Demo: Approximative Event Processing on Sensor Data Streams

Christoffer Löffler[1]
loefflcr@iis.fraunhofer.de

Christopher Mutschler[1,2]
christopher.mutschler@fau.de

Michael Philippsen[2]
michael.philippsen@fau.de

[1] Fraunhofer Institute for Integrated Circuits IIS
Locating and Communication Department
Sensor Fusion and Event Processing Group
Erlangen, Germany

[2] Friedrich-Alexander University Erlangen-Nürnberg (FAU)
Department of Computer Science
Programming Systems Group
Erlangen, Germany

ABSTRACT

Event-Based Systems (EBS) can efficiently analyze large streams of sensor data in near-realtime. But they struggle with noise or incompleteness that is seen in the unprecedented amount of data generated by the Internet of Things.

We present a generic approach that deals with uncertain data in the middleware layer of distributed event-based systems and is hence transparent for developers. Our approach calculates alternative paths to improve the overall result of the data analysis. It dynamically generates, updates, and evaluates Bayesian Networks based on probability measures and rules defined by developers. An evaluation on position data shows that the improved detection rate justifies the computational overhead.

Categories and Subject Descriptors

C.2.4 [**Computer-Comm. Networks**]: Distrib. Syst.—*Distrib. Applications*

Keywords

Event Processing; Uncertainty; Bayesian Networks.

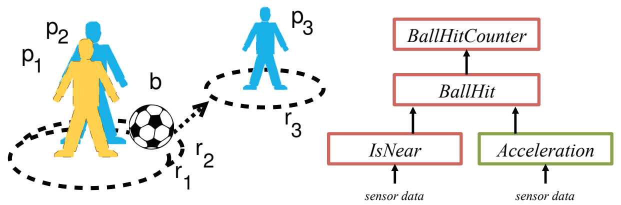
1. INTRODUCTION

Complex Event Processing (CEP) and Event-Based Systems (EBS) are the methods of choice for a near-realtime reactive analysis of data streams in applications such as surveillance, sports, RFID-systems, stock trading, etc. [1].

CEP infers knowledge from data streams by applying event patterns or it detects a-priori unknown patterns, e.g., by means of machine learning [2]. It also allows timely reactions to detected events, e.g., in Business Process Management. CEP can even fuse information along a complex dependency hierarchy to detect abstract patterns, e.g., credit card fraud. But real-world use cases often face uncertain data, which can result in false events.

In Fig. 1 a tracking system locates the soccer players p_1 to p_3 and the ball b and uses CEP to analyze the position data streams to detect and count ball hits with a hierarchy of event detectors (EDs). A player p_i can only hit a ball if it is sufficiently close, i.e., within the radius r_i. But due to noisy measurements and incomplete information it is difficult to clearly identify whether p_1 or p_2 has hit the ball as both are close. A naive *BallHit*-ED detects both events. Counting those mutually exclusive hit events is prone to errors.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from Permissions@acm.org.
DEBS'15, June 29 - July 3, 2015, OSLO, Norway.
ACM 978-1-4503-3286-6/15/06 ...$15.00.
DOI: http://dx.doi.org/10.1145/2675743.2776767.



(a) b moves from $r_1 \cap r_2$ to r_3. (b) Detector hierarchy.

Figure 1: Uncertainty in event detection.

More elaborate (and harder to develop) EDs that deal with uncertainty, as well as higher level EDs that deal with event noise on the application level may help avoid such problems. Instead, we propose to keep EDs simple, to assign probabilities to the events, and to propagate these approximative events up the detection hierarchy. The developer may also add domain knowledge to the probabilities, e.g., to express that successive ball hits are more likely to come from the same player.

Our EBS middleware uses a sliding window over the uncertain event log and lets detectors transparently and speculatively process all meaningful combinations of occurring and ignored events. We externalize only the best results after the sliding window passed. Speculation along detection hierarchies is supported through techniques used for knowledge integration in databases. The likelihood of events and their (in)validation is determined by Bayesian Networks.

2. SYSTEM ARCHITECTURE

Fig. 2 shows how the middleware works with the *BallHit* and *BallHitCounter (BHC)* detectors from Fig. 1. The *BallHit* detector subscribes to *IsNear* events (triggered when the ball gets close to a player) and *Acceleration (Acc)* events (triggered when the ball changes its direction). Because of the noise these events have probabilities (in percentages). For a sliding event log window of size 3 of the event log ($IsNear_{p_1}$, $IsNear_{p_2}$, Acc) the middleware builds a binary tree that spans all possible sets of input events. For each of these sets we hold a snapshot that the *BallHit* detector uses as

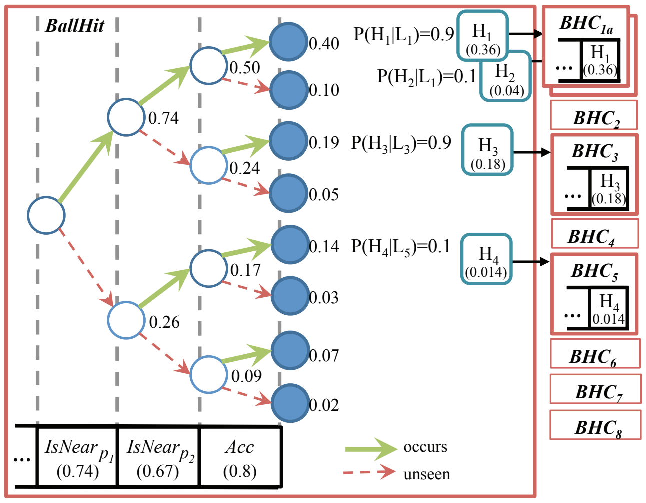


Figure 2: Speculative processing and isolated propagation of events between two event detectors.

an internal state to process the input set on. The leaves represent (active) state snapshots, i.e., L_3=($IsNear_{p_1}$, Acc) is represented by the third leaf. Conceptually, for each node we calculate its probability by multiplying the event probabilities, respectively the inverse probability for unseen events. In 3 of the 8 cases, the *BallHit*-ED generates the hit events H_1 to H_4 for each player that is close to the accelerating ball. Their probabilities are derived from the state probabilities and developer-provided rules, see below.

Conceptually, *BallHitCounter* simply counts hits. But as *BallHit* not only spawns events with probabilities but also generates incompatible (mutually exclusive) events, *BallHitCounter* receives more events than it should. Hence, for each leaf of *BallHit*'s tree there is at least one *BallHitCounter*-ED for the respective *possible world*, i.e., the set of events of the branch and the respective spawned events. Internally we annotate compatibility information to the spawned events to fully isolate further processing.

2.1 Annotating Probabilities

If there is neither noise nor uncertainty, events have a default probability of 1. This turns our approximative EBS into a traditional deterministic EBS (for this type of event).

In case of noise, low-level sensor-based EDs can annotate probabilities before handing events to the middleware. In the example, *Acc* has a probability of 0.8.

While traditional EDs further up in the detection hierarchy usually have to decide whether or not to spawn an event,

Algorithm 1: *BallHit* ED annotation example.

```
Data: NearPlayers players; Event recvd;
1  if recvd equals IsNear then
2      NearPlayers.add(recvd.getPlayer());
3  if recvd equals Acc then
4      if playersNear() equals 1 then
5          Player p ← players.getPlayer();
6          float prob ← 1.0 - p.getDistance();
7          send(BallHit(players.GetPlayer(), prob));
8      else if playersNear() equals 2 then
9          send(BallHit(players.getEarliest(), 0.9));
10         send(BallHit(players.getLatest(), 0.1));
```

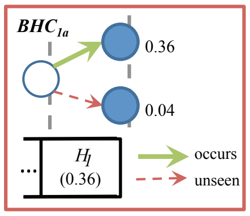


Figure 3: BHC_{1a} from Fig. 2 processes H_1.

we add a way for the domain expert to not only annotate probabilities to the spawned events but also to spawn more than one event if there are several options/possible worlds.

Consider the above example again and assume the following domain knowledge: (a) If there is only one player close to the ball, the probability that this player has actually kicked the ball (instead of a random acceleration caused by the ball bouncing off the floor) is the higher the closer the player is to the ball. Alg. 1 deals with this case in lines 4-7. From the viewpoint of the domain expert, if p_2 is 0.9m (quite far) away from the ball (and p_1 is not anywhere close), then the probability of H_4 is just $P(H_4|L_5)$=0.1, see Fig. 2.

(b) If there are two players close to the ball, either one may have kicked the ball. With our system, the ED no longer has to pick one, but instead, the domain expert can now spawn two events H_1 and H_2, one for each option. The annotated probabilities capture the domain knowledge (lines 8-10 in Alg. 1) that it is more likely that the player kicked who got close to the ball first. Events H_1 and H_2 are mutually exclusive and represent two different possible worlds.

The middleware understands the exclusiveness as can be seen in Fig. 3: When constructing the branches of the binary tree for the *BallHitCounter*-ED BHC_{1a}, it assigns the residual chance of 0.04 to the unseen-branch instead of the inverse probability (1-0.36=0.64).

2.2 Scoring Possible Worlds

Note that when implementing an ED the domain expert assigns probabilities to events without taking into account that the middleware will presents many potential event log situations to it, each of which has a different probability itself. By means of a Bayesian Network, our system transparently combines the two probabilities (middleware-provided leaf probability and domain-expert's event probability). For example for H_4 this results in a probability of

$$\begin{aligned} P(World_{H_4}) &= P(1\text{-}IsNear_{p_1}) \cdot P(IsNear_{p_2}) \cdot P(Acc) \cdot \\ &\quad P(H_4|L_5{=}\neg IsNear_{p_1}, IsNear_{p_2}, Acc) \\ &= 0.26 \cdot 0.67 \cdot 0.8 \cdot 0.1 \approx 0.014. \end{aligned}$$

The resulting probability of H_4 (and similarly for H_3 for which p_1 is just 0.1m away from the ball) is hence lower than what the leaf probability suggests, as a residual chance of a bouncing ball is taken into account.

2.3 Pruning

The larger the size of the sliding window, the deeper is the binary tree, and the more EDs on the next layer of the detection hierarchy are created, and which in turn recursively cause the same fan-out again. To restrict the total number of running ED instances and to limit the amount of computing power needed for the trees of potential event logs, we have to prune. To do so, when a threshold is reached, the middleware sorts the probabilities of the possible worlds. For that half of the tree that does not hold the most likely branch, the middleware tears down the branches and eliminates obsolete EDs by walking up the tree towards the root nodes and by sending invalidation events that rewind the event detection process. For the oldest event in the event log the middleware sets the probability to 1 or 0. The affected events are then either processed further up in the detection hierarchy or they are externalized to the user of the EBS.

3. RELATED WORK

This paper studies the problems in analyzing uncertain event streams along a detection hierarchy and presents a generic solution that is transparent for application developers. Such a generic approximative event processing must satisfy a number of requirements. As CEP-application developers mainly focus on the implementation of the EDs, the additional annotation efforts must be minimal and user-friendly. Any probability approximations should be handled under the hood. Also, as CEP often requires scaling, the approach must be suitable for distributed EBSs.

Rule-based systems such as the Event Calculus handle uncertainty with Markov Logic Networks (MLNs) extended with temporal semantics [4]. However, as they strictly rely on a formal system instead of being language-agnostic they require significant re-adjustment by developers.

Wasserkrug et al. [5] propose a formalism for knowledge representation and inference to automatically generate Bayesian Networks from application-specific rules. To boost speed they replace Bayesian Inference with Monte-Carlo sampling and use a selection mechanism that filters relevant events [6]. We require more filtering, e.g., a transparent mechanism that can eliminate unlikely interim results in distributed event detection hierarchies.

Active databases that use the event-condition-action paradigm have been extended with fuzzy conditions to handle uncertainty [7]. However, they focus on uncertain ECA-rules rather than uncertain data, and they do not support speculative calculations. Modern probabilistic databases support fast query evaluation (e.g., by using Monte-Carlo simulation) and ranked lists [8] but are not extending them to active rules. Uncertain lineage databases (ULDBs) [9] calculate confidence values quickly but do not support temporal data processing.

Another active field of research extends existing NFA- and stream-based languages or systems [10, 14] to handle uncertainty but either uses simplistic probability calculations [11] or integrates an insufficient approach for these calculations from ULDBs [12]. Li et al. [13] analyze uncertain event streams for error correction in noisy environments but not for more complex inferencing.

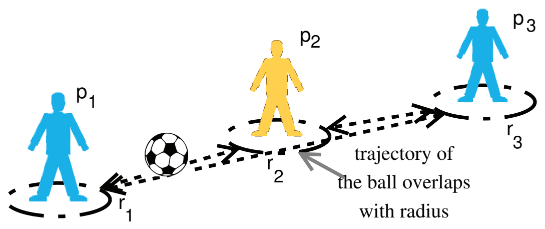


Figure 4: Players p_1 and p_3 pass to each other. The ball b's trajectory cuts into p_2's radius r_2.

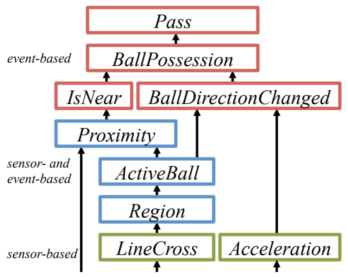


Figure 5: Benchmark event detection hierarchy.

4. EVALUATION

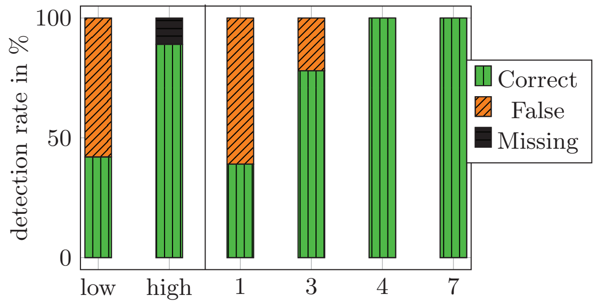
Sec. 4.1 evaluates the event detection rate and compares it to a deterministic baseline system. We examine the influence of statistical knowledge when modeling the Bayesian Networks, how it behaves as a filter for faulty or unlikely results, and how its size influences the detection rate. Sec. 4.2 focuses on the runtime cost of the approximation.

Fig. 5 shows the detector hierarchy that we use for benchmarking. *ActiveBall* detects if a ball is inside a *Region*, e.g., the soccer field, which uses *LineCross* events to detect if the ball crosses a line. *Proximity* subscribes to the streams of player and ball position and to *ActiveBall*. The *IsNear* event indicates that a player is close to the ball (threshold τ_p=1m). *BallDirectionChanged* uses peaks in the *Acceleration* (threshold $\tau_a=100\frac{m}{s^2}$). *BallPossession* spawns a *begin-of-possession* event when an *IsNear* event is followed by a *ball-direction-changed* event (for example caused by stopping or hitting the ball). An *end-of-possession* event is spawned whenever another player starts to possess the ball. Finally, a *Pass* indicates change of possession.

In the benchmark scenario in Fig. 4 players p_1 and p_3 pass the ball between them and back. Player p_2 sometimes can intercept these passes. Afterwards p_2 then passes the ball to either p_1 or p_3. The uncertainty problems are caused by the ball's trajectory that may come close to p_2's position, and by the ball's ambiguous deceleration (if it bounces off the floor or if p_2 touches it without intercepting the pass).

4.1 Detection Rate

Fig. 6 compares the approximative system to the baseline system. The baseline uses both a low detection threshold for deceleration ($\tau_a=100\frac{m}{s^2}$) and high threshold ($\tau_a=200\frac{m}{s^2}$). The approximative system uses different sliding window sizes but only the low threshold.



left: baseline CEP, low/high detection thresholds
right: approximative CEP, various sliding window sizes

Figure 6: Pass detection rates.

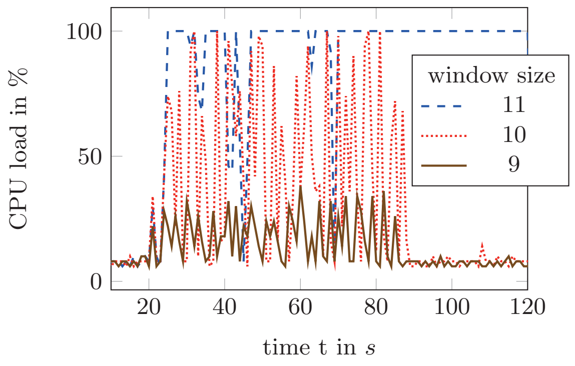


Figure 7: System loads for different window sizes.

The baseline with a low threshold finds 11 false-positive passes, whereas with the high threshold it misses 1 pass (false-negative). With a windows size of 1, the approximative CEP is identical to the baseline. With larger sliding windows for the *Pass* detector, more ED instances run and process alternative paths. This significantly improves the detection rate. A window size of 3 improves it from 39% to 79%. A size of 4 achieves perfect detection without any false-positives or false-negatives.

4.2 System Load

The ideal window size of 4 (and the corresponding number of running ED instances and trees of possible worlds) does not noticeably affect the CPU load (Intel Core i5 750 CPU @ 2.67GHz with 4GB of main memory).

As more complex event detection scenarios may require larger sliding windows, a broader set of possible worlds, and more active ED instance, the CPU load can become the bottleneck as is depicted in Fig. 7. For this measurement, the players pass the ball for 70 seconds, from t=20 to t=90. The plot shows the system load for varying numbers of *Pass* detectors, i.e., different sizes of the sliding windows. Even with 512 detectors (window size 9) we still see a good load level of 30-40%. The memory impact of the approximation is negligible due to the small size of detectors' snapshots. With a window size of 10, there are the first peaks in the CPU load that cause delays and hence missed detections, e.g., at t=33 and t=39. With 2048 detectors (window size 11) the CPU load often maxes out after t=5.

In summary, there is enough head room above this benchmark's ideal window size of 4 to deal with more challenging scenarios before more compute power is needed.

5. CONCLUSION

Our approach to approximative event processing is based on multiple worlds that increase the solution space on the set of input events. As resulting (intermediate) events can be contradictory, faulty or unlikely we isolate the processing of subscribing detectors. With a small set of generic annotation capabilities developers can express domain knowledge about uncertainty. We construct, and continuously update and evaluate a Bayesian Network on which we automate the choice of the most likely result and propagate the choices of the isolated processing to the subscribers.

The evaluation shows that approximation captures more details of the data and results in better detection rates. It also shows that the exponentially growing depth of the detection hierarchy leaves enough room for practical use.

Future work will extend the approximation from covering just event probabilities to also address event attributes. This might capture even more details of the data for improved detection quality.

6. ACKNOWLEDGMENT

This contribution was supported by the Bavarian Ministry of Economic Affairs and Media, Energy and Technology as a part of the Bavarian project „Leistungszentrum Elektroniksysteme (LZE)“.

7. REFERENCES

[1] M. Stonebraker, U. Çetintemel, and S. Zdonik, “The 8 requirements of real-time stream processing,” *SIGMOD Rec.*, vol. 34, no. 4, pp. 42–47, 2005.

[2] C. Mutschler and M. Philippsen, “Learning event detection rules with noise hidden markov models,” in *NASA/ESA Conf. Adaptive Hardware and Systems*, (Nuremberg, Germany), pp. 159–166, 2012.

[3] A. Skarlatidis, G. Paliouras, G. Vouros, and A. Artikis, “Probabilistic event calculus based on markov logic networks,” in *RuleML America*, pp. 155–170, Springer, Berlin, 2011.

[4] S. Wasserkrug, A. Gal, and O. Etzion, “A model for reasoning with uncertain rules in event composition systems,” in *21st Conf. Uncertainty in Artificial Intelligence*, (Edinburgh, Scotland), pp. 699–606, 2005.

[5] S. Wasserkrug, A. Gal, O. Etzion, and Y. Turchin, “Efficient processing of uncertain events in rule-based systems,” *IEEE Trans. Knowledge and Data Engineering*, vol. 24, no. 1, pp. 45–58, 2012.

[6] T. Bouaziz and A. Wolski, “Applying fuzzy events to approximate reasoning in active databases,” in *6th Intl. Conf. Fuzzy Systems*, (Barcelona, Spain), pp. 729–735, 1997.

[7] N. Dalvi and D. Suciu, “Efficient query evaluation on probabilistic databases,” in *30th VLDB Conf.*, (Toronto,Canada), pp. 523–544, 2004.

[8] O. Benjelloun, A. D. Sarma, A. Halevy, and J. Widom, “Uldbs: Databases with uncertainty and lineage,” in *32nd VLDB Conf.*, (Seoul, Korea), pp. 953–964, 2006.

[9] D. Gyllstrom, E. Wu, H. Chae, Y. Diao, P. Stahlberg, and G. Anderson, “Sase: Complex event processing over streams,” in *3rd Conf. Innovative Data Systems Research*, (Asilomar, CA), 2007.

[10] G. Koch, B. Koldehofe, and K. Rothermel, “Quality-aware event correlation detection,” Tech. Rep. TR-2012-06, Univ. of Stuttgart, Germany, 2012.

[11] H. Kawashima, H. Kitagawa, and X. Li, “Complex event processing over uncertain data streams,” in *2010 Intl. Conf. Parallel, Grid, Cloud and Internet Computing*, (Fukuoka, Japan), pp. 521–526, 2010.

[12] Z. Shen, H. Kawashima, and H. Kitagawa, “Efficient probabilistic event stream processing with lineage and kleene-plus,” *Int. J. Commun. Netw. Distrib. Syst.*, vol. 2, no. 4, pp. 355–374, 2009.

[13] Z. Li, T. Ge, and C. X. Chen, “ε-matching: Event processing over noisy sequences in real time,” in *Intl. Conf. Management of Data*, (New York, NY), pp. 601–612, 2013.