Distributed Event Aggregation for Content-based Publish/Subscribe Systems

Navneet Kumar Pandey
University of Oslo, Norway

Kaiwen Zhang
University of Toronto, Canada

Stéphane Weiss
University of Oslo, Norway

Hans-Arno Jacobsen
University of Toronto, Canada

Roman Vitenberg
University of Oslo, Norway

ABSTRACT

Modern data-intensive applications handling massive event streams such as real-time traffic monitoring require support for both rich data filtering and aggregation. While the pub/sub communication paradigm provides an effective solution for the sought semantic diversity of event filtering, the event processing capabilities of existing pub/sub systems are restricted to singular event matching without support for stream aggregation, which so far can be accommodated only at the subscriber edge brokers.

In this paper, we propose the first systematic solution for supporting distributed aggregation over a range of time-based aggregation window semantics in a content-based pub/sub system. In order to eschew the need to disseminate a large number of publications to subscribers, we strive to distribute the aggregation computation within the pub/sub overlay network. By enriching the pub/sub language with aggregation semantics, we allow pub/sub brokers to aggregate incoming publications and forward only results to the next broker downstream. We show that our baseline solutions, one which aggregates early (at the publisher edge) and another which aggregates late (at the subscriber edge), are not optimal strategies for minimizing bandwidth consumption. We then propose an adaptive rate-based heuristic solution which determines which brokers should aggregate publications. Using real datasets extracted from our traffic monitoring use case, we show that this adaptive solution leads to improved performance compared to that of our baseline solutions.

Categories and Subject Descriptors

C.2.4 [**COMPUTER-COMMUNICATION NETWORKS**]: Distributed Systems—*Distributed applications*

Keywords

Pub/Sub systems, Distributed aggregation, Distributed Adaptation

1. INTRODUCTION

DEBS'14, May 26–29, 2014, MUMBAI, India.

ACM 978-1-4503-2737-4/14/05 ...$15.00.
http://dx.doi.org/10.1145/2611286.2611302.

Modern distributed systems and applications require and provide support for aggregation of data [3, 1, 19, 21]. In this paper, we introduce mechanisms for supporting aggregate subscriptions in the context of the popular pub/sub messaging paradigm [7, 12, 22]. Pub/Sub is widely used in business process execution [26], workflow management [27], business activity monitoring [11], stock-market monitoring [33], selective information dissemination and RSS filtering [29], complex event processing for algorithmic trading [23], social interaction [31], and network monitoring and management [11].

In order to achieve scalability and high throughput, pub/sub has traditionally focused on performance over extended functionality. In particular, aggregation computations are not supported by typical content-based pub/sub systems, in contrast to other event processing paradigms, such as stream processing. While providing support for aggregation in pub/sub is desirable, it poses a unique set of challenges, which we further describe below.

Data-intensive applications for pub/sub can often benefit from aggregation over a filtered stream [10, 32, 13, 8, 35]. As an illustrating scenario, we consider an intelligent transport system (ITS) (see Fig. 1) within the framework of the Connected Vehicle and Smart Transportation (CVST) project [24]. This system disseminates data from road sensors and crowdsourced mobile users to ITS server farms to coordinate traffic flow and invoke the appropriate agents on demand. In addition, smart vehicles are connected to each other (Vehicle-to-Vehicle communication) to assist drivers and mitigate the chance of collisions [5].

This application requires a broad variety of filtering capabilities in order to allow each data sink to express its interests. Furthermore, the heterogeneous and mobile nature of the agents calls for a loosely coupled communication paradigm. These properties indicate that a pub/sub communication substrate (specifically, a content-based one) would be a good fit for this application.

At the same time, this application involves a large quantity of queries related to real-time monitoring that require aggregation over time-based windows. For instance, road sensors output the speed of every passing vehicle along a certain road segment during a certain time period [3]. These data points should be aggregated to compute the average speed over that segment. Another possible query is the count of the number of cars moving slowly (e.g., below 60 km/h) on the highway during a certain time window. Having the count here is beneficial for traffic redirection to alleviate congested highways.

Supporting aggregation in pub/sub, however, leads to a new set of challenges. The solution should be compatible with existing pub/sub systems. Unlike most other substrates for distributed aggregation, pub/sub aggregation should support large-scale data distribu-

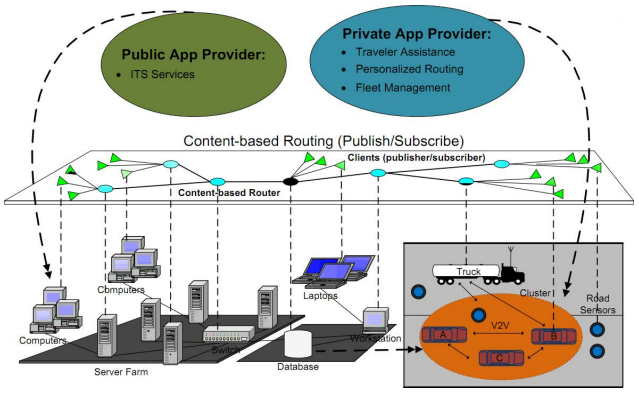


Figure 1: Conceptual CVST architecture

tion and dynamic loosely coupled matching between data sinks and sources. It should be integrated with established pub/sub infrastructures, such as hierarchies of message brokers, which impose their own logic for routing and flow control.

Since the same publication data may contribute to many aggregate subscriptions, which are spread over dispersed data sinks, an important aspect of aggregation in pub/sub is optimizing data flows from multiple sources to multiple sinks across brokers. This emphasis is different from data aggregation in other types of distributed systems, where the focus is on an efficient query processing engine for complex event patterns which scales well in terms of the number of events and queries. The challenge of optimizing data flows is exacerbated by the fact that the same data can be used for aggregate and regular (non-aggregate) pub/sub subscriptions. Finally, irregular event rates at the publishers, coupled with dynamic content-based matching and injection of subscriptions on the fly make it difficult to predict data flows in advance. This necessitates adaptive solutions with respect to aggregating and forwarding the information.

In this paper, we show the need for adaptation in the distribution of aggregation in pub/sub by considering two extreme baselines. First, we propose to aggregate at the subscriber edge brokers. This solution is the straightforward way to support aggregation where publications of interest are collected at the subscriber's end point, which requires the pub/sub system to deliver the entire stream of matching publications. Second, we consider an alternative approach that aggregates data at the publisher edge brokers: any broker directly attached to a publisher computes partial results over all matching publications received, while brokers downstream merge partial results coming from different upstream brokers. We show that, depending on the pub/sub parameters, which can vary over time (e.g., number of subscriptions, publication matching rate, etc.), one approach can outperform the other.

As our solution, we propose an adaptive algorithm, which measures the rate of publications and notifications and switches between publication forwarding and aggregation computation according to a cost model. The protocol we propose is lightweight in nature as it does not require any changes to the system model of current pub/sub implementations. Any additional information to be exchanged by brokers is piggybacked onto existing traffic.

In summary, the contributions of this paper are as follows:

- We provide two baseline algorithms to support aggregation using pub/sub. The first algorithm employs subscriber-edge aggregation (late approach), where the entire matching publication stream is disseminated to the subscriber edge brokers where the aggregation results are computed. The second baseline adopts the early aggregation approach by performing distributed aggregation at the publisher edge brokers and forwarding partial results downstream.
- We demonstrate that neither baseline is dominant, since their relative performance can vary according to the pub/subparameters. We identify the major parameters which affect the communication cost. In particular, the often dynamically varying publication and notification rates, significantly impact the aggregation computation.
- We therefore propose an adaptive aggregation framework. Our framework is reconfigurable and allows aggregation to be performed anywhere in the broker overlay without compromising consistency. We propose one adaptive solution based on our framework which considers the flow rates in its cost model to determine the appropriate placement (and reconfiguration) of aggregation computations in the overlay topology.
- We evaluate the performance of our adaptive solution and the baselines using real data traces collected from our traffic monitoring use case along with a dataset from a stock market application. We show that the performance of both baselines vis-à-vis each other depends on the properties of the workload, while the adaptive solution outperforms both in every situation. We also offer a sensitivity analysis for the major parameters affecting aggregation.

2. BACKGROUND

In this section, we describe the content-based pub/sub model before providing a taxonomy of aggregation semantics relevant for this model.

2.1 Publish/Subscribe model

Our work focuses on content-based pub/sub using advertisement-based routing [7, 12]. In this model, each publisher and subscriber is attached to a single broker, called an edge broker. Brokers are in charge of forwarding publications from publishers to the appropriate subscribers and are connected in an overlay.

Routing paths are initialized through the use of advertisements, which are flooded through the network. Subscriptions are matched against those advertisements to build a delivery path from the originating publisher to the subscribers. Publications are then matched against the subscriptions and forwarded down the appropriate nodes.

Content-based matching allows for predicate-based filtering on a per-publication basis. In other words, matching is stateless: each publication is matched in isolation against the subscriptions and is not affected by the existence of other publications.

2.2 Aggregation taxonomy

Aggregation refers to the summarization of multiple data points into a compact representation [21]. Example aggregation functions include `Average`, `Minimum`, `Sum` and `Top-k`.

Decomposability: An aggregation function is considered decomposable if the same result can be obtained by aggregating the partial results over subsets of the original input. Essentially, decomposability allows the aggregation computation to be distributed by partitioning the input set. Amongst decomposable functions, there is a distinction between *self-decomposable* and *indirectly decomposable* functions.

Self-decomposable functions can be directly applied on subsets of the input and a merge function is used to combined results of subsets. For example, given two sets of publications X and Y, `Count` operator is self-decomposable since `Count`($X \cup Y$) = `Sum` (`Count`(X), `Count`(Y)). In this case, `Sum` is used as the merge operator. *Indirect decomposable* functions are not self-decomposable but they can be transformed into an expression containing only self-decomposable functions. For instance, `Average` is indirectly decomposable: it is not self-decomposable but it can be expressed as the `Sum` divided by the `Count`. Non-decomposable functions

cannot be broken down: the result can only be derived by taking the entire input set and processing it at once. For instance, `Distinct Count` (i.e., cardinality of a set) is non-decomposable: we require the entire set of elements to avoid counting duplicates.

Duplicate sensitivity: A duplicate sensitive aggregation operation is affected by duplicate elements while an insensitive operation is not. For instance, `Maximum` is a duplicate insensitive operation while `Count` is not. To support duplicate sensitive operators, we need to ensure that each element is aggregated exactly once. This is a challenge for distributed environments where the aggregation of an element can occur at different locations.

2.3 Window parameters

Given a stream of values, aggregation functions are computed repeatedly over a sequence of windows, each with a start point and duration. In this work, we focus on windows defined using time (*time-based* windows) although other semantics exist [21]. The start point is determined by the window shift size and the start point of the previous window. The window shift size (δ) and duration (ω) are expressed either in terms of time or number of values. If $\delta < \omega$, consecutive window ranges will be overlapping (*sliding*). If $\delta = \omega$, the window is called *tumbling*, otherwise ($\delta > \omega$), it is called *sampling*.

3. PROBLEM STATEMENT

In order to enhance pub/sub with support for aggregation, we extend the semantics of a content-based subscription while retaining the standard format and semantics for publications. Specifically, we allow a content-based subscription to include an aggregation function along with the parameters of a time-based window, in addition to the standard predicates. We support the entire range of aggregation functions w.r.t. decomposability and duplicate sensitivity. We empower the application with the capability to combine aggregate subscriptions that include an aggregation function with regular subscriptions that do not.

For each aggregation subscription, the pub/sub system must repeatedly send relevant aggregated results to the subscriber. Each result is tagged with a time window to which it belongs and computed based on matching publications contained in this window.

The extended pub/sub system must face the inherent problem of aggregation in asynchronous distributed environments, namely the lack of synchronization between the publishers, subscribers, and brokers. First of all, the assignment of publications to windows is based on real-time, as it is commonly accepted in distributed aggregation. When the clocks are not synchronized, a publication may be assigned to an "incorrect" window. In practice, the problem is not significant because a typical clock drift is of a lower order of magnitude compared to the window shift size. It is still possible, however, that a publication issued close to the boundary between two windows will be assigned to a neighboring window instead.

Propagating matched publications to the subscriber may also take a non-negligible amount of time in distributed environments, such as pub/sub. In particular, it is important to timestamp publications at the source (or at least at a publisher-edge broker), otherwise a window may have shifted by the time they arrive at the subscriber so that the system may erroneously assign them to an incorrect window. Observe that without strict bounds on the propagation and processing times, the service cannot guarantee that the aggregate computation will end by a certain point in time because additional publications from distance brokers may continue to trickle in. Again, the problem is mitigated in practice by the fact that in reality, the propagation and processing times are much shorter compared to window durations, as we also illustrate in Section 7. In this work, we opt to deliver a stream of partially aggregated results to the subscriber, each result being delivered as soon as it becomes available at the subscriber edge broker. It is possible to use an alternative delivery policy; the design choice of a delivery policy is a common element of distributed aggregation schemes and, thus, orthogonal to the specific focus of this paper.

The distinguishing flavor of our study is that we are investigating the problem or reducing the communication cost of aggregation support in pub/sub. Specifically, we explore various strategies for distributing the computation for each aggregation subscription across a given overlay of pub/sub brokers. Since the traffic in pub/sub is typically dominated by publications, the choice of strategy has a profound effect on the number of messages with publications and with partially aggregated results. The choice is compounded by the lack of global knowledge about the overlay of brokers, dispersion in the location of subscribers and publishers, dynamism in the subscriptions, co-existence of aggregation and regular subscriptions, and unpredictable fluctuations in the rate of matching publications, as further described in Section 6.2.1.

In addition to optimizing the number of messages, the solution should impose only limited additional delay compared to the delivery latency of publications matching regular subscriptions. There exist two factors contributing to additional delay for aggregation subscriptions, namely, computation and collection times on individual brokers. Each broker may decide to collect publications and partial aggregation results within a given window before applying the aggregation function and sending the results further. Longer collection times on individual brokers may reduce the number of messages but adversely affect delivery latency so that the design should take this tradeoff into consideration.

4. RELATED WORK

In general, distributed aggregation in data-intensive applications focuses on two aspects: processing and distributed data flows. The approach in Cayuga [10] and [32] concentrates on processing and provides an efficient centralized query processing engine for complex event patterns, which scales well in terms of the number of events and queries. Brenna et. al. [6] provide a distributed extension of Cayuga where the matching process spans multiple machines. While our primary focus is on reducing the number of messages exchanged among the brokers, minimizing the communication cost is beyond the scope of consideration in [6]. In fact, in situations when filtering and computation for aggregation are lightweight, spreading them over several machines may bring little processing advantage, yet incur noticeable communication cost. It should be mentioned that the techniques used in [6] are complementary to our contributions and can be combined in the context of a single system.

Meta [36] and gridStat [1] address the challenge of processing distributed data flows by using a control layer for efficient routing. However, this control layer is incompatible with elaborate routing protocols used in pub/sub, in particular it does not consider regular subscriptions. It also does not align with our goal of providing a lightweight and pluggable aggregation component for existing pub/sub systems.

In contrast to aggregating the content of publications, ASIA [14, 13] provides a service that aggregates metadata of a pub/sub system such as subscriptions count. Our approach emphasizes content aggregation and faces some unique challenges as compared to metadata aggregation. First, the notification rate for content aggregation is higher compared to the rate of updates in system parameters. Second, as opposed to metadata aggregation, our traffic use case scenario requires efficient distribution of both regular and

aggregation subscriptions where the same set of publications can match subscriptions of both types, as discussed in Sect. 3. Optimizing the communication cost in presence of these challenges requires an adaptive solution. While adaptation techniques have been proposed for pub/sub in the context of efficient routing [30] and overload management [20], the need for an adaptive solution for aggregation is a unique challenge of our work.

Aggregation in distributed pub/sub systems shares some common challenges with distributed stream processing systems (DSPS). This includes aggregating data from distributed publication sources as well as processing and serving multiple queries [17]. Publication sources are a priori known in DSPS systems before the query plans are computed and the placement of operators is determined [39]. In particular, computing a query plan requires a global view of the overlay topology. Conversely in pub/sub systems, the location of sources and sinks are changing over time and broker visibility is restricted to a neighbourhood, which prohibits the use of static query plans. Moreover, source nodes in a DSPS environment are assumed to continuously produce data for long durations while, in pub/sub, publishers generate events intermittently at arbitrary points in time making query plan optimizations ineffective for pub/sub. However, some of the optimization techniques from DSPS such as multi-query optimization [25, 37] can be utilized in pub/sub. For example, the notification schedule synchronization technique in [15] that optimizes message traffic by batching is orthogonal to and compatible with our approach.

A number of distributed schemes for disseminating aggregate information over unstructured P2P overlays has been proposed in the literature [21, 34, 38, 35, 28, 19, 2]. These techniques usually create a topic for each subscription. Chen et al. [9] propose an aggregation technique for topic-based pub/sub systems that utilizes the concept of event gatherers. These event gatherers are located at the graph center of the corresponding routing tree for each topic. Even though these approaches can be extended for content-based pub/sub dissemination, the extensions may suffer from scalability issues [4]. Moreover, graph-center based approaches are incompatible with pub/sub routing protocols.

As an alternative to integrating aggregation within pub/sub a standalone aggregation engine external to the pub/sub layer could be deployed. However, as mentioned in ASIA [13], it would hamper QoS for regular publications. Most standalone approaches create their own aggregation and distribution trees for efficient computation and dissemination [39]. Similar to the graph-center-based approaches, these aggregation trees may not always align with the routing overlay created by pub/sub.

5. DISTRIBUTED AGGREGATION FOR CONTENT BASED PUB/SUB

In this section, we describe the main components of our proposed solution for aggregation in pub/sub. These components are used by different deployment strategies that we describe in Sect. 6 so that they can be regarded as building blocks for the purpose for composing deployment strategies. The methods used in the components are general and applicable to all three operator types (i.e. self-decomposable, indirect decomposable and non-decomposable). We start by presenting the flow of a subscription for aggregation before detailing the flow of a publication.

5.1 Subscription process flow

Aggregate subscription contains a standard conjunction of predicates, the aggregation operator, the attribute over which aggregation is performed, and the window parameters (ω, δ) as shown in example 1.

Example 1. (**Aggregate Subscription**) A subscription S to the moving average over one hour at every 10 minutes for the stock symbol '*IBM*' would be expressed as S={sym='*IBM*', op='`Average`', par='*value*', ω ='*1h*', δ ='*10 min*'}.

For each aggregate subscription, a broker which receives it performs the following two steps.

1. **Subscription transformation:** As presented in Sect. 2, the aggregation operators can be self-decomposable, non-decomposable and indirect decomposable type. Subscriptions having operator other than self-decomposable are first transformed before being propagated further to other brokers. Subscriptions with indirect decomposable operator get transformed into subscriptions with self-decomposable operators, whereas non-decomposable are turn into auxiliary operators.

Example 2. (**Aggregate subscription type**) Following are the examples of aggregate subscriptions containing operator of different types:

- Self-decomposable: S_1={sym='*IBM*', op='`Sum`', par='*value*', ω='*1h*', δ='*10 min*'}
- Indirect decomposable: S_2={sym='*IBM*', op='`Avg`', par='*value*', ω='*1h*', δ='*10 min'*}
- Non-decomposable: S_3={sym='*IBM*', op='`Median`', par='*value*', ω='*1h*', δ='*10 min*'}

In this subscription transformation step, the non-decomposable subscription S_3 is transformed to S_3':
{sym='*IBM*', op='`Fetch`', par='*value*', ω='*1h*', δ='*10 min'*} where "`Fetch`" is an auxiliary operator that batches all matching publications. The subscription S_2 with indirect decomposable operator (i.e. `Average`) gets transformed into a subscription containing self-decomposable operators (`Sum` and `Count`). S_2 becomes S_2': {sym='*IBM*', op='`Sum`', op='`Count`', par='*value*', ω='*1h*', δ='*10 min*'}.

2. **Subscription dissemination:** Our pub/sub system follows the advertisement-based forwarding model [7]. Subscriptions that match advertisements are propagated towards the advertisers, forming publication delivery trees rooted at each publisher. The procedure for matching aggregate subscriptions against registered advertisements is equivalent to that for regular subscriptions.

5.2 Publication process flow

If we consider the general flow of aggregating publications in a pub/sub system, it should start by matching the publication against aggregate subscriptions. If a match is found, the publication can be either slated for local computation of partial intermediate aggregate results (IRs) or forwarded to a downstream broker, which would need to repeat this procedure. We refer to these steps as *Publication Filtering Procedure (PFP)*. Since we provide aggregation for *time-based* windows, IRs need to be computed periodically based on enqueued publications and forwarded to a downstream broker. These steps are performed by the *Initial Computation Procedure (ICP)*. Finally, IRs received by downstream brokers need to be combined and forwarded further repeatedly until the final result gets computed by the subscriber edge broker. This is done by the *Recurrent Processing Procedure (RPP)*.

Note in PFP, the broker has a choice to enqueue a publication for local computation or forward it further. This choice can be externally configurable in static deployment strategies (see Sect. 6). In adaptive deployment strategies, each broker can dynamically and autonomously (or in coordination with other brokers) decide to operate in the aggregating or forwarding mode. In the rest of this

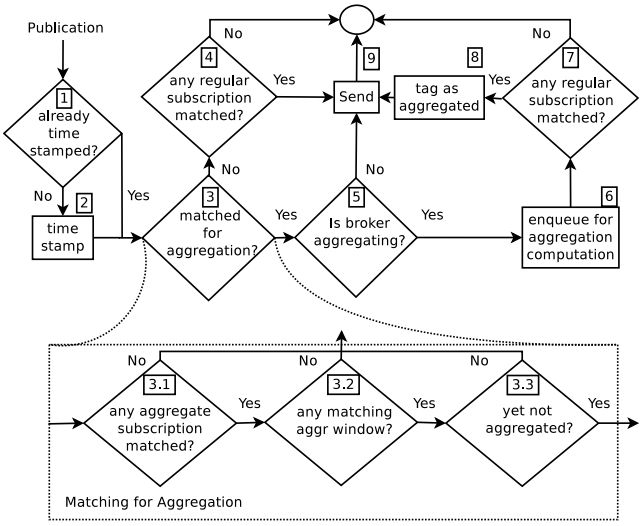


Figure 2: Publication process flow

paper, we will use the notation of PFP (aggregating) and PFP (forwarding), or simply PFP when the distinction is not important.

5.2.1 *Publication Filtering Procedure (PFP)*

This procedure is executed upon receiving a publication at every broker which has an aggregate subscription registered. The stages of the procedure are presented in Fig. 2).

(1) **Publication time-stamping (stage 1 & 2):** When a broker receives a publication directly from a publisher, the broker tags it with its local time. This publication time is subsequently used by this broker or another broker downstream to assign the publication to aggregation windows. As mentioned in Sect. 3, we assume that clocks in all brokers are loosely synchronized. Consequently, a publication would typically be assigned to the same window regardless of the broker where the computation is performed.

(2) **Subscription matching (stage 3 & 4):** Upon receiving a publication, the broker first matches it against aggregate subscriptions based on the conditional predicates. This matching is identical to the one performed for regular subscriptions using the pub/sub matching engine. For the specific case of sampling window, the broker checks whether the publication matches to a window. Finally, to support duplicate sensitive operators, a publication is taken into account when performing the aggregate computation only if this publication has not been aggregated previously at any upstream broker. Observe that when a publication matches an aggregate subscription and another regular subscription, an upstream broker may both include the publication into IRs and forward it. To provide relevant information to the downstream brokers, a broker that aggregates a publication tags it as *aggregated* (see the forwarding publication step). If the publication matches only a regular subscription but not any aggregate subscription, it gets forwarded. If it does not match any subscription at all, the publication is dropped and the procedure ends.

(3) **Processing (stage 5):** If the broker receiving the publication is not assigned to perform aggregation, the publication is simply forwarded to the next broker. Otherwise, the broker determines to which window the publication should be added. To achieve this goal, the notion of Notification Window Ranges (NWRs) is used. We define an instance of Notification Window Range (NWR) as a triplet of $\langle$*subscription*, *start-time*, *duration*$\rangle$. For a matched aggregate subscription, depending upon its window and shift size, zero

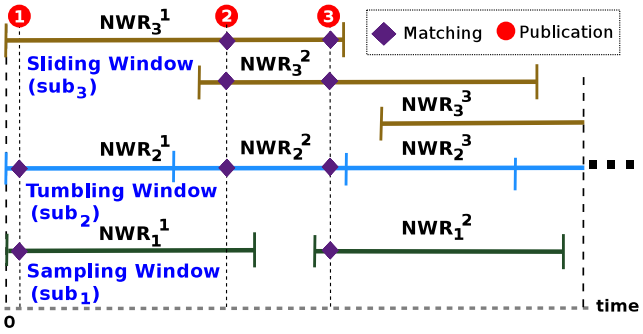


Figure 3: Notification Window Ranges (NWRs)

or more notifications are generated (see Fig. 3). NWRs are used to manage these notifications.

Example 3. (**Notification Window Range**) Fig. 3 illustrates the concept of NWR. In this example, we consider three aggregate subscriptions sub_1, sub_2 and sub_3 which are respectively for sampling, tumbling and sliding window. The first publication (around time 0) matches the subscriptions sub_1 and sub_2 only. The second publication matches sub_2 and sub_3. As sub_3 has a sliding window, this publication matches two NWRs for sub_3 (NWR_3^1 and NWR_3^2) and one NWR for sub_2 (NWR_2^2). Finally, the last publication matches all three subscriptions and contributes to four NWRs.

(4) **Forwarding publication (stage 6-9):** If a publication that has been aggregated at this broker matches a regular subscription as well, then the broker tags the publication as "aggregated" (as described in the subscription matching step) and forwards it to the downstream broker.

5.2.2 *Initial Computation Procedure (ICP)*

ICP runs in a separate periodic process on any broker that performs PFP (aggregating). It consists of the following steps:

(1) **Computing:** In this step, the broker computes the aggregated value for one NWR out of the received publications matching this NWR, using the aggregation operator. We refer to this value an an Intermediate Result (IR).

The decision when to compute the result for an NWR reveals a trade-off between notification delay and communication cost. If a broker has received all the publications for an NWR before processing them, it will send only one IR message to the downstream broker. Publications arriving after the end of the computation will result in additional IR messages, which does not violate the semantics but which has an adverse impact on performance. The problem is that in reality, due to a processing delay on upstream brokers and a propagation delay in the network, a broker is likely to continue receiving publications matching an NWR after the NWR ends. The amount of extra messages can be minimized by deferring the computation after the end of the NWR. However, delaying the computation for a long duration incurs high delay in notifying the subscriber. In our solution, we address this trade-off by delaying the computation by the amount of time, which is equal to the propagation and processing delay observed for publications matching past NWRs for this subscriptions. Each broker estimates this delay using the difference between its local clock and the timestamp of publications received for this NWR.

Example 4. (**Computing aggregation results**) For the subscriptions presented in example 2, we assume that five publications are generated in a single 10 minutes window for the symbol '*IBM*' with a '*value*' attribute of 4, 2, 5, 9 and 5. Assuming that these

publications arrive before the computation is performed, the intermediate results produced out of all five publications are 25 for S_1, $(\texttt{sum} = 25, \texttt{count} = 5)$ for S_2, and $\{4, 2, 5, 9, 5\}$ for S_3. In contrast if, for example, the two last publications are delayed for longer than the waiting time, the broker first produces the following IRs: 11 for S_1, $(\texttt{sum} = 11, \texttt{count} = 3)$ for S_2 and $\{4, 2, 5\}$ for S_3. Then, it produces 14 for S_1, $(\texttt{sum} = 14, \texttt{count} = 2)$ and $\{9, 5\}$ for S_3.

(2) **Propagating the results:** For the purpose of propagating towards the appropriate subscribers, IRs are embedded in messages of a special type, which we henceforward call *IR messages*. These messages are propagated using the regular routing mechanisms employed by the pub/sub system.

5.2.3 Recurrent Processing Procedure (RPP)

Intermediate brokers may receive several IR messages for the same NWR. These IRs may be generated by the same broker, as explained in Sect. 5.2.2, or from multiple source brokers concurrently. Since these IRs belongs to the same NWR, the broker can combine them into a single new IR. Processing of IRs differs from regular processing as IRs contain partial results, window information etc. For these reasons, an intermediate broker follows a separate processing procedure which we refer to as Recurrent Processing Procedure (RPP) (as it applies repeatedly at each broker in the notification route, in contrast to ICP). The goal of RPP is to reduce the number of messages by merging incoming IRs for the same NWR. Because of the processing and propagation delays, as well as the lack of synchronization among brokers, RPP does not ensure that all IRs are merged into a single message but it tries to reduce the number of messages exchanged. RPP consists of the following steps:

(1) **Finding the matching NWR:** Upon receipt of an IR message, a broker finds the matching NWR for the IR.

(2) **Buffering the intermediate results:** IRs matching the same NWR may not arrive simultaneously. Therefore, IRs are buffered while the broker waits for additional IRs for the same NWR. Unfortunately, the broker cannot be aware of the existence of other IR messages before receiving them. In addition, there is no upper bound on the delay before receiving an IR. Delaying an IR for a large amount of time may increase the chances of collecting several IRs which reduces the number of messages sent at the cost of a higher delay for notification to the subscriber. Note that, since the RPP is applied repeatedly on downstream brokers, the delay to receive an IR depends on the number of upstream brokers performing RPP, which is not known by a broker due to a loosely coupled nature of pub/sub. To mitigate the notification delay while trying to reduce messages, we propose to delay processing each IR publication by a configurable time interval which we refer to as *collection delay*. We investigate the impact of this parameter in Sect. 7.

(3) **Synchronizing and merging IRs for the same NWR:** After expiration of the *collection delay*, collected IRs matching the same NWR are merged using an operator which can differ from the actual aggregation operation (see Sect. 2). For instance, two streams of intermediate `Count` results can be combined together by summing the counts of the matching windows.

(4) **Propagating merged intermediate results:** A merged IR is then routed to the downstream brokers. If the next hop is also an intermediate broker, RPP is invoked again at that broker to collect and further merge IRs. Therefore, a stream of IR messages goes recurrently through RPP. At the end, the result is converted to its final form before being delivered to the client.

(5) **Transformation of results and notification:** For indirect decomposable and non-decomposable operations, results obtained from the previous step must be transformed back to fit the subscription issued by the client.

Example 5. (**Transformation of results**) In this example, we consider the IRs and subscriptions from example 4. In the result transformation step, for S_2, the average of 5 is computed as $(\texttt{sum} = 25, \texttt{count} = 5)$ out of S'_2. Similarly for S_3, the result 5 is computed from publications obtained for S'_3. The final step consist of sending the results to the subscribers, 25 for S_1, 5 for S_2, and 5 for S_3.

6. DEPLOYMENT STRATEGIES

In this section, we discuss several deployment strategies. The first strategy is to aggregate at the subscriber edge broker which we refer to as LATE aggregation. Then we consider aggregating at the publisher brokers which we call EARLY aggregation. Finally, we observe that none of these strategies outperforms the other one in all settings, and we propose an adaptive strategy in which each broker dynamically makes an independent decision about whether to aggregate publications or simply forward them. We call this approach ADAPTIVE aggregation.

6.1 Static distributed aggregation

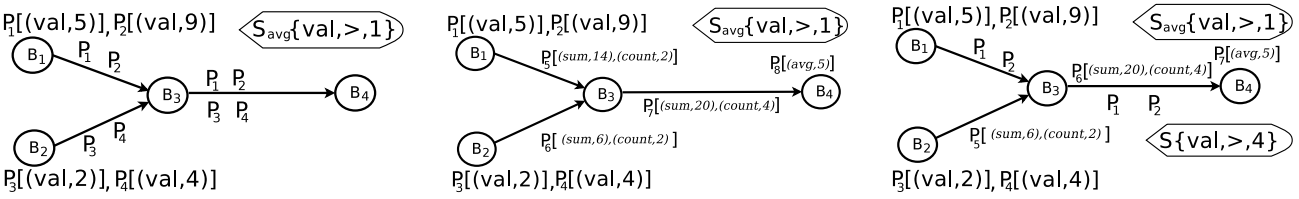
In the LATE aggregation strategy, there is only a single broker for each subscription that is assigned to aggregate, namely the subscriber edge broker directly attached to the subscriber. This broker executes PFP (aggregating) and ICP. All other brokers processing publications for this subscription execute PFP (forwarding) for the purpose of early filtering. There is no need for RPP in LATE aggregation.

Example 6. To illustrate the behavior of this solution, consider the example described in Fig. 4(a). A subscriber attached to broker B_4 sends a subscription requesting the average value for a stock symbol IBM: {sym='*IBM*', op='`Average`', par='*value*', ω='*1h*', δ='*10 min*'}. Upon receiving this subscription, B_4 sets itself to the aggregate mode and starts ICP. Then, as explained in Sect. 5.1, this subscription is transformed into {sym='*IBM*', op='`Count`', op='`Sum`', par='*value*', ω='*1h*', δ='*10 min*'} and then propagated towards the publishers.

In this example, as shown in Fig. 4(a), a publisher at broker B_1 generates two publications P_1 and P_2 having values '5' and '9' respectively. These publications are timestamped by B_1 according to the first step of PFP. Then, B_1 performs the second step of PFP which is filtering for aggregate subscriptions. For the sake of simplicity, we assume that these publications match a single NWR of the considered subscription. As B_1 is not assigned to aggregate, these publications are forwarded to B_3. Similarly, the second publisher at B_2 also generates publications P_3 and P_4 having respectively '4' and '2' as values for the same NWR. All these four publications are routed based on their content until they reach broker B_4. Upon reception of P_1, P_2, P_3 and P_4, the execution of the ICP on B_4 results in the sum '20' and the count '4'. These results are then used to generate the notification containing the average '5' for the subscriber.

In the EARLY aggregation strategy, every publisher-edge broker for a subscription (i.e., a broker that has any advertisement matching this subscription from a publisher directly attached to it) is set to the aggregate mode and executes PFP (aggregating) and ICP. Additionally, all brokers processing IRs for this subscription execute RPP.

Example 7. Figure 4(b) shows how EARLY aggregation works on a simple scenario. Similarly to example 6, an aggregate subscription issued at broker B_4 is forwarded to broker B_3. Since broker



(a) LATE aggregation (Naive approach) (b) EARLY aggregation (c) ADAPTIVE aggregation

Figure 4: Aggregation techniques

B_3 receives IRs for this subscription, it enables RPP. Finally, upon receiving this subscription, brokers B_1 and B_2, having a publisher attached to them, assign themselves to aggregation and starts ICP.

Publications P_1 and P_2 are processed by broker B_1 which sends an IR message P_5 with the following IR: [(sum=14, count=2)]. Similarly on broker B_2, an IR message P_6: [(sum=6, count=2)] is produced. When broker B_3 receives P_5 and P_6, it applies RPP and generates the result P_7: [(sum=20, count=4)]. Upon reception of this IR, broker B_4 notifies the subscriber. In this scenario, the overall number of messages is reduced from 8 to 3 when comparing EARLY to LATE aggregation.

6.2 Adaptive aggregation deployment

In this section, we motivate the need for an adaptive deployment strategy and we provide details about our adaptation engine used to reconfigure the aggregation system. Since every broker can decide to aggregate, by default every broker registering an aggregate subscription executes all three procedures PFP, ICP, and RPP. When adapting, a broker can switch from PFP (aggregating) to PFP (forwarding) or vice versa.

6.2.1 Motivation for adaptation

EARLY aggregation greedily tries to reduce the number of messages exchanged among brokers by performing the aggregation as close to the publishers as possible. However, our experiments (see Figs. 6(c) and 6(d)) show that EARLY aggregation generates more messages than LATE aggregation in some scenarios. The comparative performance of EARLY vs LATE aggregation depends upon several factors such as:

- *window shift size:* In some scenarios such as sliding NWRs, a single regular publication may lead to multiple IR messages. For instance, let us consider a window duration of 10 seconds and a window shift size of 1 second. A single matching publication can generate up to ten notifications, one for each NWR. Reducing the window shift size tends to favor LATE aggregation.
- *matching publication rate:* The total number of messages exchanged among brokers is significantly influenced by matching publication rate. For instance, in the above example, 20 matching publications within the same 10 second window would generate only 10 notifications. When the matching publication rate increases, EARLY aggregation gets advantage compared to LATE aggregation.
- *number of aggregate subscriptions:* The probability of a publication to match several aggregate subscriptions increases with the total number of subscriptions. If a publication is matched by multiple subscriptions, EARLY aggregation generates at least one IR message for each of the matched subscriptions. In contrast, LATE aggregation only sends one publication through. Therefore, increasing the number of aggregate subscriptions favors LATE aggregation.
- *number of regular subscriptions:* If a publication matches a non-aggregate (regular) and an aggregate subscription, both the publication and corresponding IR are sent in the EARLY approach. Higher amount of regular subscriptions increases the chance of overlap with aggregate subscription. Finally, increasing the number of regular subscriptions tends to make LATE aggregation generate fewer messages than EARLY aggregation.

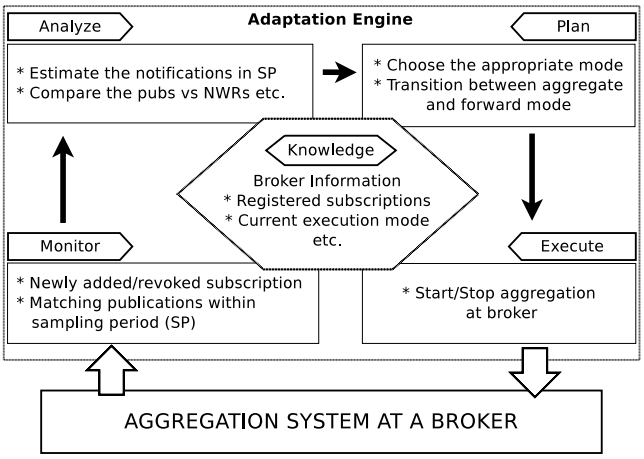


Figure 5: Adaptation tasks

Most above factors, except for the matching publication rate, tend to favor LATE aggregation when the value of the corresponding parameter increases. In contrast, a high rate of matching publications benefits EARLY aggregation. As these factors dynamically change across the brokers and over time, one approach cannot outperform the other in all situations. Therefore we propose an ADAPTIVE aggregation engine which continuously monitors and weighs in these factors and switches to the appropriate aggregation mode as needed.

6.2.2 Adaptation engine

Our adaptation engine, illustrated in Fig. 5, is based on the MAPE-K model presented in [16] which represents Monitor, Analyze, Plan and Execute along with the shared Knowledge-base. This model is general and can accommodate different cost models. It should be noted that performing aggregation at a broker consumes resources such as CPU and memory which may impact various Quality of Service (QoS) parameters such as the notification delay. Our specific cost model in this paper does not directly take these parameters into account because we primarily focus on reducing the communication cost. Our evaluation in Sect. 7 considers an indirect impact on the notification delay. It is possible to extend the cost model within our framework so as to explicitly take the amount of available resources (CPU, memory, bandwidth, etc) on a broker into account.

When adapting, each broker independently decides to run ei-

ther in *Aggregate* or *Forward* mode. If the number of publications which match only aggregate subscriptions is greater than the expected number of IR messages, the broker operates in the *Aggregate* mode, otherwise, it switches to the *Forward* mode.

Fig. 5 shows the MAPE-K tasks performed by a broker in order to adapt to the appropriate mode. The first task is to monitor the number of newly added or revoked subscriptions along with the rate of incoming matching publications over a sampling period. Next, the analyzer unit, based on the shared information, estimates the number of IR messages generated during the sampling period and compares it with the number of matching publications received within the same period. Note that the period of sampling is important as its affects the performance. We analyze the impact of the sampling period on message reduction in our experiments (see Sect. 7.3). The planner unit determines the appropriate mode (i.e., *aggregate* or *forward*) for the current setting. If the broker is not running in the appropriate mode, this unit proposes a transition. Finally, the execution unit adapts to this mode by starting or stopping aggregation on this broker.

Example 8. To illustrate the behavior of our ADAPTIVE solution, we extend example 7 by adding a regular subscription {sym= '*IBM*', value>'*4*'} issued at broker B_4. The resulting scenario is described in Fig. 4(c). Upon receiving these subscriptions, the adaptation engine of broker B_1 is notified of their arrival. Then, when P_1 and P_2 are received by broker B_1, its adaptation engine performs the analyze task and estimates that 3 notifications (P_1, P_2 and one IR) would be produced out of these 2 publications. Therefore, broker B_1 switches to the *Forward* mode and stops aggregating. In contrast, since the publications at broker B_2 match only the aggregate subscription, its adaptation engine will prefer to send one notification rather than two publications. Therefore, it switches to the *Aggregate* mode.

In this example, let us assume that broker B_1 is already in the *Forward* mode while broker B_2 is in the *Aggregate* mode. Therefore, broker B_3 receives publications P_1 and P_2 from broker B_1, and an IR message P_5 from broker B_2. On the one hand, if broker B_3 is in the *Aggregate* mode, it generates a new IR message P_6 sent along with P_1 and P_2. On the other hand, if broker B_3 is in the *Forward* mode, it sends publications P_1, P_2, and P_5. Finally, regardless of its mode, broker B_4 aggregates the received publications and runs RPP before notifying the subscriber. Note that, depending on the latency and the *collection delay*, broker B_4 may deliver either one or two notifications to the subscriber.

7. EVALUATION

Our evaluation is divided into two sections. First, we experimentally compare the performance of our proposed adaptive solution (called ADAPTIVE) to the two baseline approaches (called EARLY and LATE aggregation). We use the processing time and the total number of messages exchanged among brokers as the main performance metrics. Second, we conduct a sensitivity analysis with our ADAPTIVE solution for parameters specific to adaptation: the *collection delay* and the *sampling period*.

7.1 Experimental setup

We conduct our evaluation using two different datasets. The first dataset (called *Traffic*) contains real traffic monitoring data from the ONE-ITS service[1] that the service designers made publicly available at[2]. According to their description, the data were produced by 162 loop detectors located at major highways in Toronto and later transformed into publications. The resulting publications contain 12 distinct fields. The second dataset (called *Stock*) is from a stock market application (obtained from daily stock data of Yahoo! Finance) which is commonly used to evaluate pub/sub systems [8]. We used 62 stock symbols from the dataset with each stock publication containing 8 distinct fields. We keep the selectivity of subscriptions [18] to around 1% for the Traffic dataset and 2% for the Stock dataset.

Our deployment setup consists of a cluster of 16 servers, where 4 brokers act as core brokers forming a chain. We attach 3 edge brokers to each core broker. Out of those 12 edge brokers, 3 are exclusively dedicated to connecting publishers, 3 brokers to connecting subscribers, while the remaining 6 brokers have a mix of publishers and subscribers attached. For the Traffic dataset, we consider 162 loop detectors as publishers. Additionally, all publishers which correspond to loop detectors monitoring the same direction of the same highway are connected to the same broker. For the Stock dataset, we provide a separate publisher for each stock symbol.

For most of these experiments, we vary two parameters: the publication rate from 13 to 1080 pubs/s for the Traffic dataset and from 5 to 1033 pubs/s for the Stock market dataset, and the number of subscriptions from 45 to 990 for the Traffic dataset and from 90 to 810 for the Stock market dataset. We create a mix of aggregate and regular subscriptions with varying degrees of overlap for some experiments. By default, we keep a 50% ratio of aggregate subscriptions. Each aggregate subscription is assigned a shift size of 2 seconds ($\delta = 2$). Each experiment is running for 1000 seconds, which represents 500 aggregation periods. Finally, the window size for the Stock dataset is 2 seconds ($\omega = 2$) while for the Traffic it is 20 seconds ($\omega = 20$).

Our ADAPTIVE solution contains two additional parameters. One is *collection delay*, which is the amount of time a broker waits during the PFP to buffer IR messages. In our performance evaluation, this delay dynamically varies between 10 to 50ms through the course of an experiment. In our sensitivity analysis, the delay is set to a static value to evaluate its impact on the performance of the ADAPTIVE solution. The second parameter is the *sample window size*, which is used by the adaptation engine to predict future loads. This size is set to 2s during the performance evaluation, and is a variable in our sensitivity analysis.

7.2 Performance comparison

7.2.1 Number of messages

The main focus of this paper is to reduce the number of messages exchanged while supporting aggregation. We compare the number of messages generated by each of the three strategies presented in Section 6 while varying the number of subscriptions and the publication rate for both datasets.

Publication rate: Figures 6(a) and 6(b) show the number of messages exchanged while increasing the publication rate for the Stock and Traffic datasets, respectively. For both datasets, LATE generates less messages than EARLY at low publication rates, which supports the claims in Section 6.2.1. When the publication rate increases, this trend is reversed and EARLY aggregation generates significantly fewer messages than LATE.

ADAPTIVE adapts to the most efficient strategies between EARLY and LATE. For low publication rates, ADAPTIVE generates a similar number of messages as LATE for both datasets. However, for higher publication rates, ADAPTIVE generates more messages than EARLY, especially for the Stock dataset. We attribute this difference to estimation errors brokers make in thinking that aggregating

[1]http://one-its-webapp1.transport.utoronto.ca
[2]http://msrg.org/datasets/traffic

would generate more messages than forwarding a publication. As a consequence, we observe that brokers switch between the two modes frequently from 50 pubs/s to 300 pubs/s for Stock: a total of over 1000 transitions are recorded during this period. However, when the publication rate exceeds 500 pubs/s, the gap between ADAPTIVE and EARLY is diminished since increasing the publication rate eventually enforces the adaptation engine to opt for EARLY aggregation. Thus, we observe no more than 330 transitions at a rate of 1033 pubs/s. For the Traffic dataset, the adaptation engine performs better with only 284 transitions which leads to performance results closer to EARLY.

Number of subscriptions: Figures 6(c) and 6(d) show the number of messages while varying the number of subscriptions using the Stock and Traffic workloads, respectively. For Stock, LATE performs better than EARLY while the opposite is true for Traffic. This difference is accounted for by the choice of publication rate.

When the number of subscriptions increases, we observe that the difference in the number of messages between EARLY and LATE increases for Stock, with EARLY sending up to 1.8 times more messages than LATE. This observation is consistent with our expectations as a large number of subscriptions favors LATE aggregation (see Section 6.2.1). ADAPTIVE sends a similar number of messages as LATE in that situation.

For the Traffic dataset (see Fig. 6(d)), EARLY aggregation performs better than LATE. However, increasing the number of subscriptions reduces the performance advantage of EARLY. For example, with 45 subscriptions, LATE sends 1.8 times more messages while at 990 subscriptions, it sends only 1.2 times more messages. The graph for ADAPTIVE exhibits a stairstep shape which is due to occasions where brokers wrongly decide to forward. This error occurs when a large number of publications matches a small number of subscriptions. Indeed, our heuristic assumes that a large number of publications is likely to match a large number of subscriptions and therefore overestimates the number of messages sent when aggregating. Consequently, certain brokers decide to forward messages instead, which causes the ADAPTIVE solution to send more messages, as observed at 360 subscriptions. When increasing the number of subscriptions to 540, the number of messages remains constant for ADAPTIVE while it increases for EARLY and LATE. The additional subscriptions are matched by some of the publications received, which reduces the skewness observed for 360 subscriptions. Therefore, the estimation of the adaptation engine is more accurate, and its error diminishes. The same scenario is repeated for 630 subscriptions.

Percentage of aggregate subscriptions: Figure 6(e) compares the performance of different aggregation schemes when varying the proportion between aggregate and non-aggregate subscriptions in the workload. For this experiment, we consider 135 subscriptions and vary the percentage of aggregate subscriptions from 0% to 100%. When the proportion of aggregate subscriptions increases, LATE (shown in red with each data-point being represented by a circle) reduces the number of messages proportionally. With a mix of regular and aggregate subscriptions, ADAPTIVE achieves the most reduction by leveraging the overlap between subscriptions. If there are only aggregate subscriptions, the ADAPTIVE solution performs similarly to EARLY. In this scenario, we can also observe that LATE aggregation is able to reduce the number of messages slightly. This is accounted for by the small gain obtained by aggregating publications at subscriber edge brokers before delivering the results to the client.

7.2.2 Processing load

Publication rate: Fig. 7(a) shows the processing load when varying the publication rate using the Stock dataset. We observe that the cost for LATE is proportional to the number of messages exchanged for aggregation. The overall processing load can be split into two groups: predicate matching and aggregation processing (see Figs. 7(b) and 7(c), respectively). The former is directly proportional to the overall number of publications exchanged, while the latter depends only on the number of publications matching aggregate subscriptions.

For predicate matching (see Fig. 7(b)), we observe similar costs for all three approaches. EARLY performs better than LATE at high publication rates, since it generates less messages for aggregation (see Fig. 7(a)). The cost of predicate matching for ADAPTIVE is between those of EARLY and LATE, which follows the trend set by the number of messages exchanged.

As shown in Fig. 7(c)), the aggregation loads of EARLY and LATE follow the same pattern as their number of messages exchanged for aggregation (see Fig. 6(a)). The computation overhead for ADAPTIVE includes an additional cost due to its adaptation engine which causes the computation cost to be higher relative to those of other solutions at low publication rates. However, for higher publication rates, the predicate matching cost becomes prominent for LATE.

For the Traffic dataset (see Fig. 7(d)), we identify two cases. First, when the publication rate is lower than 135 pubs/s, LATE performs better than EARLY. In this case, the ADAPTIVE solution simulates the behavior of the LATE solution, hence, their similar costs. Second, for higher publication rates, ADAPTIVE adopts the EARLY approach, but is less efficient due to the additional overhead of the adaptation engine and the cost of sending additional messages. LATE sends a significantly larger amount of publications compared to the two others and, hence, has the highest computation overhead.

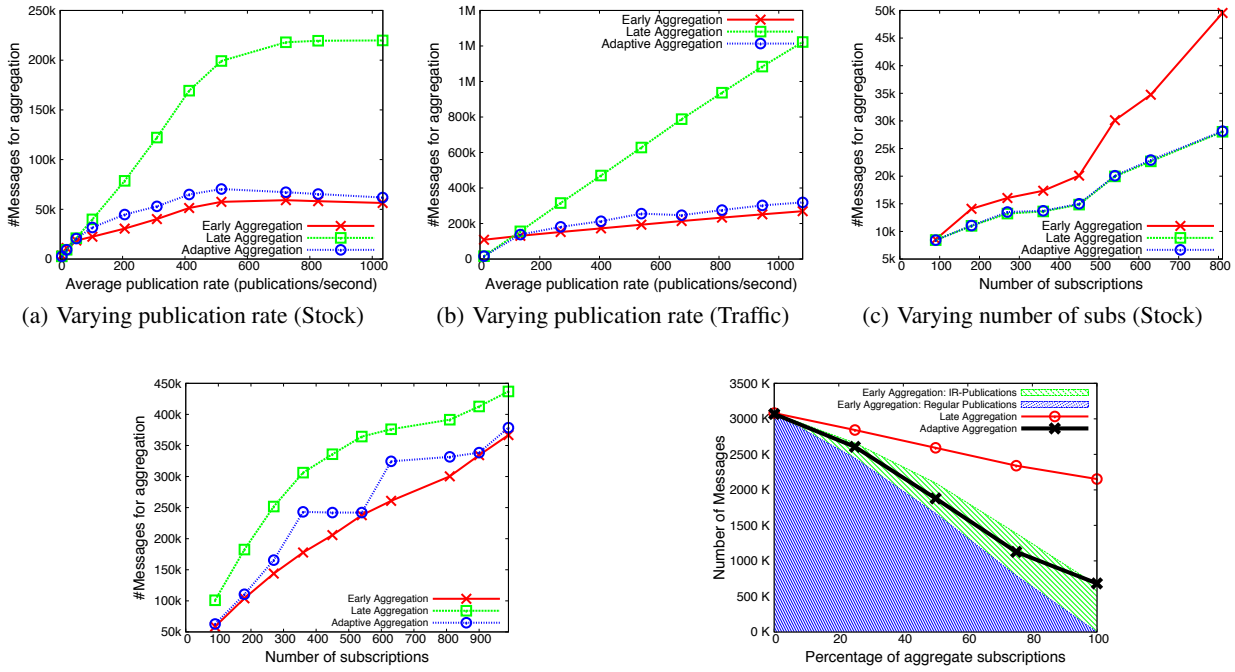
Number of subscriptions: While varying the number of subscriptions for the Stock dataset (see Fig. 7(e)), the computation cost of LATE and EARLY varies according to the number of messages exchanged.

Notably, the adaptation cost per publication for the ADAPTIVE, which is proportional to the number of messages, increases with the number of subscriptions, which causes ADAPTIVE to draw the highest computation cost. For the Traffic dataset where the number of matching publications is higher than for Stock, EARLY generates the least overhead (see Fig. 7(f)). The computation cost follows the number of messages generated by these approaches (see Fig. 6(d)), which asserts that it is proportional to the number of messages. As ADAPTIVE sends fewer messages than LATE, its computation overhead lies between LATE and EARLY.

7.2.3 End-to-end delay

Our solution for distributed aggregation in pub/sub buffers publications and IR messages. These delays are important as it impacts further messages reduction as explained in Section Sect. 5.2.2. We measure the total delay accumulated end-to-end for aggregate notifications while varying the number of subscriptions in Fig. 8(a).

A general observation is that the delay increases with the number of subscriptions. This behavior is due to the increase in processing load. Since LATE aggregation does not perform RPP at intermediate brokers, its delay is significantly lower than other strategies. For EARLY, publications are aggregated at the publisher edge broker, which forces the RPP process to be applied on each downstream broker, maximizing the delay. ADAPTIVE is mainly aggregating at the publisher edge broker for this workload configuration and thus performs similarly to EARLY. However, for more than 500



(a) Varying publication rate (Stock) (b) Varying publication rate (Traffic) (c) Varying number of subs (Stock)

(d) Varying number of subs (Traffic) (e) Percentage of aggregate subs (Traffic)

Figure 6: Comparative communication cost of different schemes

subscriptions, ADAPTIVE has a slight advantage due to some edge brokers which decide not to aggregate.

7.3 Sensitivity analysis

We now present a sensitivity analysis for collection delay and sampling period for the ADAPTIVE solution.

Collection delay: Fig. 8(c) shows the impact of the collection delay parameter on the number of messages and on the end-to-end notification delay. As expected, the notification delay varies linearly with the collection delay. We also observe that the collection delay effectively reduces the number of messages generated by ADAPTIVE. For example, increasing this delay from 1 ms to 20 ms reduces the amount of messages sent by more than 3%. We note that a collection delay greater than 20 ms does not further reduce the number of messages.

Sampling period: The sampling period used in the adaptation engine 6.2.1 of a broker to decide whether it should perform the aggregation or not. If this sampling period is too short, variations in the workload may cause a large number of transitions leading to the sending of additional messages. In contrast, if the sampling period is too long, it reduces the ability of the broker to adapt quickly and limits its effectiveness dealing with high rates of change.

Fig. 8(b) shows the impact of the sampling period length over the number of messages for ADAPTIVE. This graph can be divided into two regions. First, for a sampling period lower than 5s, the number of messages decreases linearly with the sampling period. Afterwards, the number of messages remains constant. We can explain this behavior through the pattern of the number of transitions which exhibits variations similar to that of the number of messages. We observe that, when the sampling period is below 5s, brokers frequently switch between aggregate and forward mode (up to 7838 times). Brokers therefore exchange more messages when the sampling period is small due to the transition overhead.

7.4 Summary

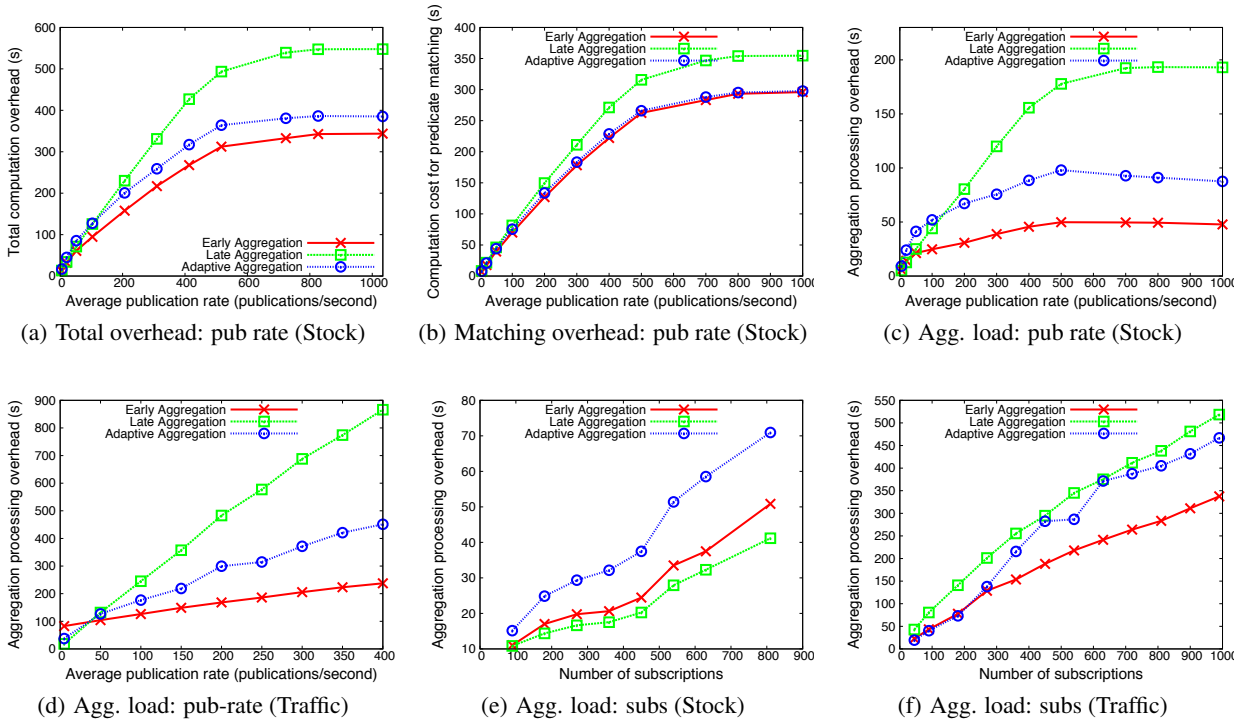
The relative performance of the two baselines, i.e., EARLY and LATE aggregation, is highly dependent on the workload used, more specifically, the publication rate and the number of subscriptions. We observe that our ADAPTIVE aggregation effectively switches between EARLY and LATE aggregation according to these factors. However, due to its predictive nature, ADAPTIVE sends slightly more messages than the technique which it adapts to at any time.

The cost of the adaptation engine is prominent at low publication rates which makes the ADAPTIVE processing cost the highest as observed in the Stock dataset. However, the gain in message reduction balances this cost at higher publication rates, and despite having additional computation overhead for the heuristic, ADAPTIVE aggregation then has a processing load between that of LATE and EARLY aggregation.

We have also shown the importance of the sampling period which can lead up to 55% more messages, if not properly set. The collection delay, which trades notification delay for message reduction, has shown sizable impact as it can provide a 3% reduction in the number of messages in exchange for a notification delay which is doubled.

8. LIMITATIONS OF ADAPTIVE AGGREGATION

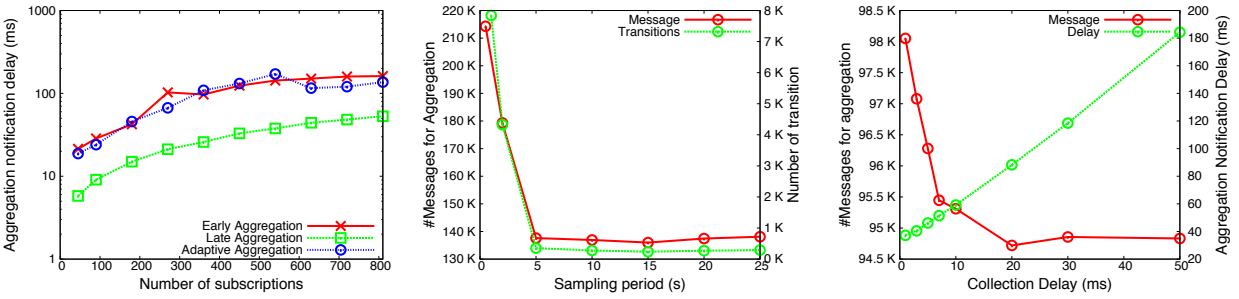
Although the experiments presented in Sect. 7 show that our adaptive aggregation technique performs generally better than the LATE and EARLY aggregation strategies, there are a number of corner cases where adaptation will generate an excessive overhead. First of all, the ADAPTIVE solution employs a predictive strategy: given an incoming publication, the broker decides whether the publication should be aggregated or not by considering a history of publications received so far. As for any predictive strategy, it can be rendered ineffective by an adversary that constantly reverses the



(a) Total overhead: pub rate (Stock) (b) Matching overhead: pub rate (Stock) (c) Agg. load: pub rate (Stock)

(d) Agg. load: pub-rate (Traffic) (e) Agg. load: subs (Stock) (f) Agg. load: subs (Traffic)

Figure 7: Comparison: Processing load



(a) End-to-end delay (Traffic) (b) Sampling period (Traffic) (c) Collection delay (Traffic)

Figure 8: End-to-end delay comparison and sensitivity analysis for ADAPTIVE

trends. Moreover, the decision is based on a specific heuristic for estimating the amount of traffic that the broker will produce in the future. A more precise heuristic may yield further traffic reduction.

In the proposed ADAPTIVE solution, the broker makes a decision for an incoming publication without considering the tradeoffs on a per-subscription basis. This works well when similar amounts of aggregation traffic are produced for different subscriptions. When this assumption is not valid, we need to consider adaptation at the granularity of individual subscriptions. Furthermore, each broker makes a decision independently. A coordinated strategy across neighboring brokers may be more effective.

9. CONCLUSIONS

Modern use cases for pub/sub require aggregation capabilities that are not natively supported by traditional pub/sub solutions. Although aggregation can be provided through the end-to-end principle, late processing at the subscriber edge is not practical for data intensive applications where disseminating the entire event stream through the pub/sub system is prohibitively expensive. To alleviate this problem, we propose a distributed approach to pub/sub aggregation which allows brokers to process and disseminate partial aggregation results. We introduce mechanisms integrated with the pub/sub system that provide aggregation of publications within the broker overlay and reduce the amount of in-network message traffic. We consider an early aggregation approach that places aggregators at the publisher edges. This approach significantly reduces the traffic for workloads where the publication rate is higher than the aggregation notification rate. However, we demonstrate that this approach is not always optimal either and propose an adaptive rate-based solution, which outperforms both baselines on datasets extracted from two real applications.

10. REFERENCES

[1] S. F. Abelsen, H. Gjermundrd, D. E. Bakken, and C. H. Hauser. Adaptive data stream mechanism for control and monitoring applications. In *Proc. of ADAPTIVE*, pages 86–91, 2009.

[2] B. Arai, G. Das, D. Gunopulos, and V. Kalogeraki. Efficient approximate query processing in peer-to-peer networks. *IEEE Trans. on Knowl. and Data Eng.*, 19(7):919–933, 2007.

[3] A. Arasu, M. Cherniack, E. Galvez, D. Maier, A. S. Maskey, E. Ryvkina, M. Stonebraker, and R. Tibbetts. Linear road: a stream data management benchmark. In *Proc. of VLDB*, pages 480–491, 2004.

[4] R. Baldoni, L. Querzoni, S. Tarkoma, and A. Virgillito. Distributed event routing in publish/subscribe systems. *Chapter 10 in the book MiNEMA*, pages 219–244, 2009.

[5] S. Biswas, M. Taghizadeh, and F. Dion. Vehicle-to-vehicle wireless communication protocols for enhancing highway traffic safety. *IEEE comm. mag.*, 44(1):74–82, 2006.

[6] L. Brenna, J. Gehrke, M. Hong, and D. Johansen. Distributed event stream processing with non-deterministic finite automata. In *Proc. of DEBS*, pages 1–12, 2009.

[7] A. Carzaniga, D. S. Rosenblum, and A. L. Wolf. Design and evaluation of a wide-area event notification service. *ACM Tran. on Computer Systems*, 19(3):332–383, 2001.

[8] B. Chandramouli and J. Yang. End-to-end support for joins in large-scale publish/subscribe systems. *VLDB Endowment*, 1(1):434–450, 2008.

[9] J. Chen, L. Ramaswamy, and D. Lowenthal. Towards efficient event aggregation in a decentralized publish-subscribe system. In *Proc. of DEBS*, pages 1–11, 2009.

[10] A. Demers, J. Gehrke, M. Hong, M. Riedewald, and W. White. Towards expressive publish/subscribe systems. In *Proc. of EDBT*, pages 627–644, 2006.

[11] T. Fawcett and F. Provost. Activity monitoring: noticing interesting changes in behavior. In *Proc. of SIGKDD*, pages 53–62, 1999.

[12] E. Fidler, H.-A. Jacobsen, G. Li, and S. Mankovski. The padres distributed publish/subscribe system. In *Proc. of ICFI*, pages 12–30, 2005.

[13] S. Frischbier, A. Margara, T. Freudenreich, P. Eugster, D. Eyers, and P. Pietzuch. ASIA: application-specific integrated aggregation for publish/subscribe middleware. In *Proc. of Middleware (Poster Paper)*, pages 1–2, 2012.

[14] S. Frischbier, A. Margara, T. Freudenreich, P. Eugster, D. Eyers, and P. Pietzuch. Aggregation for implicit invocations. In *Proc. of AOSD*, pages 109–120, 2013.

[15] L. Golab, K. G. Bijay, and M. T. Özsu. Multi-query optimization of sliding window aggregates by schedule synchronization. In *Proc. of CIKM*, pages 844–845, 2006.

[16] IBM Corp. An architectural blueprint for autonomic computing. IBM White Paper, 2004.

[17] N. Jain, D. Kit, P. Mahajan, P. Yalagandula, M. Dahlin, and Y. Zhang. STAR: self-tuning aggregation for scalable monitoring. In *Proc. of VLDB*, pages 962–973, 2007.

[18] K. R. Jayaram, C. Jayalath, and P. Eugster. Parametric subscriptions for content-based publish/subscribe networks. In *Proc. of Middleware*, pages 128–147, 2010.

[19] M. Jelasity, A. Montresor, and O. Babaoglu. Gossip-based aggregation in large dynamic networks. *ACM Trans. Comput. Syst.*, 23(3):219–252, 2005.

[20] Z. Jerzak and C. Fetzer. Handling overload in publish/subscribe systems. In *Proc. of ICDCSW*, pages 32–37, 2006.

[21] P. Jesus, C. Baquero, and P. S. Almeida. A survey of distributed data aggregation algorithms. Technical report, University of Minho, 2011.

[22] R. S. Kazemzadeh and H.-A. Jacobsen. Opportunistic multipath forwarding in content-based publish/subscribe overlays. In *ACM Middleware*, pages 249–270, 2012.

[23] I. Koenig. Event processing as a core capability of your content distribution fabric. In *Gartner Event Processing Summit*, 2007.

[24] A. Koulakezian and A. Leon-Garcia. CVI: Connected vehicle infrastructure for ITS. In *Proc. of PIMRC*, pages 750 –755, 2011.

[25] S. Krishnamurthy, C. Wu, and M. Franklin. On-the-fly sharing for streamed aggregation. In *Proc. of SIGMOD*, pages 623–634, 2006.

[26] G. Li, V. Muthusamy, and H.-A. Jacobsen. A distributed service-oriented architecture for business process execution. *ACM Trans. Web*, 4(1):2:1–2:33, Jan. 2010.

[27] V. Muthusamy, H.-A. Jacobsen, T. Chau, A. Chan, and P. Coulthard. SLA-driven business process management in SOA. In *CASCON*, pages 86–100, 2009.

[28] T. Repantis and V. Kalogeraki. Hot-spot prediction and alleviation in distributed stream processing applications. In *Proc. of DSN*, pages 346–355, 2008.

[29] I. Rose, R. Murty, P. Pietzuch, J. Ledlie, M. Roussopoulos, and M. Welsh. Cobra: content-based filtering and aggregation of blogs and RSS feeds. In *Proc. of NSDI*, 2007.

[30] A. Schröter, D. Graff, G. Mühl, J. Richling, and H. Parzyjegla. Self-optimizing hybrid routing in publish/subscribe systems. In *Proc. of DSOM*, pages 111–122, 2009.

[31] V. Setty, G. Kreitz, R. Vitenberg, M. van Steen, G. Urdaneta, and S. Gimåker. The hidden pub/sub of Spotify: (industry article). In *Proc. of DEBS*, pages 231–240, 2013.

[32] J. Sventek and A. Koliousis. Unification of publish/subscribe systems and stream databases: the impact on complex event processing. In *Proc. of Middleware*, pages 292–311, 2012.

[33] Y. Tock, N. Naaman, A. Harpaz, and G. Gershinsky. Hierarchical clustering of message flows in a multicast data dissemination system. In *Proc. of PDCS*, pages 320–326, 2005.

[34] R. Van Renesse, K. P. Birman, and W. Vogels. Astrolabe: A robust and scalable technology for distributed system monitoring, management, and data mining. *ACM Trans. Comput. Syst.*, 21(2):164–206, 2003.

[35] R. van Renesse and A. Bozdog. Willow: DHT, aggregation, and publish/subscribe in one protocol. In *Proc. of IPTPS*, pages 173–183, 2004.

[36] M. Wood and K. Marzullo. The design and implementation of Meta. In *Reliable Distributed Computing with the Isis Toolkit*, pages 309–327, 1994.

[37] S. Wu, B. C. Ooi, and K.-L. Tan. Continuous sampling for online aggregation over multiple queries. In *Proc. of SIGMOD*, pages 651–662, 2010.

[38] P. Yalagandula and M. Dahlin. A scalable distributed information management system. *SIGCOMM Comput. Commun. Rev.*, 34(4):379–390, 2004.

[39] P. Yalagandula and M. Dahlin. Shruti: A Self-Tuning Hierarchical Aggregation System. In *Proc. of SASO*, pages 141–150, 2007.