The Event Crowd: A Novel Approach for Crowd-Enabled Event Processing

Piyush Yadav
Lero- The Irish Software Research Centre, National University of Ireland Galway
piyush.yadav@lero.ie

Umair ul Hassan
Insight Centre for Data Analytics, National University of Ireland Galway
umair.ulhassan@insight-centre.org

Souleiman Hasan
Lero- The Irish Software Research Centre, National University of Ireland Galway
souleiman.hasan@lero.ie

Edward Curry
Lero- The Irish Software Research Centre, National University of Ireland Galway
edward.curry@lero.ie

ABSTRACT

Event processing systems involve the processing of high volume and variety data which has inherent uncertainties like incomplete event streams, imprecise event recognition etc. With the emergence of crowdsourcing platforms, the performance of event processing systems can be enhanced by including 'human-in-the-loop' to leverage their cognitive ability. The resulting crowd-sourced event processing can cater to the problem of event uncertainty and veracity by using humans to verify the results.

This paper introduces the first hybrid crowd-enabled event processing engine. The paper proposes a list of five event crowd operators that are domain and language independent and can be used by any event processing framework. These operators encapsulate the complexities to deal with crowd workers and allow developers to define an event-crowd hybrid workflow. The operators are: Annotate, Rank, Verify, Rate, and Match. The paper presents a proof of concept of event crowd operators, schedulers, poolers, aggregators in an event processing system. The paper demonstrates the implementation of these operators and simulates the system with various performance metrics. The experimental evaluation shows that throughput of the system was 7.86 events per second with average latency of 7.16 seconds for 100 crowd workers. Finally, the paper concludes with avenues for future research in crowd-enabled event processing.

CCS CONCEPTS

• **Computing methodologies → Distributed computing methodologies • Information systems → Crowdsourcing**

KEYWORDS

Event crowd, event processing, crowdsourcing, human-in-the-loop, event uncertainty

DEBS '17, June 19-23, 2017, Barcelona, Spain

ACM ISBN 978-1-4503-5065-5/17/06…$15.00
http://dx.doi.org/10.1145/3093742.3093922

1 INTRODUCTION

We are living in a world driven by data. With the volume of data being generated constantly increasing, a number of challenges exist to process such data especially if data is streaming in real time [1]. For example, in environmental monitoring, the data is processed from various sensors to understand pollution levels so that immediate action can be taken to counter its negative effects. Similarly, other fields like financial [2], transportation [3], and manufacturing require the processing of data in near-real-time.

Event-based systems process data from different sources like sensors and social media feeds where the event data can be veracious and have inherent uncertainties. These uncertainties can be due to multiple reasons including incomplete event streams, erroneous event recognition and imprecise event patterns [4]. Ivo et al. [2] distinguish three types of uncertainties in event data i.e. uncertainty in event content, occurrence, and rules. For example, a scenario of crime detection requires a visual surveillance system [4] to detect people and their actions in different conditions. This is highly uncertain to predict accurate patterns of crime on monitoring people actions. Thus involving human computation can reduce uncertainty aspects in event-driven applications.

Human intelligence can be used as an intermediary to gather better insights from event data where needed. Crowdsourcing, which incorporates 'human computation' as a building block, is used in various data intensive applications for comparison, classification, verification, etc. It plays a vital role in addressing tasks like entity resolution, image identification, and others, where existing algorithms have limited capabilities. In crowdsourcing, tasks are segmented into smaller microtasks and are resolved using human inputs. With the development of crowd-powered platforms like Amazon Mechanical Turk (MTurk) [5] and CrowdFlower [6], human input is no longer constrained to offline batch processing. Realtime crowdsourcing is emerging where the response wait time is getting transitioning from hours [7] to seconds [8] thus opening the opportunity to develop crowd-enabled event processing systems.

Crowd operators like filtering [9], labeling, selection [10], join, and sort have been proposed to perform common database operations using human workers. Presently, no operators in event processing systems deal with crowd interactions. In this paper, we propose an initial set of five event crowd operators to process

events using a crowd of workers. These operators introduce 'human-in-the-loop' natively within event processing systems, to solve complex real world problems. The main contributions of the paper are the following:

1. Explore the use of crowdsourcing in event based systems and its benefits in solving event related problems like uncertainty, labeling, and verification.
2. A reference architecture for crowd-enabled event processing.
3. Event crowd operators: Annotate, Match, Rank, Rate, and Verify.
4. An initial evaluation framework for crowd-enabled event system along with performance metrics over simulated events and crowds.
5. Research challenges of crowdsourcing in event based systems.

The rest of the paper is organized as follows: Section 2 covers crowdsourcing basics and works related to operators in event processing and crowdsourcing. Section 3 gives detail on the reference architecture. Section 4 conceptualizes the proposed event crowd operator design with their structure and definition. Section 5 explains methodology, assumptions, and experimental evaluation. Section 6 discusses the research challenges and implications of this work. The paper concludes in section 7.

2 THEORETICAL BACKGROUND

This section conceptualizes the theoretical framework for the need of crowd operators in event processing. It introduces event processing languages and various key concepts and terminologies used in the crowdsourcing domain. Then it focuses on realtime crowdsourcing and related work. Finally, a motivational example is presented.

2.1 Event Processing Languages

Many event processing systems use declarative and SQL-like query languages, termed as Event Processing Language (EPL) [11]. These EPL queries once registered, run continuously on an event engine and return the desired results as defined by a subscriber's rule. EPL's have their own specific set of rules and operators which have their own domain associated semantics. For example, SpaTec [12] is a composite event language which uses operators like 'same location' and 'remote location' to match event occurrences over space and time. The syntax of these languages is intuitive and provide high-level abstraction, thus facilitating users and programmers to write reactive rules in a simplistic form which execute upon detection of specific events.

2.2 Basic Concepts in Crowdsourcing

The word "crowdsourcing" is made from the words "crowd" and "outsource" which means to redistribute or contract out work to potentially large groups of people. Thus, it is the redistribution of a problem on an online platform to get it resolved by tapping the collective human intelligence in exchange for some incentives. For example, object recognition in an image is a complex task. This can easily be solved by creating a bounding box around objects and getting it labeled with different people [13]. Access to crowd resources has become easier with the development of crowd-powered interfaces [5,6]. Below is the list of core concepts related to crowdsourcing:

1. **Requester:** The individuals or organizations who want their work to be completed. They post their tasks on a crowdsourcing platform with specific budget and deadline to get their work done.
2. **Worker:** The people who perform tasks on the crowd platform. They select available tasks as per their expertise and perform them and get paid, as it is the major driver which motivates them to work.
3. **Crowdsourcing Platform:** An interface which connects both workers and requesters and provides them with a high-level abstraction to facilitate task exchange. The platform behaves like a marketplace which handles all aspect of crowd work from task generation to completion, pricing, worker availability, etc. Presently, several crowdsourcing platforms are available like Amazon Mechanical Turk [5] and CrowdFlower [6].
4. **Task Design:** Designing a task is a key aspect of crowdsourcing. The requester divides complex problems into several simple tasks called HIT's (Human Intelligence Tasks), which can easily be resolved by ordinary workers. The tasks can be of varied types like single or multiple choices, rating, clustering or labeling. The requester also defines task settings like pricing, timing, and quality control as per his specific needs [14].
5. **Answer Aggregation:** To overcome the issue of potential low-quality answers from workers, it is common in crowdsourcing to collect multiple answers for the same task and then choose the optimal result through answer aggregation. Various aggregation algorithms have been proposed including majority decision [10] and expectation maximization [15].

2.3 Realtime Crowdsourcing

Recent works have shown that responses from the crowd can be gathered at an interactive speed with less crowd latency. CRQA [16], a crowd -powered near realtime automatic question answering system has the ability to answer questions in under 1 minute. VizWiz [17] is a mobile phone based talking application, that gives 'near real-time' answers to visual questions. It follows an intelligent approach- 'quikTurkit' for recruiting worker in advance which can produce an answer to a question in an average of 56 seconds. Similarly, Bernstein et al. [8] introduced the concept of 'on demand synchronous crowds' where workers are present at the same time to do a task. They presented the 'retainer model', a pre-recruitment strategy, where workers are paid in advance so that they can be available on demand to perform a

task. Thus the response time of crowd is transitioning from hours to seconds making it suitable for near realtime systems.

2.4 Related Work

The use of crowdsourcing for computational operators has been actively studied in the databases community [14]. Crowd operators are used to perform various operations on given tasks as per the defined rules. In general, the primary focus of database researchers has been to include the crowd into data operators like selection [10], filtering [9], sort, join and aggregation [18]. There have also been efforts to define declarative approaches for introducing crowd-based operations in data processing systems [19,20]. More recent research proposals have tried to crowdsource spatial tasks [21,22]. However, none of these works have considered the use of crowd-based operators in data processing for streams or events.

Enabling crowdsourcing in event-based systems can help to solve challenging problems like event uncertainties, data veracity, inaccurate measurements, and image annotation. Artikis et al. [3] showed the utility of crowdsourcing for event detection in complex event processing. The authors use a crowdsourcing module in urban traffic modeling to deal with data sparsity and sensor disagreement in heterogeneous stream processing. It reduces the event uncertainty through crowdsourced information thus giving more accurate information. But their proposal is primarily limited to applying event detection rules on streams of data. By comparison, this paper takes a more holistic approach to crowdsourcing event-based system by defining crowd-based event operators.

Wasserkrug et al. [23] propose a framework for uncertainty in event processing which categorizes uncertainty in two dimensions: element uncertainty and origin uncertainty. Element uncertainty deals with uncertainties about event occurrence and event attributes. In our model, such an uncertainty is the result of crowd performance of operators on events, as that can include uncertainty due to lack of full agreements between workers, under-performance, or limited expertise. Origin uncertainty may come from the event source or inference over events. Our model deals with another origin which is the crowd, as crowd-based single-event processing uncertainty can propagate to a complex event processing (CEP) pattern. Wasserkrug et al. [24] propose a model to deal with such a CEP inference under uncertainty.

2.5 A Motivating Example

The US Wildfire Activity Public Information map [25] shows the recent active locations of wildfires and other information related to it. The map is generated using live feeds from US wildfire reports, MODIS satellite hotspots, weather radar and social media feeds like YouTube, Twitter, Instagram and Flickr. The map consists of inherent uncertainties and false information some of which are listed below:

- Fig. 2 shows a MODIS satellite hotspot alert stating that a 1 km zone of that location is identified 'hot' by sensors and can be a wildfire. How to quickly verify that the potential hotspot is a wildfire so that alerts can be raised to firefighting agencies?

- Fig. 3 shows a YouTube video on the wildfire map with a label of "This is Wildfire. Signing off". But on analysis, it was found that the video is related to gaming console. How can this wrong information be filtered out to avoid false alert?

- Fig. 4 shows a Flickr image talking about wildfire smoke, but the smoke might be a low cloud. Thus, how can we filter out the false information and optimize our streams?

Consider a smart city scenario where the city administrator has subscribed to an event engine for wildfire alerts as it includes threats to the city's infrastructure like water reservoir and power supply. The wildfire related streaming data is handled by an event processing engine. Although the event engine will filter out unnecessary information but then also the uncertainties will be there due to its inherent limitation. How can the event processing engine deal with the uncertainties described above? The process can become efficient if there are some crowd operators configured with the event engine to loop in human computation to verify the incoming events. As shown in Fig. 1, a crowd-enabled event processing engine posts streaming events to a crowd sourcing platform and then can collect aggregated response to generate results thus acting as a black box between event sources, sinks, and the crowdsourcing platform. Number in Fig. 1 shows the flow of information in the system.

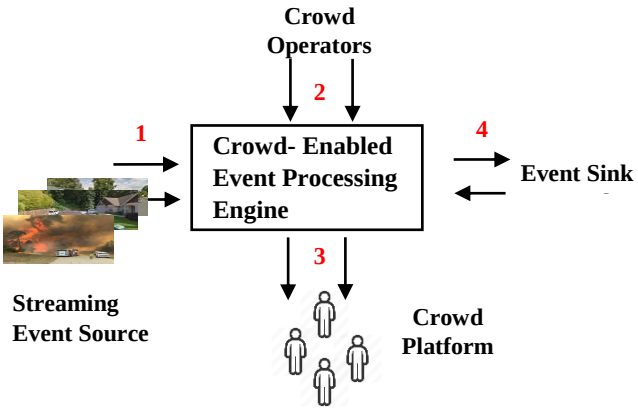

Figure 1: Event Crowd reference architecture

3 REFERENCE ARCHITECTURE

This section describes the reference architecture where crowdsourcing can be supported in event processing systems using the proposed event crowd operators. Our general framework is depicted in Fig. 5 which shows its various components. The detailed description of these components with architectural flow are as follows:

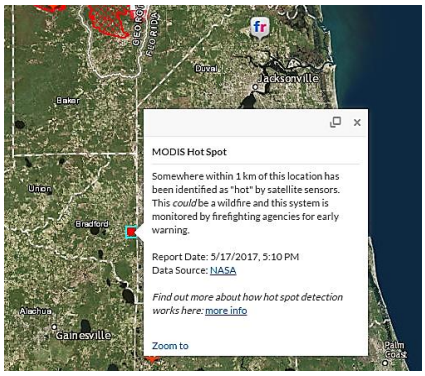

Figure 2: MODIS satellite hot spots warning

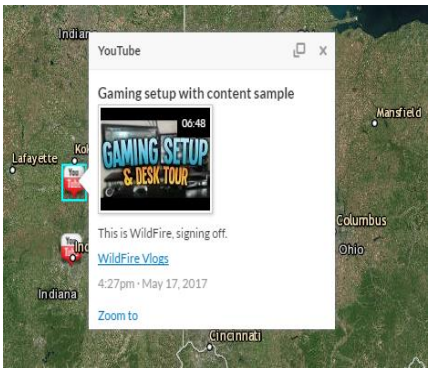

Figure 3: False YouTube video of wildfire alert

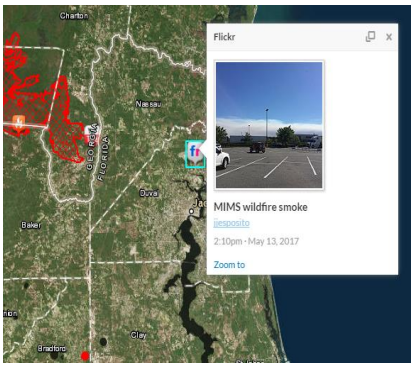

Figure 4: False Flickr post on wildfire smoke

1. The *Event Source* streams the events to be processed by the *Event Engine*. The source can be any publishing system like databases, sensors, Web feeds, and Web services. *Event Sinks* are the final destination and act as subscribers like applications, databases, dashboards, and agents.
2. The *Event Engine* receives the events of interest and processes them using the event crowd operators in the EPL query. When a crowd operator is applied over incoming events it wraps that event with crowd specifications and creates a new event termed as the *crowd event.* The *Event Engine* then sends the *crowd event* to the *HIT Engine.*
3. In crowdsourcing, Human Intelligence Tasks (HIT) are the tasks (here events with operators and crowd specifications) which the crowd/worker performs. Thus, the *HIT Engine* handles the crowd processing part and sends back the results to the *Event Engine.* This consists of three modules which are:
   - *HIT Manager:* The *HIT Manager* receives the *crowd events,* compiles them in a HTML form (HIT) as accepted by the crowd platform (e.g. MTurk) with different crowd specifications as provided by the proposed operators.
   - *HIT Scheduler:* The *HIT Scheduler* receives the HIT and sends it to the crowdsourcing platform using the application programming interface (API) of the crowd-platform. It also receives the results back from the crowd and sends it to the *HIT Aggregator.*
   - *HIT Aggregator:* This module aggregates the tasks answered by multiple workers on the same events and sends it back to the *Event Engine* as *aggregated event.* Thus the *aggregated event* itself are treated as a new event for the *Event Engine* creating a high level of abstraction for human computation.

The above architectural flow can be explained using an example. Consider an Event Source is streaming a set of social media images as events which need to get verified as wildfire or non-wildfire instances. The events will be received by the Event Engine which will further process it using the Verify crowd operator. The operator wraps the event as a crowd event with labeling specifications and sends it to the HIT Engine. The engine

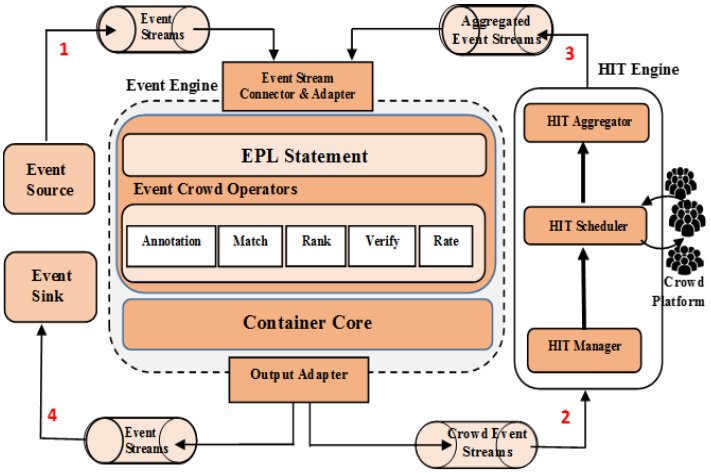

Figure 5: Crowd-enabled event processing architecture

will combine it as a HIT (Fig. 6) and send it to the crowd and receive the response back. The HIT Aggregator will aggregate the received answers as per the aggregation algorithms [26] and send the aggregated events back to Event Engine which can send it to the designated subscriber.

The above described architecture follows the push model of crowdsourcing where the tasks are pushed to the workers. Suppose a certain worker has information about events like wildfire, traffic congestion or accidents, then they can push the information to the event engine where it will be considered as an

Figure 6: Event posted as a HIT on crowdsourcing platform

event source. The crowd operators working in event engine will push the event streams to crowd workers to get the intended work done.

4 EVENT CROWD OPERATORS

This section conceptualizes five event crowd operators as shown in Table 1.

Table 1: Event Crowd Operators

Event Operator	Input	Output
Annotate	1 Event	Label
Match	2 Events	True/False
Rank	Collection of Events	Ordered List
Rate	1 Event	Score (1-5)
Verify	1 Event	True/False

In this paper, we introduce five event crowd operators for common tasks usually performed by the crowd which includes solving problems like labeling, translation, verification, and ranking. This list of operators is not comprehensive and can be extended. Each operator's functionality is explained with the help of formal semantics [27, 28] and user-defined functions (UDF) [29].

As shown in Fig. 7, when an operator is applied over incoming events it creates crowd events which have attributes including crowd tasks, crowd instructions, and crowd configuration. The crowd operator is defined using a UDF whose skeleton is shown in Table 2 with explanation of its attributes. In the below table, the operator function takes two types of input. Event Input is related to events and its specifications while Crowd Input takes the parameters related to crowdsourcing. These parameters are then assigned to the UDF attributes like crowd tasks, instructions, and configuration.

Table 2: Skeleton of User Defined Function for Event Crowd Operators

Operator Function (Event Input, Crowd Input)
Returns: Output

Crowd Task: Information regarding what operations (rank, verify, etc.) the crowd/worker has to perform over the event.

Crowd Instructions: Instructions the crowd needs to follow during operations.

Crowd Configuration: Crowdsourcing information: expiration time (*HIT exp):* the maximum time allowed to perform the operation before it will get expired, and the incentive (*HIT price):* the monetary cost that the crowd will be paid on performing operations over the event.

Response: Form where users will provide their answers as an output which will be send to *Returns.*

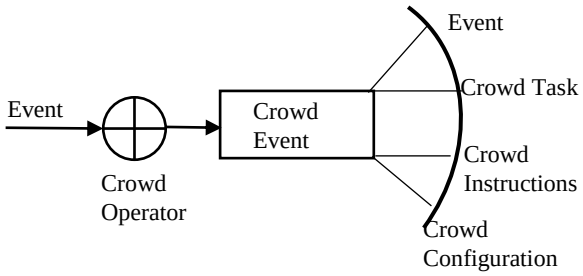

Figure 7: Event Crowd operator's design

4.1 Annotate

The annotate operator is used for labeling events like textual or image. The formal semantics of this operator is as follows:

$$\boldsymbol{Annotate(E) \coloneqq l_i \ if \ \exists l_i \ \in labels \ where \ labels = \{l_1, l_2, l_3 \ldots\ldots, l_n\}}$$

The above semantics represents that if the *Annotate* operator is applied over an event E, then it will return a primitive label l_i which belongs to the provided set- $labels$. The UDF of the annotate operator is given below:

Annotate (Object event, String [] label list, Object Crowd Input)
Returns: (String event label)

Crowd Task: Label event

Crowd Instructions: "Label the event from a given label list"

Crowd Configuration: ({*HIT exp*: 25 sec}, {*HIT price*: 0.25$})

Response: Form ((label 1', event label) … (label n', event label))

The function can be explained using the wildfire alert example. The Annotate function will take the image event as input with label list like (Wildfire, Normal Fire, and Can't Say). The function will post this as a HIT using the crowd configuration information. The Response will create a form where the crowd needs to provide the answer from predefined label list and returns the answer label.

4.2 Match

The Match operator is used to determine whether two events are the same or not. It sends the pair of events to the crowd which validates the similarity of events and sends a Boolean response {True, False} back to the event engine. Suppose there are two event instances E1 and E2:

E1:= $e1^1\ e1^2\ e1^3\ e1^4$ …….$e1^n$

E2:= $e2^1\ e2^2\ e2^3\ e2^4$ …….$e2^n$

The formal semantics of the Match operator is given below:

$$\boldsymbol{Match(E1, E2) \coloneqq True \ if \ \forall e1 \in E1 \ \Lambda \ \forall e2 \in E2 \ are \ crowd \ similar}$$

$$Match(E1, E2) \coloneqq False\ if\ \exists e1 \in E1\ \Lambda\ \exists e2 \in E2\ are\ crowd\ distinct$$

In the above semantics if all the instances of E1 and E2 are similar then it is matched and the query result is 'True' else if any of instance of E1 and E2 are not the same it will be 'False'. The Match UDF defined below takes two events and a crowd specification as input and creates a HIT with response of 'True' or 'False' and returns a Boolean match status. In the UDF under crowd instructions, the property 'P' refers to what characteristics event are going to get matched. For example, if a crowd is asked to match whether two events are from the same location then the property (P) is- location.

Match (Object event1, Object event2, Object Crowd Input)
Returns: (Boolean match status)
Crowd Task: Match events
Crowd Instructions: "Match whether the two events are same as per property P"
Crowd Configuration: ({*HIT exp*: 25 sec},{*HIT price:* 0.25$})
Response: Form((True', match status),('False', match status))

4.3 Rank

The Rank operator ranks a collection of events in an ordered list on the basis of some defined criteria. For example, if there are 5 events and they need to be ordered according to properties like value, importance or priority then the Rank operator will take these 5 events as an input and rank them. The semantics for the Rank operator is as follows:

$$Rank(E_1, E_2, E_3 \ldots\ldots E_n) \coloneqq (E_1; E_2; E_3 \ldots\ldots E_n)$$
$$if\ \forall i,j\ \Rightarrow\ val(E_i) \geq\ val(E_j)\ where\ i \leq j\ and\ i,j \in \{1, 2, 3, 4 \ldots\ldots n\}$$

In the above semantics, the operator takes a collection of events as an input and arranges them in a sequence which is denoted by a sequence operator **(;)** [28] . It denotes that if the value **(*val*)** of all instances of any event E_i is greater than the value of any other events E_j then it will be placed higher in the sequence.

Rank (Object [ ] event, Integer rank range, Object Crowd Input)
Returns: (Object [ ] ranked events)
Crowd Task: Rank events
Crowd Instructions: "Rank the list of events as per their property P"
Crowd Configuration: ({*HIT exp*: 25 sec},{*HIT price*: 0.25$})
Response: Form((Rank no.', event[1]),
.
.
(Rank no.', event[n]))

The above rank function takes a collection of events, rank range and crowd specification as an input. Here rank range means that the rank will be given in the provided range. In response, the crowd will get the list of events and needs to provide a rank within the specified rank range. For example, the collection of image events needs to be ranked on the basis of their quality like high resolution, blurred, and out of focus. If rank range is 5 then the image events will be ranked between 1 to 5 i.e. high quality and sharp resolution images will get the higher rank like 5 or 4 and the rank will decrease based on the quality of images.

4.4 Rate

The Rate operator rates the event on the basis of a specified property. Suppose there is a stream of events related to different restaurants of a given location. The Rate operator will rate these restaurants on the basis of a specific defined property like cuisine, service, etc., using crowdsourcing. The formal semantics of the Rate operator is given below:

$$Rate(E) \coloneqq crowdrate\ x\ where\ crowdrate\ x \in X\ and\ X_{min} \leq x \leq X_{max}$$

In the above semantics, the operator rates an event E with the specific value($crowdrate\ x$) where it belongs to the range of property X. Here X is the range in which a rating can be given like in the above example scenario it can be high, medium and low. As shown below, the UDF of the Rate function takes an event with various rate specifications and posts it to the crowd to get the event rated.

Rate (Object Event, String rate specification , Object Crowd Input)
Returns: (String, Integer rate)
Crowd Task: Rate event
Crowd Instructions: "Rate the event from the given rate specification"
Crowd Configuration: ({*HIT exp*: 25 sec},{*HIT price*: 0.25$})
Response: Form((rate spec 1', rate), ('rate spec 2', rate), ……, ('rate spec n', rate))

4.5 Verify

In [3] crowdsourcing has been used for handling event uncertainty in traffic modeling. Thus the Verify operator can leverage this functionality of verifying events through human computation. In the below semantics, when the Verify operator is applied over an event (E) it returns a Boolean response in terms of '*True'* or '*False'*, verifying the specified nature of event.

$$Verify(E) \coloneqq True/False\ if\ property\ of\ E\ is\ True/False\ according\ to\ the\ Crowd$$

In the below defined UDF, the Verify operator takes the event and the specifications as an input. The Verify specification gives instructions to the crowd based on what they need to verify in the event. For example, if there is streaming data from social media to verify whether there is traffic congestion in particular location or

not. The Verify operator can post the event to the crowd to get the responses and send it back to the event engine which can take further decisions on rerouting traffic for instance.

Verify (Object Event, String verify specification , Object Crowd Input)
Returns: (Boolean verification status)
Crowd Task: Verify events
Crowd Instructions: "Verify the content of the events"
Crowd Configuration: ({HIT exp: 25 sec},{HIT price: 0.25$})
Response: Form((‘True’, verification status), (‘False’, verification status))

4.6 Crowd Operators Implementation

We have implemented these operators in Esper [30] to run our experiments. It is a component for complex event processing written in Java. EPL queries can be easily written in Esper engine which can process large volumes of events form historical or real time scenario. Esper provides highly flexible extension API's from which the engine functionality can be extended by integrating new functionality.
The event crowd operators have been implemented using the Esper extensions API. The operators have been written as a Java class file and then are integrated with the engine. The operators can then be directly used within EPL queries. A simple match operator query can be written as:

```
Select Match(event1, event2, crowd input )
from Image_Event.win:length(1)
```

The above query will take two Image events with crowd input specifications and will create a new *crowd event* which will be posted on the crowd platform. Since here the answer will come in ‘*True’* (image matched) or ‘*False’* (image not matched) so the operator wraps the resulting event with these Boolean options. Similarly, a simple Annotate query can be written as:

```
Select Annotate(event, label list, crowd
input) from Image_Event.win:length(2)
```

The above query will create a new crowd event having a list of labels. The crowd will select labels from this list and annotate the event. The operators design is independent and can easily be written in any event processing framework. Overall we want to be language agnostic in defining the operator for their ease of use in any other EPL. The implementation in Esper is done as proof of concept for event crowd idea.

5 EXPERIMENTAL EVALUATION

Our experiment has two main goals: 1) Assess the average latency and throughput of human-in-the-loop in event processing using crowd operators. 2) Assess the crowd operators and HIT Aggregator latency.

5.1 Methodology

We performed our simulation using an Intel core i7 machine with 2.60 GHz CPU and 8 GB of RAM. The events were generated using Poisson distribution with different average arrival rates ($\boldsymbol{\lambda}$). We simulated the experiments with different arrival rates ranging from 1 to 100 events per second. The Esper engine receives these events and event crowd operators wraps these events as per rules and crowdsourcing information and push it to *HIT Scheduler* queue. This queue stores the events in first in first out (FIFO) order. The crowd simulator receives the events from queue and performs the tasks. We have followed the retainer model [8] for realtime crowdsourcing. This is a recruitment approach where the crowd is pre hired (with some extra cost) to work on specific tasks and will be available when tasks arrived to them. We have used queuing theory [31] to retain the worker pools. Suppose the worker pool size is W_0, as the specific tasks comes the W_k workers starts working on the tasks with worker pool size remaining to W_0- W_k. If the overall worker pool size is zero (all workers busy) then it will not accept any tasks until some sets of workers get free to take job. In short it is a M/M/c/N queue [31], which has a ‘c’ parallel servers with N buffer size where tasks(events) arrives at rate $\boldsymbol{\lambda}$ and have processing time µ. The probability that an event has to wait (when all ‘c’ servers are busy) to get processed can be determined by using Erlang's Loss formula [31].

5.2 Results

We ran our experiments for 4000 events at different arrival rates ranging from 1 to 100 events per seconds. Events are generated from the source according to a Poisson process with a specified rate. Fig. 8 shows the graph between throughput and average latency of our system. Average latency is the average time taken by each events to get processed by the system. This includes the time from when an event is generated, crowd operators applied on it, queuing and dequeuing time in the *HIT Scheduler*, time taken by the crowd to process the events, and the time taken by HIT Aggregator to aggregate the responses for each event. Throughput is considered as the number of events being processed by the system in every second. In our experiment (Fig. 8) we have taken different worker pool size ranging 40 to 100 workers where, when an event arrives will be served by set of crowds. Bernstein et al. [8] have shown that the minimum response time by the crowd to get an answer is approximately 10 seconds. We have used this response time in our simulation as this can give us a minimum threshold time to get answers in event processing systems using crowd operators. It can be seen that the system throughput increases with an increase in the number of workers. Initially, for 40 workers the throughput is 3.61 events per second which increased to 7.86 events per second for 100 workers. There is little change in the throughput after 70 workers. The average latency for events is 246.47 seconds for 40 workers which decreases with increase in throughput. The average latency for events is 7.16 seconds for 100 workers. Thus, from the graph we can say that system is sustainable with no backpressure of events when 100

workers are pooled for events coming at arrival rate less that 7.16 seconds.

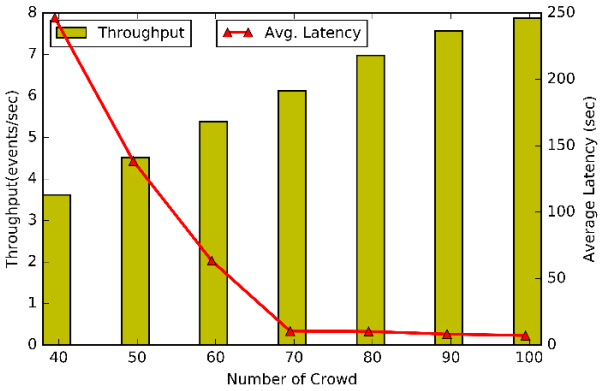

Figure 8: Throughput vs. Latency

Fig. 9 shows the computation time of five event crowd operators. In order to understand the real operator processing time we passed the events to these operators under a 'for' loop. Thus the arrival rate of events is equal to the system processing speed. We have averaged 1000 runs to calculate the computation time of operators. Fig. 9 shows the operator's time to process 500 events. Since the Rank operator processes a collection of events, its computation time is little higher (1349 milliseconds/500 events) as compared to other operators. Similarly, the Match operator's computation time (1,337 milliseconds/500 events) is second highest as it takes two events as input for processing.

In crowdsourcing platform, each event is answered by specific set of workers. Thus, each event has multiple responses which need to be aggregated to get the final answers. There are multiple aggregation algorithms to get the final answers based on workers quality. We have integrated the simulator given in [26] to test our system. The simulator run over specified number of questions and apply different aggregation algorithms. We have used four aggregation algorithms to determine the performance which is been used by the *HIT Aggregator*. Fig. 10 shows the computation time for aggregating answers per events. The simulation has been run for 4,000 events responded by 100 workers where each event has responses ranging from 1 to 5.

It can be seen that the Majority Decision [10] and GLAD [32] approaches have the least aggregation time and are nearly constant with different answers per events while the Expected Maximization (EM) [15] and SLME [33] computation time increases with increase in number of answers per events. The EM and SLME take more time because in every iteration they update the aggregated value of answers on the basis of worker expertise and adjust the worker expertise as per there response.

The above experimental evaluation are preliminary to test the event crowd concept. There is no present competitor or system against which we can compare or benchmark our results due to the immaturity of the field. The evaluation shown gives an indication of performance under certain assumptions which will vary across different applications.

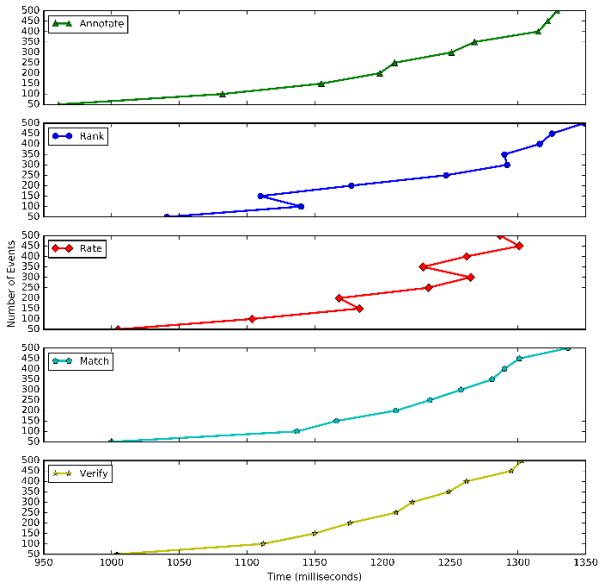

Figure 9: Event Crowd operator's computation time

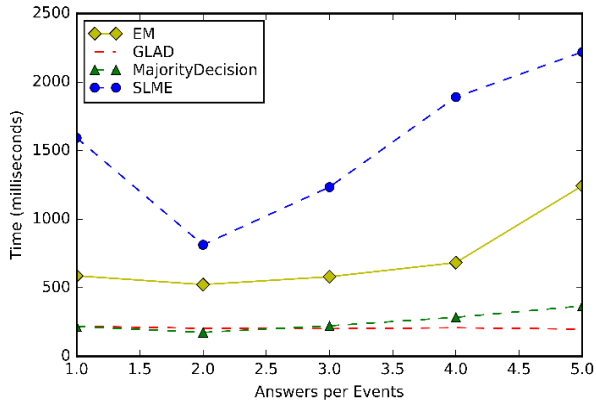

Figure 10: Aggregation of events with different aggregation algorithms

5.3 Limitations and Assumptions

Our simulated experiments have a number of assumptions:

1. The *HIT Scheduler* queue size is large so that it can add up the incoming events. The queue size is dynamic and can grow up to its buffer limit so that no events can be lost. Thus there is no throttling of events.
2. In event processing the arrival rates of events can be millions per second but due to limited experimental setup we have limited our arrival rates to a maximum of 100 events/second.
3. In real crowdsourcing, the workers have different expertise levels like normal, expert, spammers, etc. We have assumed that all the workers have the same quality. In the experiments worker quality is considered on the basis of their response time and not on the basis of their expertise.

4. The fastest crowd response of 10sec is taken from literature [8].
5. We have not considered any expiration time of event and assumed that all events are being answered by the workers.
6. In order to get better quality answers some known tasks which are termed as gold units are used to find better quality answers. In our experiments no gold units are injected in streaming events.
7. Aggregation of answers for each event is based on Majority Decision [10] algorithm.
8. No variation of pricing is considered for the crowd.

6 CHALLENGES AND IMPLICATIONS

The inclusion of crowd operators into event processing languages is sort of an event enrichment process [34] which has number of research challenges associated:

1. *CEP Pattern Matching:* One research challenge lies in the extension of the model proposed in this paper to pattern matching and complex event processing. Research questions include how crowd operators in different parts of a pattern are scheduled to be submitted to the HIT server, and how the uncertainty of single-events operators are propagated to evaluate the uncertainty scores of patterns and the derived CEP events.
2. *Optimization of HIT Scheduling*: The proposed model poses challenges on how to optimize the management and update of the cache of crowd responses. Research questions also arise on the potential subsumption relationships that can exist between crowd operators which can lead to opportunities to reduce latency.
3. *Crowd Routing and Real-time Availability:* One of the primary research challenges of crowdsourcing in CEP is to meet the varying latency requirements of processing human intelligence tasks. Specifically, overcoming the differences between near real-time processing of events and variance in availability of crowds. This problem poses a quality versus latency trade-off. The quality of crowd answers can also be affected by the expertise of crowd workers, which becomes more apparent when HIT's require domain specific knowledge.
4. *Scaling with Machine Learning:* Another challenge of crowdsourcing in CEP is the scaling of the proposed approach in case of a large number of parallel events. As crowdsourcing becomes more popular, applications are competing for human attention on crowdsourcing platforms. In this respect, it is interesting to investigate the use of machine learning for approximate crowd answers or routing HIT's to appropriate workers.
5. *Realtime Crowdsourcing:* Latency is the biggest bottleneck for event based systems. Bernstein et al. [8] introduce the concept of realtime crowdsourcing, where pre-recruited workers are present for doing certain tasks. But pre-recruiting workers itself is a challenge from the perspective of availability, no. of workers needed to be retained, extra incentives, and to keep them standby until the task is assigned, are all an open areas of research.
6. *Other Event Crowd Operators:* The paper details five event crowd operators which can be extended further depending on specific application scenarios. There is still a challenge to identify other event crowd operators. The operators list can be related to incentives, evaluate quality of experience, and categorization. It is also interesting to extend the design of present crowd operators by adding extra attributes like quality of service.
7. *Geographic Density of Workers:* In crowdsourcing, for spatial crowd operators the geographic density of workers is also essential. As shown in Fig. 2, some tasks requires people in the near vicinity to verify the information related to events. In case of low density workers, the task can then be assigned to multiple people in near vicinity [35] to ensure its completion which is itself an area of research for task assignment.

7 CONCLUSION AND FUTURE WORK

In this paper, we proposed five crowd operators for event processing. The aim of operators is to bring 'human in the loop' in event systems. We discuss the design of each operator using formal semantics and user-defined functions. The working model of operators has been implemented in Esper. The paper details a reference architecture for event systems using an event engine, crowd operators, HIT manager, and a crowdsourcing platform. Finally, the paper discusses the experimental evaluation for the system by calculating throughput and average latency. The experimental result shows that the throughput of the system increases with the increase in worker pool size and is associated with a decrease in the average latency. The system throughput for 100 workers was 7.86 events per second with average latency of 7.16 seconds for each events. The computation time for Rank and Match operator is relatively higher than other operators as they take more input events for processing. The fusion of crowdsourcing and event processing poses a number of new research challenges and implication. We plan in the future to scale our system with real crowdsourcing platform to benchmark our results in real-world settings.

ACKNOWLEDGEMENTS

This work was supported with the financial support of the Science Foundation Ireland grant 13/RC/2094 and co-funded under the European Regional Development Fund through the Southern & Eastern Regional Operational Programme to Lero - the Irish Software Research Centre (www.lero.ie).

REFERENCES

[1] T. Becker, E. Curry, A. Jentzsch, and W. Palmetshofer. 2016. New Horizons for a Data-Driven Economy: Roadmaps and Action Plans for Technology, Businesses, Policy, and Society. In *New Horizons for a Data-Driven Economy* (pp. 277-291). Springer International Publishing.

[2] I. Correia, F. Fournier, and I. Skarbovsky. 2015. The uncertain case of credit card fraud detection. *In Proceedings of the 9th ACM International Conference on Distributed Event-Based Systems* (pp. 181-192). ACM.

[3] A. Artikis, M. Weidlich, F. Schnitzler, I. Boutsis, T. Liebig, N. Piatkowski, C.

Bockermann, K. Morik, V. Kalogeraki, J. Marecek, and A. Gal. 2014. Heterogeneous Stream Processing and Crowdsourcing for Urban Traffic Management. In *EDBT* (Vol. 14, pp. 712-723).

[4] A. Artikis, O. Etzion, Z. Feldman, and F. Fournier. 2012. Event processing under uncertainty. In *Proceedings of the 6th ACM International Conference on Distributed Event-Based Systems* (pp. 32-43). ACM.

[5] Amazon Mechanical Turk. Available from: https://www.mturk.com/.

[6] CrowdFlower: AI for your business. Available from: https://www.crowdflower.com/.

[7] P.G. Ipeirotis. 2010. Analyzing the amazon mechanical turk marketplace. XRDS: Crossroads, The *ACM Magazine for Students*, 17(2), pp.16-21.

[8] M.S. Bernstein, J. Brandt, R.C. Miller, and D.R. Karger. 2011. Crowds in two seconds: Enabling realtime crowd-powered interfaces. In *Proceedings of the 24th annual ACM symposium on User interface software and technology* (pp. 33-42). ACM.

[9] A.G. Parameswaran, H. Garcia-Molina, H. Park, N. Polyzotis, A. Ramesh, and J. Widom. 2012. Crowdscreen: Algorithms for filtering data with humans. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data* (pp. 361-372). ACM.

[10] T. Yan, V. Kumar, and D. Ganesan. 2010. Crowdsearch: exploiting crowds for accurate real-time image search on mobile phones. *In Proceedings of the 8th international conference on Mobile systems, applications, and services* (pp. 77-90). ACM.

[11] F. Bry, M. Eckert, O. Etzion, J. Riecke, and A. Paschke. 2009. Event processing language tutorial. In *Proceedings of the 3rd ACM Int. Conf. on Distributed Event-Based Systems. ACM.*

[12] S. Schwiderski-Grosche and K. Moody. 2009. The SpaTeC composite event language for spatio-temporal reasoning in mobile systems. In *Proceedings of the Third ACM International Conference on Distributed Event-Based Systems* (p. 11). ACM.

[13] H. Su, J. Deng, and L. Fei-Fei. 2012. Crowdsourcing annotations for visual object detection. In *Workshops at the Twenty-Sixth AAAI Conference on Artificial Intelligence* (Vol. 1, No. 2).

[14] G. Li, J. Wang, Y. Zheng, and M.J. Franklin. 2016. Crowdsourced data management: A survey. *IEEE Transactions on Knowledge and Data Engineering*, 28(9), pp.2296-2319.

[15] P.G. Ipeirotis, F. Provost, and J. Wang. 2010. Quality management on amazon mechanical turk. In *Proceedings of the ACM SIGKDD workshop on human computation*. ACM.

[16] D. Savenkov and E. Agichtein. 2016. CRQA: Crowd-Powered Real-Time Automatic Question Answering System. *In Fourth AAAI Conference on Human Computation and Crowdsourcing.*

[17] J.P. Bigham, C. Jayant, H. Ji, G. Little, A. Miller, R.C. Miller, R. Miller, A. Tatarowicz, B. White, S. White, and T. Yeh. 2010. VizWiz: nearly real-time answers to visual questions. *In Proceedings of the 23nd annual ACM symposium on User interface software and technology* (pp. 333-342). ACM.

[18] A. Marcus, D. Karger, S. Madden, R. Miller, and S. Oh. 2012. Counting with the crowd. In *Proceedings of the VLDB Endowment* (Vol. 6, No. 2, pp. 109-120). VLDB Endowment.

[19] A.G. Parameswaran, H. Park, H. Garcia-Molina, N. Polyzotis, and J. Widom. 2012. Deco: declarative crowdsourcing. In *Proceedings of the 21st ACM international conference on Information and knowledge management* (pp. 1203-1212). ACM.

[20] M.J. Franklin, D. Kossmann, T. Kraska, S. Ramesh, and R. Xin. 2011. CrowdDB: answering queries with crowdsourcing. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data* (pp. 61-72). ACM.

[21] H. To, G. Ghinita, and C. Shahabi. 2014. A framework for protecting worker location privacy in spatial crowdsourcing. Proceedings of the VLDB Endowment, 7(10): p. 919-930.

[22] U. ul Hassan, and E. Curry. 2016. Efficient task assignment for spatial crowdsourcing: A combinatorial fractional optimization approach with semi-bandit learning. Expert Systems with Applications. 58: p. 36-56

[23] S. Wasserkrug, A. Gal, and O. Etzion. 2006. A taxonomy and representation of sources of uncertainty in active systems. In *International Workshop on Next Generation Information Technologies and Systems*. Springer.

[24] S. Wasserkrug, A. Gal, O. Etzion, and Y. Turchin. 2008. Complex event processing over uncertain data. In *Proceedings of the second international conference on Distributed event-based systems DEBS 08*, pp. 253–264.

[25] ESRI. US Wildfire Activity Public Information Map. [Online] ESRI. 2017. https://www.arcgis.com/apps/PublicInformation/index.html?appid=4ae7c683b9574856a3d3b7f75162b3f4.

[26] N. Quoc Viet Hung, N. T. Tam, L. N. Tran, and K. Aberer. 2013. An evaluation of aggregation techniques in crowdsourcing. In Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), vol. 8181 LNCS, no. PART 2, pp. 1–15.

[27] G. Cugola and A. Margara. 2010. TESLA: a formally defined event specification language. In *Proceedings of the fourth international conference on Distributed event-based systems DEBS*, pp. 50–61.

[28] D. Zimmer and R. Unland. 1999. On the semantics of complex events in active database management systems. In *Proceedings 15th International Conference on Data Engineering* (Cat. No.99CB36337), 1999, no. Dml, pp. 392–399.

[29] A. Marcus, E. Wu, and D. Karger. 2011. Demonstration of qurk: a query processor for humanoperators. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*. ACM.

[30] EsperTech. EsperTech: Event Series Intelligence. Available from: http://www.espertech.com/esper/.

[31] D. Gross, J. Shortle, F. Thompson, and C. Harris. 2008. Fundamentals of queueing theory. 2008: John Wiley & Sons.

[32] J. Whitehill, P. Ruvolo, T. Wu, J. Bergsma, and J. Movellan. 2009. Whose Vote Should Count More: Optimal Integration of Labels from Labelers of Unknown Expertise. In *Proceedings of Advances in Neural Information Processing System*, vol. 22, no. 1, pp. 1–9, 2009.

[33] V.C. Raykar, S. Yu, L.H. Zhao, A. Jerebko, C. Florin, G.H. Valadez, L. Bogoni, and L. Moy. 2009. Supervised learning from multiple experts: whom to trust when everyone lies a bit. In *Proceedings of the 26th Annual international conference on machine learning* (pp. 889-896). ACM.

[34] S. Hasan, S. O'Riain, and E. Curry. 2013. Towards Unified and Native Enrichment in Event Processing Systems. In Proceedings of 7th *international conference on Distributed event-based systems DEBS (DEBS 2013)*, pp. 171–182, 2013.

[35] U. ul Hassan, and E. Curry. 2015. Flag-Verify-Fix: Adaptive Spatial Crowdsourcing leveraging Location-based Social Networks. In *Proceedings of 23rd ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems.* (ACM SIGSPATIAL 2015), pp. 1–4, 2015.