Future Generation Computer Systems 71 (2017) 202–211

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

Exploiting user feedback for online filtering in event-based systems☆

Fabio Petroni [a], Leonardo Querzoni [a,*], Roberto Beraldi [a], Mario Paolucci [b]

[a] Department of Computer Control and Management Engineering Antonio Ruberti, Sapienza University of Rome, Italy

[b] Institute of Cognitive Sciences and Technologies, CNR, Italy

ARTICLE INFO

Article history:
Received 29 April 2016
Received in revised form
10 October 2016
Accepted 13 October 2016
Available online 26 October 2016

Keywords:
Event-based systems
Recommendation systems
Content filtering
Distributed systems

ABSTRACT

Modern large-scale internet applications represent today a fundamental source of information for millions of users. The larger is the user base, the more difficult it is to control the quality of data that is spread from producers to consumers. This can easily hamper the usability of such systems as the amount of low quality data received by consumers grows uncontrolled. In this paper we propose a novel solution to automatically filter new data injected in event-based systems with the aim of delivering only content consumers are actually interested in. Filtering is executed by profiling producers and consumers, and matching their profiles as new data is produced. Profiles are built by aggregating feedback submitted by consumers on previously received data.

1. Introduction

In the recent years the world of global information has been dramatically changed by the widespread participation of people to social networks, whose user base continue to grow, as of today, at an astonishing pace. In such systems users can freely exchange data in several forms: status updates, tweets, links, comments, etc. Users select data they are willing to receive using coarse grained selection methods like friend/follower lists, tag selection or free text searches; then the social network provides each user with a personalized data stream whose content takes into account user selection criteria, but also includes other data that is considered to be potentially interesting for the user or that simply is the output of some advertising campaign. Users are sometimes given the possibility to provide feedback on data they receive (e.g. "I like" on Facebook or re-tweets on Twitter), but this feedback is commonly taken into account only to increase the amount of data injected in the stream or to suggest users new selections and connections. This coarseness in the available selection criteria, mixed with the current data growth rate, starts to limit their effectiveness in conveying useful information to users: as user streams grow to unmanageable sizes they tend to contain more and more information whose perceived quality is low from the user standpoint. This problem, if not adequately addressed, could hamper the usability of these systems. An ideal system should be able to filter-out at run-time from each user stream all those data that the user would personally rate as "low quality", only delivering data that matches the user selection criteria and satisfies some personal quality constraints. Current solutions in the state of the art related to *reputation systems* or *collaborative filtering* cannot be easily adapted to solve this online filtering problem as they either lack generality, or are inherently centralized.

In this paper we propose a novel online filtering solution for event-based systems (social networks are, in fact, moving from batch data mining to on-the-fly event processing [1]). Our solution leverages feedback expressed by users on received events to profile data sources. Profiling is based on a reputation metric calculated as a result of a collaborative process performed among all destinations that received and evaluated previous events injected by a same source. The profiling process takes into account that a source can produce events with different quality levels on different topics. New events injected in the system are filtered online depending both on the profile associated with their sources and on profiles built for characterizing the minimum quality thresholds of each potential destination. A destination profile is built considering all the feedback each user expressed on previously delivered events.

In social networks like Facebook or Twitter, such a solution could be adopted to automatically filter incoming updates from the user timeline or tweet stream, allowing him to focus only on updates that shall, with high probability, capture his interest. A

☆ A preliminary version of this work appeared at the 31st ACM Symposium on Applied Computing.

* Corresponding author.
E-mail addresses: petroni@dis.uniroma1.it (F. Petroni), querzoni@dis.uniroma1.it (L. Querzoni), beraldi@dis.uniroma1.it (R. Beraldi), mario.paolucci@istc.cnr.it (M. Paolucci).

http://dx.doi.org/10.1016/j.future.2016.10.017
0167-739X/

different example could be represented by social news aggregation services, like *Feedly*, that aggregate syndicated content from multiple RSS feeds and propose it to their users; users could express their interest in a news by signaling it via Feedly's "add to knowledge board" functionality, or by simply sharing the content with other users; lack of interest in a news could be identified at browser level by tracking the amount of time each stream item remains visible in the browser window and then pushing back this info toward Feedly's servers; information on various RSS streams and topics (tags) collected from its user base could be used by Feedly to build profiles needed to filter out uninteresting items from user streams, or to promote more interesting news to the top of the stream.

We evaluated the performance of our solution through a simulation-based study whose results show that it is able to effectively profile both source and destination users and to automatically filter out a large percentage of events that would be negatively evaluated by their recipients with a low percentage of false positives (high quality events that are erroneously filtered out).

The rest of this paper is organized as follows: Section 2 presents related works; Section 3 defines the system model and states the problem; Section 4 presents our solution, evaluated in Section 5; finally, Section 6 concludes the paper.

2. Related work

The interest in reputation shown by economy and game theory (see for example [2]) was quickly followed by a surge of attention in ICT. As the number of online users and transactions increased, transcending geographical limitations and personal acquaintances, traditional one-to-one word of mouth proved insufficient.

Very soon, the first systems supporting reputation appeared online, proving themselves essential for trust maintenance and partner selection, and at the same time showing their vulnerabilities under specific attacks [3], pointing out the best statistics for reputation estimation [4] and using simulation to show the importance of cognitive aspects [5].

More recently, surveys on applications for reputation systems began to appear [6,7]. Finally, as the field consolidated in the most recent years, simplified recipes [8] have been proposed. The interaction between reputation and crowdsourcing has also been explored recently [9].

Reputation in large scale dynamic systems has been studied in EigenTrust [10] that offered a fully decentralized solution to calculate a global reputation rating for every peer in a P2P network, based on their previous behavior. While the fully distributed approach is common to our solution, we aim at building quality estimators that characterize each publisher in different ways depending on the content of events he publishes. The same goal could be accomplished in Eigentrust only by running an instance for each CBA, an approach that would run extremely inefficiently.

A different solution in the area of reputation system has been proposed by Ismail and Josang in [4] where they introduced the *Beta Reputation System*. Their solution builds reputation scores using the beta probability density function, thus creating trust on the basis of a sound statistical methodology. Differently from the *beta reputation system*, that is proposed as a centralized solution, the system proposed in this paper provides a fully decentralized approach that still bases its output on the same beta function.

The authors of [11] pursued our same goal (i.e., thin out the user streams) with a different approach. Their solution is tailored to the Twitter social network and consists in profiling user by using the categories extracted from URLs posted in their tweets, and then using these categories to filter information stream. Conversely, the solution proposed in this work is of general applicability for any event-based systems.

Similarly to our work the solution proposed in [12] aims at offering a personalized stream of randomized web pages to each user. Their mechanism is strongly tied to collaborative filtering [13,14], a thriving subfield of machine learning, which involves finding similarities between users and making recommendations based on what similar users like. Collaborative filtering solutions in the literature are often divided in two groups: *memory-based* and *model-based* [15]. Memory-based methods are used in a lot of real-world systems because of their simple design and implementation. However, they impose several scalability limitations that make their use impractical when dealing with large amounts of data. Model-based approaches have been investigated to overcome the shortcomings of memory-based algorithms. The most successful Model-based techniques are by far those based on low-dimensional factor models, as the Netflix Prize (www.netflixprize.com) established, in particular those based on *matrix factorization* [16]. The most popular matrix factorization solutions are *Alternating Least Squares* (**ALS**) and *Stochastic Gradient Descent* (**SGD**). Recently, collaborative filtering has been applied to the area of binary rating recommendations with an efficient solution based on statistical profiling of user feedbacks [17]. None of the solutions in the literature, to the best of our knowledge, has been applied in a event-based messaging scenario. Furthermore, all these solutions build a model from an online training phase by continuously updating a large matrix including data for any user in the system. This approach typically delivers great recommendations but is extremely difficult to decentralize making it unappealing for distributed event-based systems. Conversely, our solution has been designed with the aim of being easily and effectively deployable is a fully distributed setting.

A typical problem that affects many currently deployed reputation/recommendation systems is represented by malicious users whose aim is to subvert the normal system behavior such to drive its output toward their desired goals. While systems are typically protected from direct attacks at their protocols and algorithms, defending against potentially large groups of colluding users is extremely difficult. Colluding users may, in fact, submit perfectly legit feedbacks to purposely drive the rating of an item toward unnaturally large or small values. This phenomenon, studied for example by Lien et al. in the Maze file-sharing system [18], has several complex facets and nuances that have been extensively analyzed in the survey by Hoffman, Zage and Nita-Rotaru [19]. Negative effects of such attacks can be mitigated by avoiding user impersonation or multiple identities (i.e. sybil attacks), limiting the generation and spread of false feedbacks and avoiding short-term abuses of the system. Our solution, designed with the aim of providing maximum efficiency and decentralization, does not adopt any of these countermeasures, and is thus susceptible to colluding attacks; however, by its nature it is not incompatible with the techniques described in the cited literature, and could thus be enhanced with collusion resistance capabilities. This evolution is left for future work.

3. System model and problem statement

We consider a system where a set of users can produce or consume information. Users that exchange information adopt an *event-based publish/subscribe* communication model [20]. Without loss of generality, we assume that a user is either a producer of events (*publisher*) or an event consumer (*subscriber*). Each event is characterized by a *content-based address* (CBA), defined in an *event space*, and a *payload*. Publisher can publish events in the whole event space, i.e. there is no restriction on the CBA of events. A subscriber can select the events he wants to receive by issuing a *subscription* that defines a subset of the event space, and thus restricts the CBAs accepted by the subscriber. If an

event is characterized by a CBA included in the set defined by a subscription we say that the event *matches* the subscription. Interactions between publishers and subscribers are decoupled by an *event notification service* (ENS) that receives events injected by publishers and notifies them to all and only the subscribers whose subscriptions are matched. This event selection model is general enough to encompass the so-called *content-based model* and the *topic-based model* (where CBA would simply represent *topics*) as our solution to the online filtering problem fits both.

From an information quality point of view, here we assume that a publisher is characterized by a different level of expertise in different CBAs. This models the fact that a user can be an expert on a specific topic and have only a superficial knowledge on a different one. The expertise a publisher sports on each specific CBA has an impact on the quality of events he publishes on that CBA. We also assume that the expertise of a publisher does not change over time. This simplifying assumption intuitively models the fact that we expect the usage of our system to last for a timeframe smaller than the time needed for a user to sensibly vary his expertise on a topic. Each Subscriber is notified about events from the ENS and can express a feedback (*OK*/*KO*) on each event depending on its personal evaluation of the event quality. Feedbacks can be explicit (a vote) or implicit (i.e. an event that is discarded by the subscriber can be interpreted as a negative feedback or one that is delivered at the application level can be interpreted as a positive one). To simplify the discussion and without loss of generality, we assume for the rest of this work that votes are explicitly provided. As for the publishers, we assume that a subscriber applies a consistent judgment approach when voting for events (i.e. if an event is voted *OK* by a subscriber, the same event cannot be voted *KO* by the same subscriber at a different point in time), and, furthermore, that this judgment approach is the same regardless of the subscriber's subscription. The ENS does not know the publishers' expertise, nor it knows the judgment approaches applied by subscribers, but we assume it only collects votes expressed on notified events.

The problem we want to solve can be expressed as follows: how can we reduce the number of notified events that are voted *KO* by the receiving subscribers?

An ideal solution to this problem consists in an ENS able to notify to subscribers all and only the events that will be voted as *OK*. Given the assumptions that characterize our model, this goal can be reached by reasoning on complete knowledge, i.e. letting the ENS collect votes for each triple ⟨*publisher*, *CBA*, *subscriber*⟩ and then deterministically discard those notifications characterized by a triple that previously received a *KO* vote. However, this solution is clearly poor from a scalability point of view as the number of triples can quickly explode, and consequently the amount of information that the ENS should maintain would quickly become unmanageable. In the next section we introduce a solution that is able to provide performance that approach those of the ideal solution at a fraction of its cost.

4. Online filtering solution

4.1. General overview

Our filtering solution is constituted by an algorithm that runs within the ENS throughout the whole system lifetime. The algorithm is structured in two phases: a *learning* phase (LP) and a *working* phase (WP). The aim of the learning phase is to collect votes and combine them to build profiles. Publisher and subscriber profiles are then used in the working phase to perform filtering on events that match subscriptions. Similarly to most *machine-learning* approaches, our solutions is not able to continuously adapt to variations in expertise levels, it should thus be run periodically. More advanced strategies (e.g. with continuous adaptation) are out of the scope of this paper and left for future work. The publisher profiling process consists in assigning them a reputation score for each CBA where they issued events. Subscribers are profiled using votes they express to find for each of them a reputation threshold (RT). Intuitively, reputation scores assigned to publishers should capture their expertise, while reputation thresholds should capture subscribers quality thresholds.

During the learning phase the algorithm gathers event votes from multiple subscribers, and combines these feedback to compute a publisher reputation score (REP) for the CBA. Multiple votes for a specific target (i.e., a specific ⟨*publisher*, *CBA*⟩ pair) from the same subscriber are discarded as they would all be identical. The REP value for a specific ⟨*publisher*, *CBA*⟩ pair changes over time due to votes coming from different subscribers that at runtime decide to subscribe/unsubscribe that CBA. As more and more votes are collected the REP value approaches an ideal value that represents the reputation score obtainable by collecting votes from all the subscribers. Limiting the LP duration our system faces a trade off between the quality of its estimation of the ideal REP value and the amount of information (i.e. votes) it must collect.

Every time a publisher publishes a new event in the system the current estimation of its REP value for the target CBA, together with an uncertainty measure called GAP, is piggybacked with the event. During the working phase, before notifying the event to a target subscriber, the algorithm checks if the REP value associated to the event is larger than the RT value of the subscriber. In this case the event is notified, otherwise the event is dropped. Whenever some information is missing (e.g. an event without a REP value, or a non profiled subscriber without a reputation threshold) the event is notified. Ideally, our solution should create publisher and subscriber profiles such that the REP value associated to an event is greater than the RT value of all and only the subscribers that would vote OK for it.

Given an event space model, our solution builds an estimator for each publisher on each CBA and an estimator for each subscriber. While the total number of estimators could possibly grow to a very large size it is important to note that (i) this is only an upper bound as most event-based systems see events and subscriptions concentrate on few popular spots in the event space thus greatly reducing the actual number of estimators, (ii) each estimator consists in a small and constant amount of information (usually a real number), (iii) only estimators for publishers depend on the CBA, while a single estimator is maintained for each subscriber and finally (iv) whenever a complex event selection method is used (e.g. a content-based model with a multiple-dimensions event space and a mix of equality and range selection constraints) it is always possible to map the event space to a desired number of discrete channels (or topics) [21,22].

4.2. Beta reputation

The *Beta Reputation* [4] provides a mathematical method for computing reputation scores on the basis of binary evaluations (the *OK*/*KO* votes). It is based on the Beta distribution, a continuous family of probability functions indexed by two parameters α and β. Given a number of received votes $OK + KO$, the unknown relative frequency of *OK* votes a producer will receive in the future has a probability distribution expressed by a Beta function with parameters α and β set to the number of *OK* and *KO* incremented by one respectively, as in the following: $\alpha = OK + 1$ and $\beta = KO + 1$.

The expected value (mean) of the Beta distribution is the most natural way to estimate the reputation score REP:

$$Rep(OK, KO) = \frac{\alpha}{\alpha + \beta} = \frac{OK + 1}{OK + KO + 2}. \quad (1)$$

REP will then have values in the range (0, 1), where value 0.5 represents a neutral rating. We can interpret this expectation (reputation score) as the most likely value for the relative frequency of OK votes that the publisher will obtain in the future.

The standard error of the Beta distribution provides a measure of the inaccuracy of the reputation estimator:

$$se(OK, KO) = \frac{1}{OK + KO + 2}\sqrt{\frac{(OK+1)(KO+1)}{(OK+KO)(OK+KO+3)}}. \quad (2)$$

This measure is inversely proportional to the number of votes ($OK + KO$). Intuitively, the more representative is the subset of voters, the more accurate will be the reputation estimation.

The expected value of the Beta distribution and his standard error form a basis for producer profiling, and are the building blocks of our filtering algorithm.

4.3. Filtering algorithm

4.3.1. Producers profiling

The algorithm associates to each pair $\langle publisher, CBA\rangle$ two values: a reputation score (REP) and an inaccuracy estimation of this score (GAP). The idea behind this latter score is to define an uncertainty range around the reputation estimation: the boundaries of this interval are defined by adding and subtracting the GAP value from the reputation score. As more and more votes are collected for a pair $\langle publisher, CBA\rangle$ this range shrinks, converging to 0 for an exact estimation of REP. The reputation score REP is computed using the expected value of the Beta distribution (Eq. (1)) and his standard error (Eq. (2)) constitutes the GAP. The system updates these values during the learning phase each time a publisher publishes a new event on a CBA, and new votes are collected.

4.3.2. Consumers profiling

Subscribers are profiled using an explorative mechanism. The goal of the algorithm is to identify a RT value for each subscriber using the votes it expresses. When a subscriber emits a new vote v on a notified event with REP r and GAP g, the algorithm updates the reputation score and the GAP measure for the target $\langle publisher, CBA\rangle$, and it also stores a triple $\langle v, r, g\rangle$ associated to the subscriber. All triples collected for a subscriber at runtime are sorted by placing them on a [0, 1] scale according to a *worst case estimation* approach: given a triple $t = \langle v, r, g\rangle$ if $v = KO$ the position of the triple is $r - g$, otherwise it is $r + g$.

The subscriber's RT value is then chosen by picking the smallest value τ that minimizes the number of positive votes with a position $x_{OK} \leq \tau$ on the scale and the number of negative votes with position $x_{KO} > \tau$. This *worst case estimation* approach prevents inaccurate reputation estimations from corrupting the RT. Indeed, without this mechanism negative votes with over-estimate reputation scores would lead to over-strict RTs (positive votes with under-estimate reputation would lead to over-permissive RTs respectively).

Subscriber profiling does not stop with the end of the learning phase. Given the imprecision in the REP estimation, false positive notifications can still take place (i.e. an event not filtered by our solution could receive a KO vote from an already profiled subscriber) during the working phase. A KO vote expressed by a subscriber during the WP is considered as an error notification by our algorithm and triggers a subscriber profile update: the update follows the procedure used for regular votes.

4.3.3. Profiles match

The algorithm drops all notifications of events whose attached REP value is lower or equal than the reputation thresholds on the target subscribers. The initial values for publishers REP and GAP is ∞ and 0 for subscribers RTs. In this way the first event issued by a publisher on a specific CBA is always delivered to all the subscribers subscribed to that CBA. RTs initialization to value 0 guarantees that the filtering system will not filter-out any incoming event for users who refrain from voting, thus representing a fully transparent solution for them.

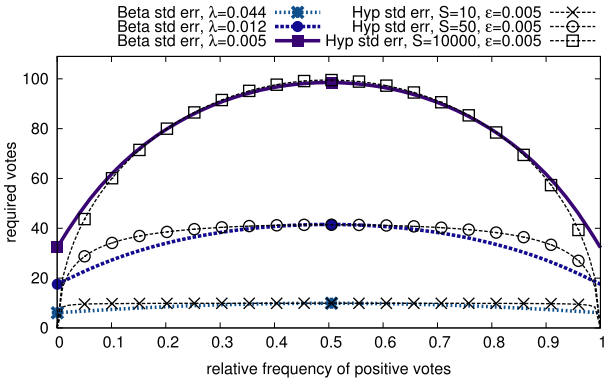


Fig. 1. Number of votes required to obtain a specific standard error given a ratio of positive votes. Beta standard error (solid points) assume infinite population and might exceed the number of existing subscribers if high precision is required (low λ). Hypergeometric standard error (empty points), is plotted for high precision, but takes into account the finite population S.

4.3.4. Learning phase stop condition

During the learning phase votes are collected to improve the quality of publisher profiles. The more votes are collected for a pair $\langle publisher, CBA\rangle$, the more the REP value will converge toward its final value and the GAP value will converge to 0. This convergence process is unlikely to be linear: the first few votes will strongly drive REP toward its target value strongly reducing the GAP, while further votes will marginally refine this result. Obviously, the system incurs some form of overhead for vote collection.[1] For these reasons, the decision about when a learning phase should be stopped represents a crucial point of our algorithm: the goal is to build high quality profiles to provide highly performant filtering with a small number of votes. When the learning phase stops for a pair $\langle publisher, CBA\rangle$ the algorithm moves to the working phase where the REP value is kept constant (and set to the last value it has during the LP) and the GAP value is cleared and set to 0. Given that the GAP represents the inaccuracy of our REP estimation it make sense to consider a condition on its value to stop the LP: when the GAP is smaller than a given threshold the LP can be stopped.

Fig. 1 depicts, with a solid curve, the amount of votes that the algorithm should collect to have the Beta distribution *se* smaller than a desired value λ ($\lambda = 0.005$ in this specific case). This amount of votes varies depending on the ratio between OK and KO votes received thus far; when we receive only OKs (or only KOs), the number of required votes to reach the desired accuracy level is lower than the case where OKs and KOs are equally split (0.5); in the latter case the algorithm requires a lot more votes to be "confident enough" that the subscriber population is equally split in its opinions. This curve plays a fundamental role as it tells us, given a set of votes already received for a pair $\langle publisher, CBA\rangle$, how many more votes we will collect before the LP will be stopped as the GAP size threshold is met. Moreover, it (probabilistically) guarantees us that when the LP will be stopped the obtained REP value will have an error bounded by $\pm\lambda$.

[1] The exact quantification of this overhead depends on the specific approach used to implement vote collection in the system of choice.

This solution, however, does not take into account the total number of subscribers S available in the system (it actually assumes $S = \infty$) and can thus lead to stopping conditions that will never be met because there are simply not enough subscribers to provide all the votes needed to stop the LP.

If total number of subscribers S is known by the system, the voting process can be modeled by an *Hypergeometric distribution*, a discrete probability distribution that in this case describes the probability of collecting x OK votes out of $x + y$ votes (without duplicate votes) from a finite set of potential votes of size S containing a maximum of X OK votes.

The standard error of the Hypergeometric distribution is:

$$se = \frac{1}{S}\sqrt{r(1-r)\frac{S-n}{S-1}} \tag{3}$$

where r is the relative fraction X/S of OK votes on the subscriber population.

If we impose a bound ϵ on this standard error we can again plot the number of votes that must be collected given a certain value for r. The three dashed curves with empty dots reported in Fig. 1 shows how this number of plots varies against r (reported on the x axis) for three different population sizes. While the Hypergeometric distribution *se* overcomes the limitation previously outlined for the Beta distribution *se*, it cannot be adopted in our system as we do not know r.

To get the best of both approaches our solution is to scale the Beta curve on the basis of the population[2] S such that the maximum of the Beta *se* curve matches the maximum of the corresponding Hypergeometric *se* curve. After scaling the curve, the corresponding λ value can be computed (In the figure, for example, $\epsilon = 0.005$ and $S = 50$ for the Hypergeometric leads to $\lambda = 0.012$), and this will constitute the threshold on the GAP value that will decide the LP conclusion. This strategy provides a practical solution to adapt the GAP threshold to various populations and strongly reduces the possibility that some LPs will never end. It is worth noticing that if votes for a given pair $\langle publisher, CBA\rangle$ arrive at a reduced rate, either because few subscribers have subscribed for that CBA or only a fraction of the subscribers express their votes, this will possibly slow down the learning phase with limited impact for other CBAs and publishers.

5. Experimental evaluation

In this section we report on the experimental evaluation we conducted on our solution. The goal of this evaluation was to assess how much our solution is effective in filtering undesired events and how sensitive it is to different characteristics of the input load. In the following subsection we will first introduce the metrics of interest for our evaluation, we will then detail the simulation model we adopted and finally, we will describe the results of the evaluation.

5.1. Performance metrics

In general, by looking at the final effect of our filtering solution applied to a specific notification we can identify four possible cases: an event notified to a subscriber that votes *OK* represents a *true positive* outcome (**TP**); an event notified to a subscriber that votes *KO* represents a *false positive* outcome (**FP**); an event that is filtered for a subscriber that would vote *OK* on its notification represents a *false negative* outcome (**FN**); an event that is filtered for a subscriber that would vote *KO* on its notification represents a *true negative* outcome (**TN**). To measure the performance of a *binary classifier* the concepts of *sensitivity* and *specificity* are often adopted. If we apply their definitions to our system we have:

- *Sensitivity* (or True Positive Rate (**TPR**)) is the fraction of subscribers that are notified about an event (TP) among all those that would express an *OK* vote on its notification (TP + FN). It measures the proportion of actual positives which are correctly identified as such;
- *Specificity* (or True Negative Rate (**TNR**)) is the fractions of subscribers that are not notified of an event (TN) among all those that would express a *KO* vote on its notification (TN + FP). It measures the proportion of negatives which are correctly identified as such;

Furthermore, we consider the following metrics:

- *Matthews correlation coefficient* (**MCC**) measures the quality of binary classifications. It returns a value between -1 and $+1$ where $+1$ represents a perfect prediction, 0 no better than random prediction and -1 indicates total disagreement between prediction and observation.

$$MCC = \frac{TP \times TN - FP \times FN}{\sqrt{(TP + FP) \times (TP + FN) \times (TN + FP) \times (TN + FN)}}. \tag{4}$$

- *Knowledge size* (**K**) is the average ratio, among all subscribers in the system, between the number of pairs $\langle publisher, CBA\rangle$ for which a subscriber s expressed at least a vote and the global number of pairs for which s received an event. $K = 1$ means that every subscriber expressed a vote for each publisher on each CBA, i.e. the system collects full knowledge and the cost paid for this is maximum.[3] Therefore, a good filtering solution should work keeping K as small as possible.

5.2. Simulation model

To perform our evaluation we integrated our filtering solution within a distributed event notification service and implemented it with the *OMNeT++* discrete event simulation environment. We used SIENA [23] as a reference notification service. Note, however, that this specific choice has no practical impact on the results reported in this section as the measured metrics do not depend on the specific event selection model and event routing strategies used by the underlying system. The evaluation, in fact, would give the very same results if implemented simulating a single process keeping a map of CBAs and subscribers.

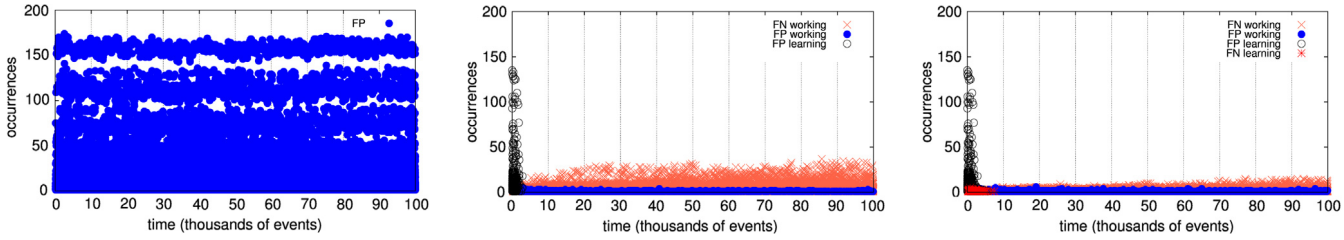
We consider a simplified data model where the event space is constituted by a single integer attribute with values ranging from 1 to 30. This *topic-based* model does not impact the generality of the results reported in the remainder of this section as the considered metrics only depend on the distribution and amount of events published on each CBA.

In our simulations we do the simplifying assumption that events published by a publisher on a CBA are characterized by a quality level that corresponds to the expertise level of that publisher on that CBA. Every time a subscriber with quality threshold t is notified about an event with quality q such that $q \leq t$ it votes *KO*, otherwise it votes *OK*.

The behavior of publishers and subscribers was modeled taking into account existing studies on the dynamics of publish/subscribe

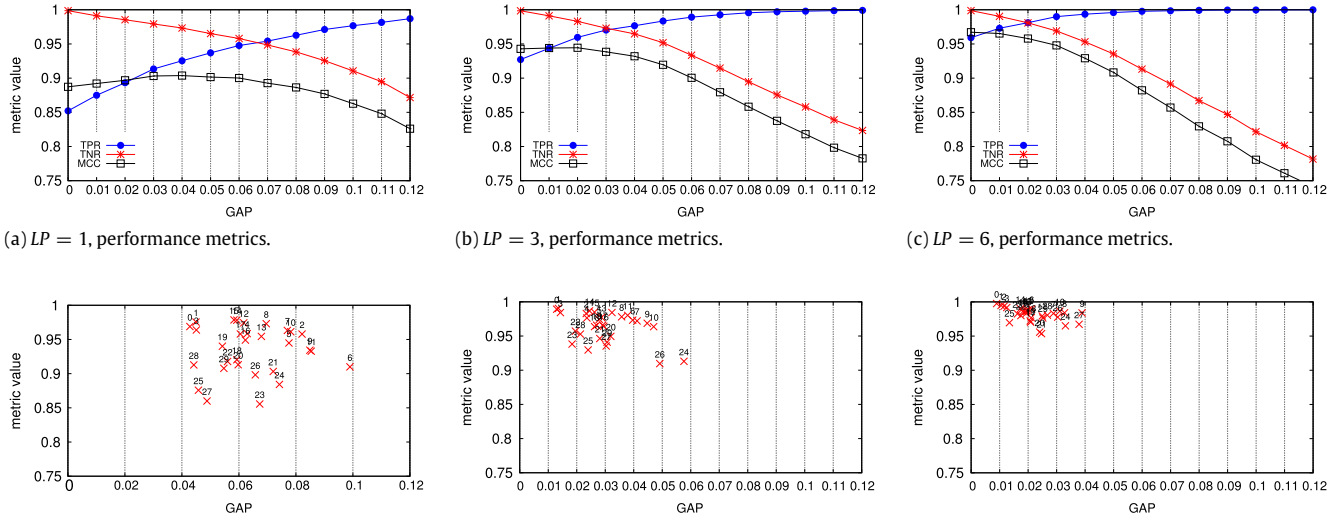
[2] Note that S may be known or estimated depending on the specific event-based system that is considered.

[3] The knowledge size K could be considered as an abstract measure of the overhead generated by our solution. The actual overhead (in terms of messages, bandwidth, CPU usage, etc.) strictly depends on the specific system hosting our solution.



(a) No filtering. (b) $LP = 1$. (c) $LP = 6$.

Fig. 2. General filter behavior. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)



(a) $LP = 1$, performance metrics. (b) $LP = 3$, performance metrics. (c) $LP = 6$, performance metrics.

(d) $LP = 1$, intersection points. (e) $LP = 3$, intersection points. (f) $LP = 6$, intersection points.

Fig. 3. Tradeoff between performance metrics varying LP and GAP.

applications and of churn in peer-to-peer networks [24]. We considered a fixed set of 200 subscribers. Every subscriber expresses 3 subscriptions each targeting a single CBA. A subscription lasts for a limited amount of time, defined by a Weibull distribution (shape $k = 0.5$ and scale $\lambda = 5$), after which it is replaced with a new one. Intuitively most of the subscriptions will last for a short timeframe (minutes) while some subscriptions will be characterized by a long duration (days or weeks). Subscription allocation over CBAs follows a Pareto distribution (scale $x_m = 1$ and shape $a = 1$), mimicking a common behavior in event-based systems, where few CBAs represent very popular topics while a lot of CBAs only have few subscriptions. The popularity of CBAs decrease proportionally to the CBA id (i.e., CBA with id 1 is the most populous, with id 30 the least). Publishers publish events at a rate characterized by a Poisson distribution ($\lambda = 5$) on CBAs that are chosen uniformly at random in the event space. We considered only 18 publishers in our tests as the metrics of interest do not depend from this parameter. Given that our tests are driven by the injection of new events, we limited the duration of each test run to the 100.000th event injected in the system. The publishers' expertise on the available CBAs was modeled using different continuous functions, in order to mimic an application scenario where different publishers sport widely different knowledge levels on different topics. Subscriber quality thresholds were modeled as a Normal distribution (mean $\mu = 0.6$ and standard deviation $\sigma = 0.15$).

5.3. Evaluation results

General filter behavior. Our filtering solution has a two-phases behavior (cfr. Section 4.1): an initial learning phase of length LP where the system collects votes to build profiles, followed by a working phase where profiles are used to filter events. Fig. 2(a)–(c) show the evolution of the false positive FP metric as new events are injected for three different settings: no filtering applied to events (Fig. 2(a)), filtering with LP limited to the first event produced in each CBA by each publisher ($LP = 1$ in Fig. 2(b)), and filtering with LP limited to the first 6 events ($LP = 6$ in Fig. 2(c)). In the last two graphs the different phases can be easily recognized.

Fig. 2(a) shows how the number of false positives is large if no filtering is applied. Graphs in Fig. 2(b) and (c) show a sharp decrease of the number of false positives during the learning phase. At steady-state (working phase) the FP metric is stable with a low number of occurrences. It is worth noting that the introduction of the filter can create false negatives (reported as red crosses in the graphs) whose amount can be controlled by increasing the learning phase duration. Intuitively, by learning for a longer amount of time the filter precision can be improved. From a theoretical standpoint, by extending the learning phase to the whole system execution, we could obtain an "ideal" system that performs with no false negative/positive at steady state. However, such system would incur the maximum cost, i.e. it would need to gather votes from all subscribers on all CBAs for all the publishers, and thus reason on complete knowledge.

Learning phase duration vs. knowledge. The above results show the importance of characterizing the tradeoff between the performance of our filter and the cost we incur by increasing the learning phase duration. A fundamental role in this tradeoff is played by the GAP introduced in Section 4.3 as it offers a way to measure the residual uncertainty during the learning phase.

Fig. 3 reports the behavior of the filter varying the GAP size (0.00–0.12) and the learning phase duration (1, 3 and 6 events). The graphs on the top show the evolution of the TPR, TNR and MCC

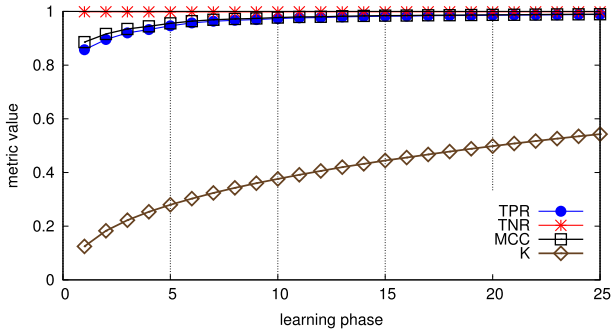


Fig. 4. K varying learning phase duration.

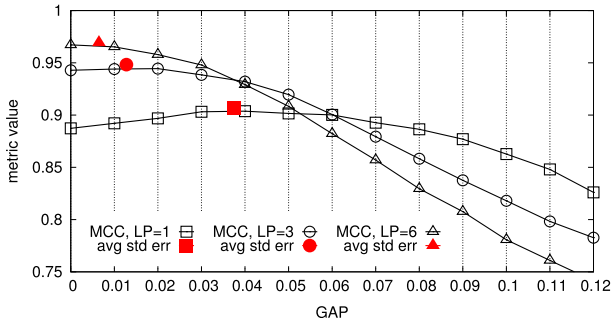


Fig. 5. MCC varying the GAP size.

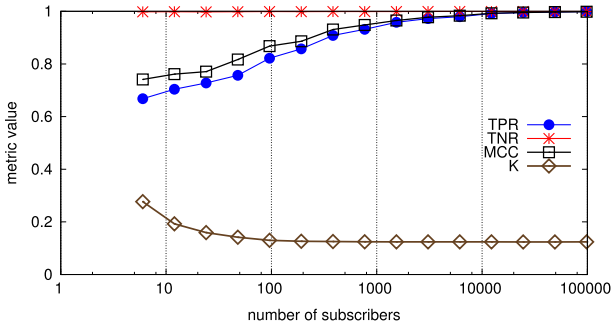


Fig. 6. Performance metrics and K vs. number of subscribers ($LP = 1$, $GAP = 0$).

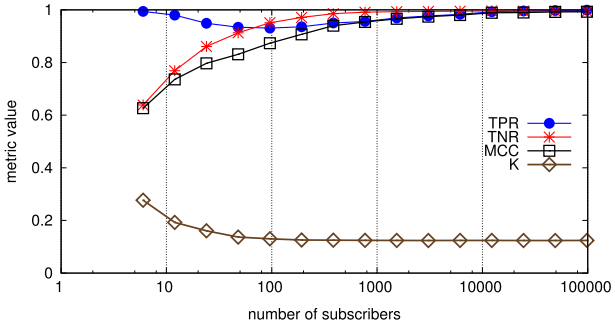


Fig. 7. Performance metrics and K vs. number of subscribers with adaptive GAP ($LP = 1$).

metrics varying the GAP size. The curves clearly show how the GAP size controls the tradeoff between TPR and TNR: when the GAP is set to 0 the filter is very effective in avoiding the notification of undesired events, but this comes at the cost of a lot of events that are discarded even if they would be rated OK by their target subscribers. As the GAP grows more events fall in its uncertainty window and are thus notified; as a consequence, the TPR metric tends to 1 while the TNR quickly drops to low values.

The intersection point of these two curves may look interesting at a first glance as it represent the "best" performance compromise between TPR and TNR. However, it does not have a specific significance for our solution as the two curves only represent the average TPR and TNR values among all the CBAs. The graphs on the bottom of the Figure depict all the intersection points between these two curves calculated separately for each CBA (the number attached to each point is the CBA identifier), and show how the choice of the "best" GAP size when $LP = 1$ is quite difficult as no size fits all. This is a consequence of the complex dynamics that link event production, publishers expertise, and subscription distribution among the different available CBAs. The MCC metric confirms this as its maximum is obtained with lower values of GAP with respect to the intersection of the average TPR and TNR curves. When we increase the learning phase duration to 3 and 6 events the TPR and TNR curves both tend to 1 and their intersection points moves left toward lower values of the GAP.

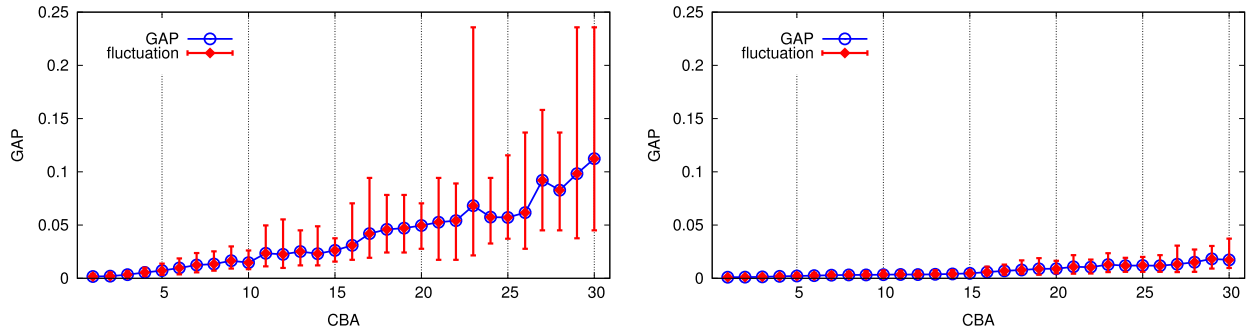
These results clarify a fundamental point: given a learning phase duration, the GAP should be sized independently and differently for each CBA as their characteristics may vary, especially when the learning phase is short. On the other side, increasing the LP duration seems to improve performance. This, however, comes at the cost of a larger knowledge size. Keeping LP small while attaining good performance is a desirable result as the knowledge K, needed by the filter to build profiles, grows with it as shown by the graph in Fig. 4.

Fig. 5 reports the three MCC curves extracted from Fig. 3. At a first glance the evolution of the three seems contradictory: when the GAP value is small the maximum MCC is obtained with the longest learning phase ($LP = 6$), and this is consistent to what we claimed before; however, when the GAP is large shorter LPs give the best results. By looking back at the results from Fig. 3 they actually reveal that while for $LP = 1$ a decrease of TNR corresponds to an increase of TPR even when the GAP is large, and so the MCC remains relatively stable, the same is not true for $LP = 6$; in this case the TPR metric soon reaches its maximum value and thus the MCC is driven only by the TNR value that drops as the GAP size grows.

Also the total number of subscribers in the system has a strong impact on all the metrics. Fig. 6 reports the evolution of TPR, TNR and MMC metrics when the number of total subscribers in the system grows. The graph shows clearly how the filter effectiveness grows with the total number of subscribers approaching the ideal case (i.e. all metrics tend to 1). This was expected as a larger number of subscribers provide a larger number of votes and this means that REP can be calculated on a larger population sample. The same graph also reports the amount of knowledge collected by our solution that shrinks as the number of subscribers grows converging to an asymptotic minimum, whose value can be roughly approximated by the rate between the total number of subscriptions and the number of subscribers times the number of available CBAs. When the number of subscribers is small the value is perturbed by a border effect caused by the fact that there are less subscribers than available CBAs.

Adaptive GAP size estimation. Intuitively the GAP is a measure of the imprecision of the reputation estimation REP. The *standard error* (se) of the Beta distribution represents a viable solution to characterize this imprecision by calculating the variability of the estimator. By adopting this strategy, our solution is able to estimate the uncertainty for each pair $\langle publisher, CBA \rangle$.

Fig. 5 reports the three MCC curves varying the GAP size (0.00–0.12) and the learning phase duration (1, 3 and 6 events). We added to Fig. 5 three red points representing the average values of the GAP estimated through the Beta distribution se for all $\langle publisher, CBA \rangle$ combinations. The adaptive GAP size estimation is able to select the size delivering maximum MCC value. Intuitively this means that the Beta se can be used to track the "sweet spot" of



(a) $LP = 1$. (b) $LP = 6$.

Fig. 8. Beta *se*.

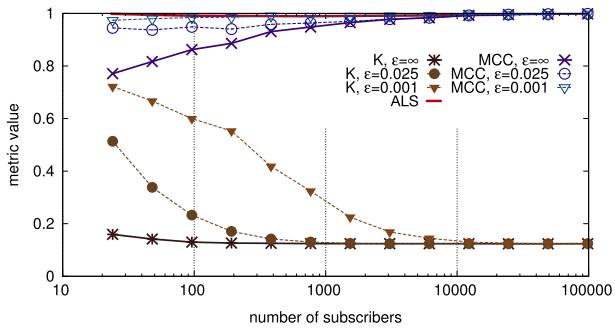


Fig. 9. Evolution of MCC and K with number of subscribers.

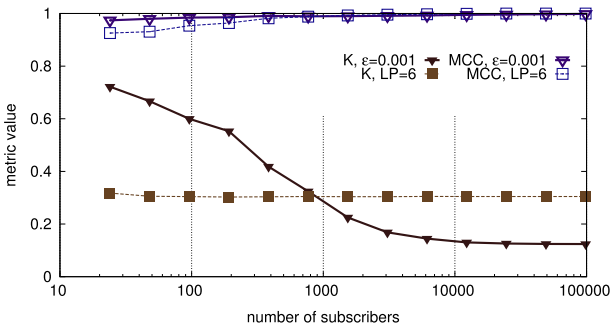


Fig. 10. Fixed vs. GAP-based LP stopping conditions.

the MCC curve, i.e. the configuration that provides the best tradeoff between TNR and TPR.

Looking at the filter performance in Fig. 7 the result seems counterintuitive: when we use the adaptive GAP size estimation the MCC value drops and the knowledge K grows for smaller subscriber populations. However, by looking at the TPR and TNR metrics it is possible to understand what happened: when the number of subscribers is small, the number of samples used as input to the REP function is so small that the corresponding GAP size is kept very large. As a consequence the number of undesired event that are filtered is kept low in order to avoid filtering out too much desired events. This is a desirable behavior as we assume that in most application scenarios subscribers could be willing to tolerate a certain number of undesired events but could be upset by not receiving desired events.[4]

Fig. 8(a) and (b) plots the Beta *se* of two runs, respectively with a learning phase duration of 1 and 6 events, for distinct CBAs. The first observation is that the *se* is inversely proportional to the population of a CBA, which follows from Formula (2). As the learning phase grows, Beta *se* curve tends to flatten, due to the increase of samples collected in the less popular CBAs. Actually, the imprecision estimation in the most popular CBAs is small even with a short learning phase (a single event). It is therefore pointless to increase the knowledge for those CBAs since the system already has an accurate reputation estimation for them. This leads to the idea of tuning the learning phase duration according to the GAP calculated on each specific pair $\langle publisher, CBA \rangle$ through the Beta *se*: the smaller is the GAP estimation, the shorter the learning phase will last on that specific $\langle publisher, CBA \rangle$.

Learning phase stopping condition. Tuning the learning phase duration on the Beta standard error means defining a threshold for it. In Section 4.3 we have shown how to define a threshold ϵ on the Beta *se* having a rough estimation of the number of subscribers.

Fig. 9 depicts how *MCC* and K evolve as the number of subscribers grows for three ϵ values. When $\epsilon = \infty$ the system has a single event LP duration for all CBAs and thus requires the least knowledge K. As we reduce ϵ the filter quickly becomes more effective, and the required knowledge grows as well. Interestingly, when ϵ is set to 0.001 or lower performance stabilize as *MCC* is close to 1. Therefore, setting a threshold lower than this only increases K without any perceivable performance improvement. As a reference for the reader the graph also shows the MCC obtainable by using a state-of-the-art collaborative filtering algorithm, namely ALS. Note that the MCC values for ALS are very close to 1.0 as the algorithm works in a centralized fashion on the full dataset of ratings (4/5 of the dataset were used for training the model and 1/5 to evaluate it). Conversely, our solution shows comparable performance at a fraction of the cost.

With this mechanism it is possible to obtain an almost perfect filter ($MCC \approx 1$), whatever the number of subscribers. To achieve this goal the system automatically calibrate the learning phase duration according to the standard error computed for the specific pair $\langle publisher, CBA \rangle$. Fig. 10 wraps up the latter results and compares the performance of our solution against a straightforward approach based on a fixed LP duration (6 events). The graph shows that when the number of subscribers is small, the GAP-based stopping condition is able to adapt to different CBA popularities providing performance that closely approach the optimum at the cost of a larger knowledge size K; conversely, when the number of subscribers grows, the MCC values of both solutions tend to 1, but the solution with the GAP-based stopping condition sports a much smaller K value.

Wrapping-up these results the system administrators that adopt our solution should only care about correctly tuning two parameters: the expected number of subscribers S and the threshold ϵ. The former is usually the result of an educated guess based on prospected system usage that is any case commonly

[4] All applications where this assumption does not hold would be better served by setting the gap to 0.

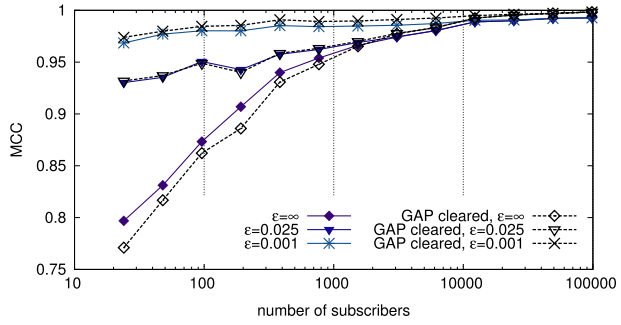


Fig. 11. GAP strategies comparison.

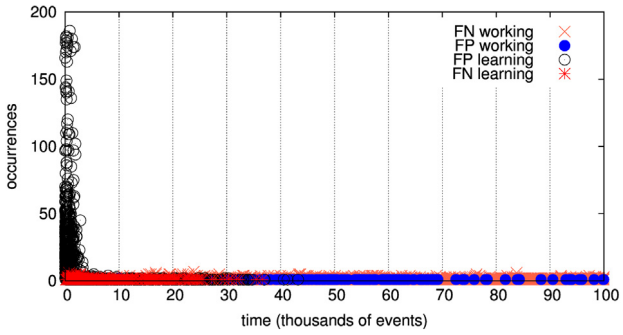


Fig. 12. Online filtering solution.

performed at deployment time to correctly provision hardware to run the underlying event notification service. Overestimating this value will only increase the length of the learning phases thus creating more overhead, without impact on quality metrics. Defining the correct value for the threshold ϵ boils down to tuning a tradeoff between quality versus cost. As Fig. 9 pointed out, setting the value for ϵ very low will provide the best filtering quality (i.e. MCC close to 1) at the cost of an increased overhead (K) but only when the number of subscriber S is small. Given that most event-based systems sport thousands to millions of subscribers, setting $\epsilon = 0.001$ will provide a good starting point for further fine tuning.

Working phase GAP size. The last point that must be clarified is what is the best strategy to adopt when the learning phase ends as the GAP size shrinks below the ϵ threshold: either we can maintain the final calculated GAP or clear its value and set it to 0.

Fig. 11 compares the two strategies using the MCC as the number of subscribers grows and for various ϵ thresholds. As the graph shows, clearing the gap when the LP ends delivers worse performance only when the REP estimation is poorly accurate due to both a small number of subscribers and a large threshold ϵ. Contrarily, as the estimation accuracy improves (i.e. the MCC grows), either because there are lots of subscribers or due to a strict threshold ϵ, clearing the GAP becomes a good choice as it delivers better performance.

Full system evaluation. Fig. 12 wraps up all the results reported above by showing the evolution of the FP and FN metrics as events are injected in the system. By comparing this graph with those previously shown in Fig. 2(b) and (c), the improved results brought by the GAP adaptivity that drives the learning phase durations are evident. The amount of FPs and FNs during the working phase can be considered residual.

6. Conclusions

In this paper, we introduced a novel solution for online filtering of information in event-based system. Our solution targets applications where consumers express binary preferences on data they receive to capture their personal perception of the data quality. Our solution collects these preferences to create producer/consumer profiles that are then used to filter out events that consumers would vote negatively. In this way our solution is capable of reducing the amount of undesired information received by consumers. Some points remain still to be explored. In its current form our solution bases its functioning on the assumption that producer expertise will not change over time and the assumption that quality evaluations of subscribers do not depend from the information topic. While the algorithm could be easily adapted to circumvent these issues (e.g. by update profiles periodically, or by building several profiles per consumer), these adaptation would lead to poor performance paving the way to the investigation of better solutions.

References

[1] D. Eyers, T. Freudenreich, A. Margara, S. Frischbier, P. Pietzuch, P. Eugster, Living in the present: on-the-fly information processing in scalable web architectures, in: Proceedings of the 2nd International Workshop on Cloud Computing Platforms, CloudCP'12, ACM, New York, NY, USA, 2012, pp. 6:1–6:6. [Online]. Available: http://doi.acm.org/10.1145/2168697.2168703.

[2] D. Kreps, R. Wilson, Reputation and imperfect information, J. Econ. Theory 27 (2) (1982) 253–279. [Online]. Available: http://dx.doi.org/10.1016/0022-0531(82)90030-8.

[3] C. Dellarocas, Immunizing online reputation reporting systems against unfair ratings and discriminatory behavior, in: EC'00: Proceedings of the 2nd ACM Conference on Electronic Commerce, ACM, New York, NY, USA, 2000, pp. 150–157. [Online]. Available: http://dx.doi.org/10.1145/352871.352889.

[4] R. Ismail, A. Josang, The beta reputation system, in: Proceedings of the 15th Bled Conference on Electronic Commerce, 2002.

[5] M. Paolucci, False reputation in social control, Adv. Complex Syst. 3 (4) (2000) 39–52. [Online]. Available: http://ccs.mit.edu/dell/aa2001/aa2001papers%5CFalseReputation.doc.

[6] A. Josang, R. Ismail, C. Boyd, A survey of trust and reputation systems for online service provision, Decis. Support Syst. 43 (2) (2007) 618–644. [Online]. Available: http://dx.doi.org/10.1016/j.dss.2005.05.019.

[7] M. Paolucci, S. Picascia, S. Marmo, Electronic reputation systems, in: Handbook of Research on Web 2.0, 3.0, and X.0, IGI Global, 2010, pp. 411–429. (Chapter 23) [Online]. Available: http://dx.doi.org/10.4018/978-1-60566-384-5.ch023.

[8] C.N. Dellarocas, Designing Reputation Systems for the Social Web, Social Science Research Network Working Paper Series, Jun. 2010. [Online]. Available: http://ssrn.com/abstract=1624697.

[9] L. De Alfaro, A. Kulshreshtha, I. Pye, B.T. Adler, Reputation systems for open collaboration, Commun. ACM 8 (2011) [Online]. Available: http://cacm.acm.org/magazines/2011/8/114943-reputation-systems-for-open-collaboration/comments#.TlYt9LgjO4c.citeulike.

[10] M.S.S. Kamvar, H. Garcia-Molina, The eigentrust algorithm for reputation management in p2p networks, in: World Wide Web Conf., WWW, 2003.

[11] S.G. Esparza, M.P. O'Mahony, B. Smyth, Towards the profiling of twitter users for topic-based filtering, in: Research and Development in Intelligent Systems XXIX, Springer, 2012, pp. 273–286.

[12] S. Ioannidis, L. Massoulié, Surfing the blogosphere: Optimal personalized strategies for searching the web, in: INFOCOM, 2010 Proceedings IEEE, IEEE, 2010.

[13] A.S. Das, M. Datar, A. Garg, S. Rajaram, Google news personalization: scalable online collaborative filtering, in: Proceedings of the 16th International Conference on World Wide Web, ACM, 2007, pp. 271–280.

[14] X. Su, T.M. Khoshgoftaar, A survey of collaborative filtering techniques, in: Advances in Artificial Intelligence, vol. 2009, 2009, p. 4.

[15] J.S. Breese, D. Heckerman, C. Kadie, Empirical analysis of predictive algorithms for collaborative filtering, in: Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence, 1998.

[16] Y. Koren, Factorization meets the neighborhood: a multifaceted collaborative filtering model, in: Proceedings of the 14th International Conference on Knowledge Discovery and Data Mining, 2008.

[17] F. Petroni, L. Querzoni, R. Beraldi, M. Paolucci, Lcbm: Statistics-based parallel collaborative filtering, in: Business Information Systems, Springer, 2014, pp. 172–184.

[18] Q. Lian, Z. Zhang, M. Yang, B.Y. Zhao, Y. Dai, X. Li, An empirical study of collusion behavior in the maze p2p file-sharing system, in: 27th International Conference on Distributed Computing Systems, ICDCS'07, IEEE, 2007.

[19] K. Hoffman, D. Zage, C. Nita-Rotaru, A survey of attack and defense techniques for reputation systems, ACM Comput. Surv. 42 (1) (2009) 1:1–1:31. [Online]. Available: http://doi.acm.org/10.1145/1592451.1592452.

[20] R. Baldoni, L. Querzoni, S. Tarkoma, A. Virgillito, Distributed event routing in publish/subscribe communication systems, in: B.G.H. Miranda, L. Rodrigues (Eds.), MiNEMA State-of-the-Art Book, Springer, 2009.

[21] M. Adler, Z. Ge, J. Kurose, D. Towsley, S. Zabele, Channelization problem in large scale data dissemination, in: Network Protocols, 2001. Ninth International Conference on, 2001, pp. 100–109.
[22] R. Beraldi, A. Cerocchi, F. Papale, L. Querzoni, Brief announcement: distributed self-organizing event space partitioning for content-based publish/subscribe systems, in: Stabilization, Safety, and Security of Distributed Systems, Springer, 2011, pp. 437–438.
[23] A. Carzaniga, D. Rosenblum, A. Wolf, Design and evaluation of a wide-area notification service, ACM Transactions on Computer Systems 3 (19) (2001) 332–383.
[24] D. Stutzbach, R. Rejaie, Understanding Churn in Peer-to-Peer networks, in: Proceedings of the 6th ACM SIGCOMM Conference on Internet Measurement, IMC'06, 2006.

Fabio Petroni received his Master Degree cum Laude in Computer Engineering from Sapienza University of Rome in 2012. He is currently a Ph.D. student in Computer Engineering at Sapienza University of Rome. His research interests include data mining, natural language processing, factorization methods, machine learning and distributed computing. Fabio is author of 8 papers about these topics, published on international conferences, workshops and journals (e.g., EMNLP, ACM CIKM, ACM RecSys, PNAS). He collaborates as a reviewer for international conferences and journals in related research areas (e.g. EDBT, ACM TIST, ACM PODC).

Leonardo Querzoni is assistant professor at Sapienza University of Rome. He obtained a Ph.D. in computer engineering with a work on efficient data dissemination though the publish/subscribe communication paradigm. His research activities are focused on several computer science fields including distributed stream processing, dependability and security in distributed systems, large scale and dynamic distributed systems, publish/subscribe middleware services. The results of these activities have been published on several papers appeared on renewed international conferences and journals. He regularly serves in the technical program committees of conferences in the field of dependability and event-based systems like DSN 2016 and ACM DEBS 2009–2015, and collaborates as a reviewer for international conferences and journals in related research areas (e.g. ICDCS, Middleware, ACM surveys, IEEE TPDS and TSC, Distributed Computing). He has been general chair for the 2014 edition of the OPODIS conference. He has been appointed as Ph.D. Symposium co-chair for the 2016 edition of the ACM DEBS conference. Since 2007 he has always been deeply involved in the activities of several EU funded projects in the area of Smart cities, home automation, ubiquitous computing, and collaborative security.

Roberto Beraldi received his Ph.D. in computer science from University of Calabria, Italy in 1996. Until 2002 he worked at Italian National Institute of Statistics. Since then he has been with the Department of Computer, Control, and Management Engineering at Sapienza University of Rome, Italy. He has published more than 70 papers in journals and main conferences related to wireless networks. His current research interest includes mobile and cloud computing, wireless networks, and distributed system.

Mario Paolucci, Ph.D. in "Telematics and Information Society" (2005), is a researcher at LABSS (Laboratory for Agent Based Social Simulation, http://labss.istc.cnr.it), ISTC CNR (Institute for Cognitive Science and Technology), Rome, where he collaborates to research since 1999. He is studying and applying multiagent-based social simulation and agent theory to understand social artifacts, in particular Reputation, Norms, Responsibility, and the cultural evolutionary mechanisms that support them. His publications include a book on reputation with Rosaria Conte and articles on JASSS, Adaptive Behavior, the International Journal of Cooperative Information Systems, the International Journal of Approximated Reasoning, and Scientometrics. He is the coordinator of the CNR workgroup on "Artificial Societies and Social Simulation" and a member of the PEERE COST initiative on Peer Review; he has been the scientific coordinator of the eRep "Social Knowledge for e-Governance" FP6 project, managing four international partners and a budget of 1.7MEuro. He has organized the first (ICORE'09) and the second (ICORE'11) International Conference on Reputation; the First International Workshop on Agent-Based Social Simulation and Autonomic Systems; and several other scientific events.