. The query plan is optimized using the context-driven optimization techniques (Sections 3.2 and 3.3) to produce an efficient execution plan.

Execution Layer. The optimized query plan is forwarded to the scheduler that guarantees correct context derivation and processing at runtime (Section 3.4).

Storage Layer. Context windows and history are compactly stored and efficiently maintained at runtime.

3. KEY INNOVATIONS OF CAESAR

3.1 Context-aware Event Query Model

While the CAESAR model is formally defined in [9, 10], below we briefly summarize its key components and benefits.

Application Contexts are real-world higher-order situations the duration of which is not known at their detection time and potentially unbounded. This differentiates contexts from events. The duration of an application context is called a *context window*. For example, *Congestion* is a higher-order situation in the traffic use case. Its bounds are detected based on position reports of cars in the same area at the same time. As long as a road remains congested, the context window *Congestion* is said to hold. Hence, the duration of a context window cannot be predetermined.

At each point of time, the CAESAR model re-targets all efforts to the current situation by activating only those context-

deriving and context-processing queries which handle the current contexts. Irrelevant queries are suspended to save resources. For example, Uber surge pricing kicks in only during *HighDemand* on a road. This query is neither relevant in the *Normal* nor in the *Congestion* contexts. Thus, it is evaluated only during *HighDemand* and suspended in all other contexts.

Context-Deriving Queries are associated with a particular context and determine when this context is terminated and when a particular other context is initiated based on events. For example, once many slow cars on a road are detected during the *Normal* context the system transitions into the *Congestion* context. Thereafter, the query detecting *Congestion* is no longer evaluated. All event queries that are evaluated during *Congestion* leverage the insight detected by the context-deriving query rather than re-evaluating the *Congestion* condition at each individual query level.

Context-Processing Queries react to events that arrive during a context in an appropriate way. Contexts provide queries with *situational knowledge* that allows to specify simpler event queries. For example, if the query computing surge pricing is evaluated only during the *HighDemand* context, the complex conditions that determine that there is a high demand in this geolocation are already implied by the context. Thus, there is no need to repeatedly double-check them in each of the context-processing queries.

3.2 Context Window Push-Down Optimization

Our CAESAR algebra consists for the following six operators: Context initiation, context termination, context window, filter, projection, and pattern [10]. With filter, projection and pattern common in stream algebras [12], traditional multi-query optimization techniques [11] are applicable to our CAESAR queries. In addition, we propose two context-driven optimization techniques, namely we push context windows down and share workloads of overlapping contexts. Pushing context windows down in a query plan prevents the continuous execution of operators "out" of their respective contexts and thus reduces the costs. To guarantee correctness, we group event queries by contexts. By definition, a context window specifies the scope of its queries. Thus, pushing a context window down in each group of queries does not change the semantics of these queries.

In contrast to traditional predicates, context windows are not just filters on a stream that select certain events to be passed through. Context windows *suspend* the entire query plan "above them" as long as the application is in different contexts. Furthermore, our context-driven stream router directs entire *stream portions during contexts* to their respective queries (Section 3.4) rather than filtering events one by one at the *individual event level* which is a resource-consuming process.

3.3 Context Workload Sharing Optimization

Similar computations may be valid in different contexts. For example, an accident on a road is detected during all contexts in Figure 1. In such cases, substantial computational savings can be achieved by sharing workloads between overlapping contexts. For example, *Congestion* and *HighDemand* may overlap. To avoid repeated computations and storage, we split the original user-defined *overlapping* context windows into finer granularity context windows and group them into *non-overlapping* context windows by merging their workloads. Within each newly produced non-overlapping context window, we apply traditional multi-query optimization techniques [11]. Our context window grouping strategy divides the query workload into smaller groups based on their time overlap. As additional benefit, the search space for an optimal query plan within each group is substantially reduced compared to the global space.

3.4 CAESAR Execution Fabric

The core of the CAESAR execution fabric consists of the context derivation, context-aware stream routing, context processing, and scheduling of these processes (Figure 2). While we briefly describe these components below, we refer an interested reader to our full paper [10] for more details.

Context Derivation. For each stream partition (a geolocation in the traffic use case), the context bit vector W maintains the currently active contexts. This vector W has a time stamp $W.time$ and a one-bit entry for each context. The entry 1 (0) for a context c means that the context c holds (does not hold) at the time $W.time$. Since contexts may overlap, multiple entries in the vector may be set to 1. $W.time$ is the application time when the vector W was last updated by the context-deriving queries. This time stamp is crucial to guarantee correctness of interdependent queries.

Context-Aware Stream Routing. Based on the context bit vector, the system is aware of the currently active contexts. For each current context c, the system routes all its events to the query plan associated with the context c. Query plans of all currently inactive contexts do not receive any input. They are suspended to avoid waste of resources.

Context Processing. The CAESAR model uses contexts to specify the *scope* of queries. When a user-defined context ends, all associated queries are suspended and thus will not produce new results until they become activated again. Hence, their partial results, called *Context history*, can be safely discarded. However, if a user-defined context c with its associated query workload Q^c is split into smaller non-overlapping contexts c_1 and c_2, then partial results of the queries Q^c must be maintained across these new contexts c_1 and c_2 to ensure completeness of the queries Q^c.

Correctness. Context-processing queries are dependent on the results of context-deriving queries. To avoid race conditions and ensure correctness, these inter-dependencies must be taken into account. To this end, we define a *stream transaction* as a sequence of operations that are triggered by all input events with the same time stamp. An algorithm for scheduling read and write operations on the shared context data is *correct* if conflicting operations[2] are processed by sorted time stamps. While existing stream transaction schedulers could be deployed in the CAESAR system, we currently deploy a time-driven scheduler.

4. DEMONSTRATION SCENARIO

In this section, we demonstrate the above key innovations of the CAESAR system based on the urban transportation services using two real data sets [2, 1] that contain millions of taxi and Uber trips in New York city in 2014 and 2015.

Visual CAESAR Model Design. The audience will view and edit CAESAR models using simple drag-and-drop interaction tools. Figure 1 shows that the model captures

[2]Two operations on the same value such that at least one of them is a write are called conflicting operations.

the complex application logic in a succinct and readable manner. The audience can view the specification at different levels of abstraction. There are three composed contexts, namely, *Normal, Congestion*, and *High Demand*. All other contexts are atomic. The composed contexts can be collapsed and expanded with a click of a button. For ease of follow-through, color schemas of composed contexts and their interior structures are consistent. To keep the model clean and readable, the contexts and transitions between them are depicted in the middle panel separately from their respective context-deriving and context-processing queries shown in the bottom panel. When the cursor is over a transition, its corresponding context-deriving query appears as a label of the transition. When the designer clicks on a label, (s)he can conveniently edit it in the bottom panel. Similarly, when the designer clicks on a context, the list of context-processing queries appears in the bottom panel. We will demonstrate the ease with which CAESAR models can be dynamically reconfigured by editing contexts, transitions between them, and their respective queries.

Execution Visualization. At runtime, the model view provides insights into event-driven context transitions (Figure 1). The current context and triggering transitions are temporally highlighted. Besides the real-time monitoring, the model view offers a slow-motion-replay mode that allows the users to step-through the history of prior execution to better understand, debug, and reconfigure the model. This functionality provides the audience a visual opportunity to learn how the CAESAR model functions.

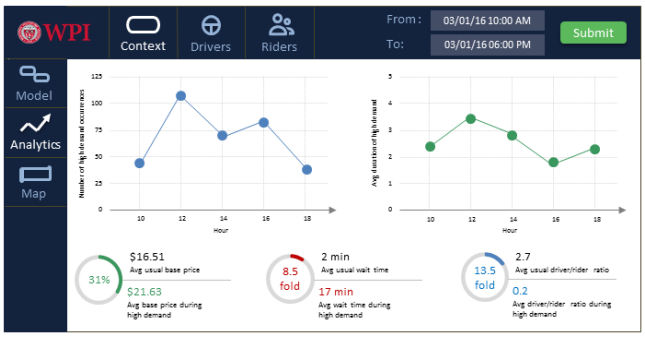

Figure 3: Analytics view

Execution Optimization. The analytics view will allow the audience to monitor the effect of the context-driven optimization techniques. The audience will first chose to show statistics either about contexts, or drivers, or riders in the top panel of Figure 3. Also, the audience can specify the time interval of interest in the top panel. Thereafter, charts visualizing runtime statistics will appear in the middle panel. They provide a summary about the chosen topic during the time interval of interest. For example, to summarize the contextual information, the number of high demand occurrences, average duration of this context, as well as the price, wait time, and driver vs. rider ratio during 8 hours are compactly presented in Figure 3.

Interactive City Map offers the audience an abstract view of the current situation by highlighting the areas in different colors depending on their contexts. For example, a high demand area is identified in the middle panel in Figure 4 highlighted by a red circle. Green and blue circles visualize riders outside of the high demand area. In addition to the map, runtime statistics are shown in the top panel. They include the number of current high demand or congested areas, the number of recent requests and current trips, the number of available drives and waiting riders.

In addition to the complex events that are automatically derived by queries, the audience will learn about common manual actions which include area specific information such as accidents, road construction, gas prices, police cars etc. This information will be added by clicking on the respective location on the map and choosing the information in a drop-down menu. A respective icon will appear on the map. For example, one traffic hazard is depicted in Figure 4. Based on this information, travel time and cost will be estimated to compute the best route of each trip.

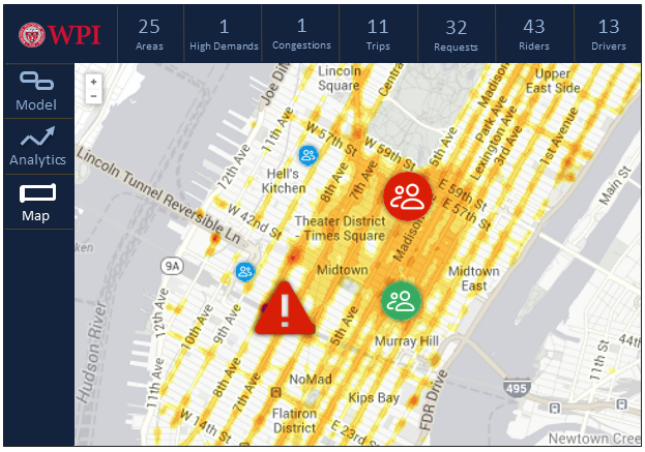

Figure 4: Map view

Conclusion. The CAESAR technology offers a principled end-to-end solution for context-aware stream analytics.

Acknowledgement. This work was supported by NSF grants IIS 1018443, IIS 1343620, IIS 1560229, and CRI 1305258.

5. REFERENCES

[1] Uber TLC FOIL Response. https://github.com/fivethirtyeight/uber-tlc-foil-response.

[2] Unified New York City Taxi and Uber data. https://github.com/toddwschneider/nyc-taxi-data.

[3] J. Agrawal, Y. Diao, D. Gyllstrom, and N. Immerman. Efficient pattern matching over event streams. In *SIGMOD*, pages 147–160, 2008.

[4] R. Alur and D. Dill. Automata for modeling real-time systems. In *ICALP*, pages 322–335, 1990.

[5] A. Arasu, S. Babu, and J. Widom. The CQL continuous query language: semantic foundations and query execution. *VLDB Journal*, 15(2):121–142, 2006.

[6] T. M. Ghanem, W. G. Aref, and A. K. Elmagarmid. Exploiting predicate-window semantics over data streams. *SIGMOD Rec.*, 35(1):3–8, 2006.

[7] A. Grosskopf, G. Decker, and M. Weske. *The Process: Business Process Modeling using BPMN*. Meghan Kiffer Press, 2009.

[8] J. Li, D. Maier, K. Tufte, V. Papadimos, and P. Tucker. No pane, no gain: Efficient evaluation of sliding-window aggregates over data streams. In *SIGMOD*, pages 39–44, 2005.

[9] O. Poppe, S. Giessl, E. A. Rundensteiner, and F. Bry. The HIT model: Workflow-aware event stream monitoring. In *TLDKS*, pages 26–50. 2013.

[10] O. Poppe, C. Lei, E. Rundensteiner, and D. Dougherty. Context-aware event stream analytics. In *EDBT*, pages 413–424, 2016.

[11] P. Roy, S. Seshadri, S. Sudarshan, and S. Bhobe. Efficient and extensible algorithms for multi query optimization. In *SIGMOD*, pages 249–260, 2000.

[12] E. Wu, Y. Diao, and S. Rizvi. High-performance Complex Event Processing over streams. In *SIGMOD*, pages 407–418, 2006.