Minimizing the Communication Cost of Aggregation in Publish/Subscribe Systems

Navneet Kumar Pandey*, Kaiwen Zhang†, Stéphane Weiss*, Hans-Arno Jacobsen†, Roman Vitenberg*
*Department of Informatics, University of Oslo, Norway
†Department of Computer Science, University of Toronto, Canada
*{navneet,stephawe,roman}@ifi.uio.no,†{kzhang,arno}@msrg.utoronto.ca

***Abstract*—Modern applications for distributed publish/subscribe systems often require stream aggregation capabilities along with rich data filtering. When compared to other distributed systems, aggregation in pub/sub differentiates itself as a complex problem which involves dynamic dissemination paths that are difficult to predict and optimize for a priori, temporal fluctuations in publication rates, and the mixed presence of aggregated and non-aggregated workloads. In this paper, we propose a formalization for the problem of minimizing communication traffic in the context of aggregation in pub/sub. We present a solution to this minimization problem by using a reduction to the well-known problem of minimum vertex cover in a bipartite graph. This solution is optimal under the strong assumption of complete knowledge of future publications. We call the resulting algorithm "Aggregation Decision, Optimal with Complete Knowledge" (*ADOCK*). We also show that under a dynamic setting without full knowledge, *ADOCK* can still be applied to produce a low, yet not necessarily optimal, communication cost. We also devise a computationally cheaper dynamic approach called "Aggregation Decision with Weighted Publication" (*WAD*). We compare our solutions experimentally using two real datasets and explore the trade-offs with respect to communication and computation costs.**

I. Introduction

Aggregation is known to bring benefits to many distributed systems and applications [1], [2], [3], [4]. One of the main reasons behind its widespread use resides in its ability to extract intelligible information out of large sets of data. This paper discusses the challenges posed by introducing aggregation to publish/subscribe, a paradigm which has been widely applied in a variety of areas ranging from business process execution [5], [6], stock-market monitoring [7] to social interactions [8].

Traditionally, pub/sub has focused on high throughput and scalability rather than on extended features such as aggregation. Nonetheless, aggregation in pub/sub promises to provide support for new application areas and to deliver improved performance for existing big data applications [9], [10], [11], [12], [13].

We illustrate the need for aggregation in pub/sub through two application scenarios. We first consider an intelligent transport system (ITS) [14], designed within the framework of the Connected Vehicle and Smart Transportation (CVST) project [15]. In such a system, road sensor data as well as crowdsourced data from mobile users are transmitted to ITS server farms to coordinate traffic flow and to invoke the appropriate agents on demand (e.g., police, first responders, radio networks.) Supporting this application can only be done through the use of sophisticated filtering capabilities to allow each data sink to express its subscription interests vis-à-vis streams of publication events. Furthermore, agents may be mobile and versatile, thus, they are in need of a loosely coupled and dynamic communication paradigm. Put together, these requirements indicate that content-based pub/sub represents a promising solution for our scenario. Additionally, ITS requires real-time monitoring on aggregate queries over time-based windows. For instance, the average car speed on a road segment can be obtained by aggregating the speed of individual cars at the location [1]. Another example is measuring the amount of traffic during rush hours, which can be computed by aggregating the number of individual cars during that period of time.

As a second illustrating scenario, we consider a stock market application, where stock values must be disseminated to traders located in different agencies. Since the traders' interests are changing over time, content-based pub/sub represents a viable communication substrate for enabling such data dissemination. In addition to individual stock value updates, aggregated values such as stock market indices or stock indicators are commonly requested. For example, the MACD[1] indicator can be obtained by computing the exponential moving average of stock values.

These scenarios illustrate the need for a scalable content-based pub/sub system which can support aggregate subscriptions. A more comprehensive discussion on the need for aggregation in various pub/sub applications is presented in our former paper [16]. However, the focus of the current paper is different; see Section II for a detailed comparison.

Traditionally, scalability is achieved by reducing the communication cost of the pub/sub infrastructure. One popular technique to reduce the number of messages exchanged is to build publication dissemination trees over an overlay of brokers [17], [18]. Brokers are lightweight processes which are in charge of handling and matching pub/sub messages in order to route publications towards interested subscribers. This overlay approach proves to be beneficial for aggregation since brokers constitute ideal candidates for the deployment of aggregation processing [16]. One important observation is that aggregating publications at the correct broker reduces the amount of in-network traffic by balancing the communication cost of disseminating raw publications with the cost of disseminating aggregation results. Minimizing the number of messages required to satisfy all the aggregation queries can be

[1]Moving average convergence/divergence

formalized as a decision problem for each broker to determine which publications need to be forwarded and which ones need to be aggregated. This paper is a study of this theoretical problem, its ramifications for proposed practical solutions, and the evaluation of said solutions.

The contributions of this paper are the following:

1) We introduce and formalize the Minimum-Notifications-for-Aggregation (MNA) problem in content-based pub/sub, a decision problem which determines for each broker which publications to forward, and which ones to aggregate.

2) We show that MNA is solvable in polynomial time by reducing it to Minimum-Vertex-Cover over bipartite graphs.

3) We present the *Aggregation Decision, Optimal with Complete Knowledge* (*ADOCK*) solution based on the reduction. It is optimal under the assumption that the broker knows all publications and subscriptions in a given aggregation window.

4) We describe the *Weighted Aggregation Decision* (*WAD*) algorithm, a heuristic which is computationally less expensive than ADOCK.

5) We evaluate the two solutions against a known baseline and explore the trade-off between communication and computation costs. The evaluation is performed using two real datasets extracted from our presented application scenarios (traffic monitoring and stock market).

II. Related Work

Distributed data aggregation is a well explored area of research where diverse techniques have been proposed and developed [3], [4], [19], [20], [12], [21]. We can classify these techniques into two categories: communication-oriented and computation-oriented.

Communication-oriented work: Communication optimization, the primary focus of our research, has been considered for decentralized data aggregation with the goal of determining system-wide properties [4]. However, the consideration was limited to the same level of filtering expressiveness as topic-based pub/sub. Thus, extending it for rich content-based dissemination leads to scalability issues due to the exponential explosion of the number of delivery paths as mentioned in [22], [23]. Meta [24], Shruti [20], and gridStat [2], which all provide efficient routing using a control layer. The use of such a control layer is incompatible with the loosely coupled nature of pub/sub systems as it requires global knowledge of the overlay. In addition, these solutions do not take into account the existence of non-aggregate subscriptions. Techniques from distributed stream processing systems (DSPS), such as efficient notification scheduling [25], which optimizes message traffic by batching, are orthogonal to and compatible with our approach.

Our previous paper [16] introduces aggregation techniques for pub/sub systems and provides a preliminary adaptive solution. The proposed solution, called per-Broker Adaptive aggregation Technique (BAT), allows brokers to adaptively switch between two modes: forwarding all publications or aggregating all publications. Switching is based on a simple rate-based heuristic. In contrast, the approach introduced in this paper captures and formalizes the concept of an optimal solution. We show how the problem can be solved in an optimal manner under the assumption of knowing future publications. Our techniques allow us to derive two practical heuristics that do not require this assumption. These heuristics outperform BAT, as we show in Section VIII. The granularity of the decision problem and the adaptive scheme is also refined in this paper.

Computation-oriented work: DSPSs and other distributed systems that focus on optimizing computations for aggregation face a number of challenges common to distributed content-based pub/sub. These systems also aggregate data from distributed publication sources and serve multiple queries [26]. However, in contrast to aggregation in pub/sub, the publication sources are a priori known before making query plans and operator placements [20]. Such static query plans require a global view of the overlay topology, and hence can not be directly applicable to pub/sub wherein the location of sources and sinks is dynamic and each broker's knowledge of the topology is restricted to its neighborhood. Furthermore, 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 planning optimizations ineffective. On the other hand, some of the optimization techniques such as multi-query optimization [27], [28] can be applied in pub/sub. Cayuga [9], [29] and the approach in [10] provide distributed aggregation and focus on efficient and scalable query processing for complex event patterns. In contrast, our primary focus is on optimizing distributed data flows. Systems which aggregate metadata [30], [31] are orthogonal to our goal of aggregating content.

More closely related to our work is [32], which also provides support for aggregation in pub/sub. Here, the task of aggregation is allocated per topic to the broker located at the center of the routing tree. This paper shares similarities with our approach in that aggregation is performed at brokers in an overlay defined by pub/sub routing. The key differences are that our work allows the aggregation task to be itself distributed to multiple brokers instead of being assigned to a central broker. Furthermore, we provide a formal treatment of the problem which demonstrates an optimal distribution of the aggregation task amongst brokers.

III. Background

In this section, we describe the relevant terminology for aggregation in pub/sub systems.

Publish/Subscribe model: We employ an overlay broker network where each publisher and subscriber is connected to a broker. Brokers forward publications through the overlay to be delivered to the appropriate subscribers. Our work is based on the content-based matching model with advertisement-based forwarding. Publications are forwarded based on their content following routing paths which are initialized by advertisements flooded through the network. Subscriptions are matched against advertisements to construct delivery paths from the publishers to the subscribers. The delivery paths form a *subscription delivery tree* (SDT) for each subscriber. The SDT is rooted at the subscriber node and contains all

the brokers necessary to reach publishers, which form the leaves of the tree. Publications are then matched against the subscriptions and forwarded down to the appropriate next hops according to the SDTs of matching subscriptions. We also consider only point-to-point communication between pairs of brokers. Both advertisement and subscription covering are not considered. Relaxation of the model to allow for cyclic routing, multicasting, publication batching, and their implications on the complexity of the optimal solution are out of the scope of this paper.

Aggregation overview: A comprehensive description of relevant aggregation terminology can be found in [4]. An aggregation function takes a multiset of elements and produces a result. Example aggregation functions include *average*, *minimum*, *sum*, and *top-k*. An aggregation function is considered *decomposable* if the same result can be obtained by aggregating the partial results obtained by aggregating subsets of the original input. Essentially, decomposability allows the aggregation computation to be distributed by partitioning the input set.

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 in order to avoid counting duplicates.

Given a stream of values, aggregation functions are computed repeatedly over a sequence of windows, each with a start point and duration. These window properties can be defined using time (*time-based* windows) or number of elements (*count-based* windows). 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 are overlapping (*sliding*). If $\delta = \omega$, the window is called *tumbling*, otherwise for ($\delta > \omega$), it is called *sampling*.

IV. Publish/Subscribe aggregation

Our design choice is to enhance publish/subscribe by supporting subscriptions with aggregation semantics. This is more communication- and computation-efficient than running two separate infrastructures for pub/sub and aggregation. This is especially important because pub/sub systems are communication-intense and characterized by a large number of publications.

To this end, we introduce aggregation subscriptions, which are an extension of regular subscriptions: They include values for δ and ω in addition to conventional conditional predicates. Note that this design choice results in a system where aggregation and regular subscriptions co-exist so that the system needs to support a dynamic combination of both.

In order to support aggregation, the pub/sub system must repeatedly compute results for windows of matching publications for each aggregation subscription. These results are then delivered to the corresponding subscribers as soon as they are available. Note that there are no guarantees enforced on the delivery policy for those results (e.g., ordering); such a QoS policy is orthogonal to the focus of our paper.

In order to maximize communication efficiency, we want to reuse the dissemination flow and mechanisms already present in pub/sub to the extent possible. Publications matching aggregation or regular subscriptions are propagated along SDTs using the same match-and-forward algorithms. When a publication matches both an aggregation and a regular subscription, we can forward it only once to conserve bandwidth.

The main difference and new challenge compared to existing solutions for aggregation, however, is distributing the computation of aggregation across multiple brokers in the overlay. When a broker in a SDT receives publications p_1 and p_2 related to the same window of a decomposable aggregation subscription, it might be able to aggregate p_1 and p_2 and route the partial aggregate result further along the SDT instead of forwarding p_1 and p_2 separately. This reduces the publication load (both forwarding and matching) on subsequent brokers along the SDT paths. It also produces a load of partial results, yet not necessarily in the same proportion. For instance, if five publications can be aggregated in one single result, the publication load decreases by five messages while the result load increases only by one. In other words, the cumulative load of publications and results is not constant such that adjusting the placement of aggregation-related tasks can minimize it. The trade-off is non-trivial, however, and can be affected by the rate of matching publications, the proportion between aggregation and regular subscriptions, overlap between successive time windows for the same aggregation subscription, broker topology, placement of publishers and subscribers, etc.

The challenge is significant because (a) SDTs are determined by the pub/sub system and, thus, are outside of the control of the aggregation scheme, (b) SDTs and several other factors affecting the trade-off are formed and changed dynamically during the execution, and (c) there is no global knowledge about these factors in a loosely-coupled pub/sub system: Each broker can monitor its local situation and requires a distributed protocol in order to learn the situation at its neighboring brokers in the SDT. This challenge distinguishes our work vis-à-vis other aggregation systems.

We now present two naive solutions, which represent the two extremes with respect to distributing the computation of aggregation across the brokers. The first solution, called *subscriber-edge aggregation*, assigns the tasks of computing and disseminating results to brokers directly connected to subscribers (called *subscriber edge brokers*). The rest of the brokers in the overlay are simply in charge of delivering publications all the way to those subscriber edge brokers. Another solution, called *publisher-edge aggregation*, allocates the task of computing results to the brokers directly connected to publishers. Those results are then disseminated by the intermediary brokers in the SDTs to the subscriber edge brokers, who then aggregate the results together before sending the compiled data to the subscribers. Other solutions fall between these two extremes and allocate aggregation computation to various intermediate brokers located within the SDTs of aggregation subscriptions.

Our goal in this paper is to find a solution for aggregation in pub/sub that minimizes the total load of all brokers. Minimizing the communication and computation loads are different yet related optimization objectives. We consider both in our experiments in Section VIII-C and show that minimizing

communication cost typically (but not always) leads to reduced computation cost because a smaller number of messages need to be matched. Our theoretical treatment of the problem focuses on optimizing the communication.

Aggregation as a local decision problem: We cast our performance objective as a local decision problem run at each broker in the SDTs for aggregation subscriptions. Each broker must individually decide when to aggregate and disseminate results and when to forward publications based on its limited knowledge of the entire system. Although coordination might produce a globally optimal solution, such a coordination is impossible in loosely-coupled pub/sub systems. Thus, every broker monitors its own local performance parameters, such as the rate of matching publications for aggregation and regular subscriptions and the notification frequency, which can be derived from the normal routing operations of the pub/sub system. We show how these parameters are sufficient to make informed decisions.

In absence of coordination and other control messages, the total communication cost depends on messages with publications and partial aggregation results. The bandwidth is spent on forwarding the content, along with pub/sub metadata and headers of different protocols at various levels. Because publications and even aggregate results typically have fairly small size, the space in the messages is mostly occupied by metadata and headers. Therefore, the consumed bandwidth is roughly proportional to the number of messages, which we use as the specific optimization objective.

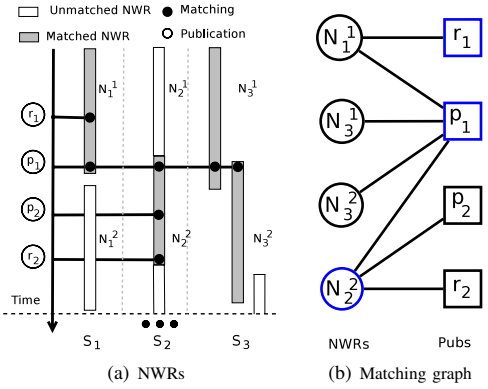
Even though the decision problem is local, its granularity is still non-trivial. Consider that the same broker b' can be a parent of broker b in multiple SDTs T_1 and T_2, and the same publication p may match the subscriptions of both T_1 and T_2. If b has already decided to forward p to b' for T_1, there is no need to take a separate decision for T_2. On the other hand, if b has decided to aggregate for T_1 but the aggregate operator for T_2 is different, p may need to be forwarded or aggregated anew. At the same time, the decisions for SDTs where the parent of b is a different neighbor broker are always separate because SDTs are acyclic and because we assume unicast communication between neighbor brokers, as commonly accepted in pub/sub. Therefore, we consider the local decision on a per-neighbor basis for each broker, that is for all SDTs that have the same neighbor as the SDT parent.

V. Broker Decision Process

In this section, we present the aggregation decision process at the broker level. We first define the concept of Notification Window Range (NWR) before moving on to the flow of aggregation processing.

A. Notification Window Range (NWR)

To manage notifications for each time window of a subscription, we employ the concept of Notification Window Range (NWR), which is a triplet of ⟨*subscription*, *start-time*, and *duration*⟩. A publication p is said to match one NWR $\langle sub, time, duration\rangle$ if and only if p matches the subscription sub, and has been published between *time* and *time* + *duration*. After *time*+ *duration*, this NWR is considered "expired" and becomes ready for processing. The following example illustrates the relationship between aggregate subscriptions and NWRs.



(a) NWRs (b) Matching graph

Fig. 1: Diagrams for Example 1

Example 1. We consider an example from the traffic use case in which sensors are distributed over a city. These sensors detect the presence of a car and estimate its speed. Each car detection generates a publication which contains various measurements including the estimated speed. In this example, we consider three aggregate subscriptions (s_1, s_2, s_3) defined as follows:

- s_1 aims at counting daily the number of cars passing a crossing during a given period of two hours. It uses a *sampling* window with δ =*24 hours* while ω =*2 hours*, aggregating the number of publications received. Assuming that s_1 is issued at t_1, we consider the following NWRs: $N_1^1 = \langle s_1, t_1,$ 2h$\rangle, N_1^2 = \langle s_1, t_1$+24h,2h$\rangle, N_1^3 = \langle s_1, t_1$+48h,2h$\rangle$, etc.
- s_2 needs to estimate the average car speed on a particular road segment for each hour. It uses a tumbling window with $\delta = \omega$ =*1 hour* and generates $N_2^1 = \langle s_2, t_2,$ 1h$\rangle, N_2^2 = \langle s_2, t_2$+1h, 1h$\rangle$, etc.
- Finally, the moving average of the number of cars passing a traffic light is used for statistical analysis. This information can be computed using a sliding window subscription s_3 with the operator *average* applied to the number of publications received. The parameters of s_3 in this example are ω =*2 hours* and $\delta = 0.5 * \omega$ = *1 hour* and the corresponding NRWs are $N_3^1 = \langle s_3, t_3,$ 2h$\rangle, N_3^2 = \langle s_3, t_3$+1h, 2h$\rangle, N_3^3 = \langle s_3, t_3$+2h, 2h$\rangle$, etc.

These subscriptions, which are expressing sliding, tumbling, and sampling window semantics respectively, are shown in Figure 1(a). The first publication (labelled p_1) matches all subscriptions. Since s_3 uses a sliding window, p_1 matches two NWRs (N_3^1 and N_3^2), as opposed to one NWR each for subscriptions s_2 (N_2^2) and s_1 (N_1^1). The second publication p_2 matches only s_2 and contributes to a single NWR N_2^2.

B. Processing flow

Figure 2 illustrates the aggregation flow at a single broker. When a publication is received, the broker performs its regular matching process and forwards it downstream to the next

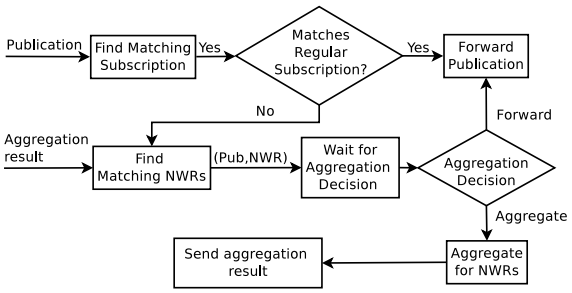


Fig. 2: Aggregation flowchart at a single broker

hop of any matching non-aggregate (regular) subscription. If the publication does not match any regular subscriptions, the broker processes aggregate subscriptions to find matching NWRs. Publications are stored at the broker until all associated NWRs expire.

At the expiration time of an NWR, the broker runs an aggregation decision algorithm to decide whether or not to aggregate. If the decision is to aggregate, publications attached to the NWR are processed to obtain an aggregation result which is sent down the link to the next hop leading to the target subscription. This result, when received at a broker downstream, is processed to find associated NWRs and is thus taken into account in the decision algorithm of those NWRs when processed further along the path.

VI. Problem formulation

For aggregation in distributed pub/sub systems, the problem of finding the minimum number of notifications required for aggregation can be formalized as a decision problem which determines which publications need to be forwarded and which ones need to be aggregated by each broker.

A. Minimum-Notifications-for-Aggregation

The objective of the aggregation decision algorithm is to minimize the number of notified data items, which consist of publications and aggregation results, sent by a broker to one of its neighbor. We formalize this optimization problem, which we call *Minimum-Notification-for-Aggregation* (MNA).

Let I be a set of data items received by a broker and S_b the set of subscriptions sent to the broker by a neighboring broker b. For the purpose of this section, we treat regular subscriptions in S_b as a special case of aggregation. We can convert those non-aggregate subscriptions to the NWR model by creating one NWR for each matching publication. Notifying an NWR then involves sending its content, which is a single publication. Elements in I can be either raw publications originating from publishers or aggregation results processed by upstream brokers. A subscription s is treated as a set of NWRs extracted from its window semantics: $s = \{N_1, N_2, \ldots, N_n\}$, as described in Section V-A. The content of an NWR N is a subset of data elements from I denoted as I_N, which matches the NWR N. We define $f(I_N)$ as the aggregation result for N, where f is the aggregation function for subscription s. In this model, the f function only accepts the entire set I_N, meaning that aggregation for a subset of I_N is not allowed. This restriction allows the solution space to remain solvable in polynomial time. Furthermore, we assume without loss of generality that each NWR is non-empty since empty windows do not generate any notifications.

We define O_b as the set of notifications a broker outputs for neighbor broker b. O_b is defined over the same domain as I, which contains data elements which are either raw publications or results.

For instance, let us consider a broker who is serving subscription s_1 in Example 1 (see Figure 1(a)). I contains one publication p_1, and an aggregation result $r_1 = f(\{p_3, p_4\})$ matching N_1^1, i.e., the first NWR of subscription s_1 in S_b coming from a neighbor broker b. In this situation, $I_{N_1} = \{p_1, r_1\}$. Let assume that N_1^1 is aggregated, leading to the creation of $f(I_{N_1^1}) = f(\{p_1, r_1\})$. The output set $O_b = \{f(I_{N_1^1})\}$: O_b contains the result to be sent over to b. If the decision is to not aggregate, then $O_b = \{p_1, r_1\}$: O_b contains the elements in $I_{N_1^1} = \{p_1, r_1\}$.

To ensure that all aggregate subscriptions receive the correct results with all their matching publications taken into account, a candidate decision solution for *MNA*(I, S_b) must generate, for each broker to each of its neighbor broker b, a set of notifications O_b which satisfy the following correctness criteria:

$$\forall s \in S_b, \forall N \in s : f(I_N) \in O_b \vee I_N \subseteq O_b$$

This property states that for each NWR N of subscriptions known from broker b, the local broker must send either the aggregation result $f(I_N)$, or all the elements I_N required to compute it. As a result, the broker b receives enough information to produce all the aggregation result $f(I_N)$.

We present the *Minimum-Notifications-for-Aggregation* problem: for a given broker and its neighbor broker b, an optimal solution for *MNA*(I, S_b) generates a set of notifications O_b amongst all possible sets satisfying the correctness criteria such that $|O_b|$ is minimal.

B. Reduction to Minimum-Vertex-Cover

Minimum-Notifications-for-Aggregation can be reduced to Minimum-Vertex-Cover over an undirected bipartite graph, which is solvable in polynomial time using a maximum matching algorithm according to König's theorem [33], [34]. This demonstrates that an MNA solution can be computed by solving Minimum-Vertex-Cover on a graph constructed using the local knowledge of a broker. In this reduction, we assume that a broker is not allowed to decompose aggregation results to be shared by multiple NWRs. Relaxing this assumption leads to an increase in complexity that cannot be handled by this reduction.

For an instance of the problem called *MNA*(I, S_b), we map the sets I and S_b for a broker and its neighbor broker b to a bipartite graph. Let $G = (U, V, E)$, where G is a bipartite graph with vertices $U \cup V$ and edges E. We construct the vertex set $U = \{u(i) | \forall i \in I\}$. Each data item in I is therefore mapped uniquely to a vertex in U. We construct $V = \{v(N) | \forall N \in s, \forall s \in S_b\}$ similarly to map each NWR of subscriptions in S_b to a vertex in V. The edge set is constructed as follows: $E = \{(u(i), v(N)) | \forall s \in S_b, \forall N \in s, \forall i \in I_N\}$. The

edges of the graph connect input data items from I to matching NWRs from subscriptions in S_b.

This graph construction is performed in $O(|I| + card_A(N) \times |N|)$, where N is the set of all NWRs for subscriptions in S_b and $card_A(N)$ is the average number of data items contained in each NWR. Since the matching graph only needs to take into account data items matching at least one NWR, we can traverse the set of NWRs once and build edges for each data item matching. Lookup for data items can be performed in amortized $O(1)$ time using a hash table after performing $|I|$ insertions.

Suppose we have M, a minimum vertex cover on graph G. According to the definition of vertex cover, M has the following property:

$$\forall (u, v) \in E : u \in M \vee v \in M$$

Figure 1(b) shows the matching graph generated from Example 1. The vertices outlined in blue (N_2^2,r_1,p_1) form a minimum vertex cover set for the graph.

We can use a one-to-one mapping to create a set M' from a set M as follows:

$$M' = \{i | \forall u(i) \in M\} \cup \{f(I_N) | \forall v(N) \in M\}$$

The set M' mapped from M contains the elements in I corresponding to the vertices coming from U, while it contains the aggregation results for the NWRs associated to the vertices in V. Note that M' can be mapped back to M by using the inverse mapping. Since the mapping is one-to-one, $|M| = |M'|$.

We demonstrate that the vertex cover M can be mapped to a candidate solution M' for aggregation as it satisfies the correctness property described in Section VI-A:

Lemma 1. *The vertex cover M obtained from G can be mapped to a candidate solution M' for which the correctness property for MNA(I, S_b) at a broker and its neighbor broker b holds:*

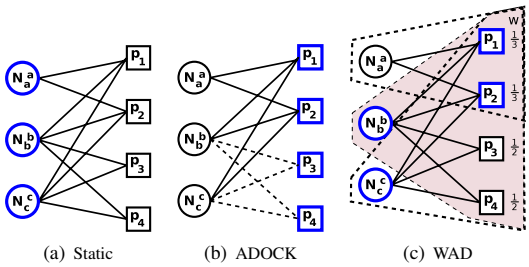
$$\forall s \in S_b, \forall N \in s : f(I_N) \in M' \vee I_N \subseteq M')$$

Proof: Proof by contradiction. Suppose the property is not enforced. Then $\exists s \in S_b, \exists N \in s$ *s.t.* $f(I_N) \notin M' \wedge (\exists i \in I_N$ *s.t.* $i \notin M')$. By virtue of our mapping, $\exists s \in S_b, \exists N \in s$ *s.t.* $v(N) \notin M \wedge (\exists i \in I_N$ *s.t.* $u(i) \notin M)$ This implies $\exists (u(i), v(N)) \in E$ *s.t.* $u(i) \notin M \wedge v(N) \notin M$, which contradicts the vertex cover property of M. ■

Lemma 1 shows that M' contains sufficient information for a broker to compute the aggregation. Lemma 2 establishes that any candidate solution set O_b for aggregation can be mapped to a vertex cover O'_b on the constructed graph G:

Lemma 2. *A solution set O_b for MNA(I, S_b) at a broker and its neighbor broker b can be mapped to a vertex cover O'_b for graph G.*

Proof: Proof by contradiction. Suppose we have a set O_b which satisfies the correctness criteria for aggregation, but for which the corresponding O'_b does not satisfy the vertex cover property. For O'_b, we observe that $\exists (u(i), v(N)) \in E$ *s.t.* $u(i) \notin O'_b \wedge v(N) \notin O'_b$. However, this means $\exists s \in S_b, \exists N \in s$ *s.t* $f(I_N) \notin O_b \wedge (\exists i \in I_N$ *s.t.* $i \notin O_b)$, which violates the correctness criteria for aggregation. ■



(a) Static (b) ADOCK (c) WAD

Fig. 3: Matching graphs

With Lemma 1 and 2, we can prove the reduction:

Theorem 1. *Minimum-Notifications-for-Aggregation can be reduced to Minimum-Vertex-Cover by constructing a minimum vertex cover M for graph G derived from MNA(I, S_b) at a broker and its neighbor broker b. The mapping set M' from M is an optimal solution for Minimum-Notifications-for-Aggregation.*

Proof: Proof by contradiction. Suppose we have a solution M for Minimum-Vertex-Cover over G. Using Lemma 1, M' from M is a candidate solution for aggregation. Now suppose we have an optimal solution set O_b for *MNA*(I, S_b) such that $|O_b| < |M'|$. According to Lemma 2, O'_b mapped from O_b is a vertex cover of graph G. However, since $|O'_b| < |M|$, a contradiction arises since M is minimal vertex cover for G. Therefore, O_b cannot exist, which means M' is an optimal solution for *MNA*(I, S_b). ■

We conclude that MNA can be solved in polynomial time by first performing the graph construction, and then computing the minimum vertex cover of the bipartite graph. The resulting minimum vertex cover corresponds to the set of data items to be sent to the next hop. Vertices selected from the U set correspond to input data items that are forwarded as is, while vertices selected from V correspond to NWRs that need to be aggregated such that $f(I_N)$ is sent to the next hop.

C. Static implementation

In this section, we summarize implementation considerations for the above theorem. To implement an MNA-based solution, each broker needs to maintain an undirected bipartite graph (called the *matching graph*) for each outgoing link where the two disjoint sets of nodes represent publications and NWRs. In this matching graph, an edge connecting a publication to an NWR node represents the publication matching the particular NWR. At the arrival of a publication, a publication node and the corresponding edges to all matching NWRs are added to the matching graph. Two NWRs are said to be interconnected if there is at least one publication matching these two NWRs. Figure 3(a) shows an example graph created from the NWRs and publications. Once all publications have been received, the minimum vertex cover is determined. In this example, the minimum vertex cover set contains three nodes corresponding to $f(I_{N_a^a})$, $f(I_{N_b^b})$ and $f(I_{N_c^c})$. In other

words, we aggregate the three NWRs and send a total of three notification messages. The complete matching graph, which includes all NWRs and all publications, is used to determine the minimum number of notifications. This also implies that aggregation decisions are computed only after receiving all publications. This modus operandi is unrealistic for all practical purposes as it defers the aggregation decision until complete publication knowledge is acquired. Moreover, executing the decision algorithm on a graph containing large numbers of publications and NWRs creates a computational burst. Next, we discuss how these assumptions can be relaxed to obtain a practical solution.

VII. Dynamic approaches

We now investigate dynamic solutions, where aggregation decisions are made at runtime without complete knowledge of the pub/sub state, specifically of future publications. Upon receipt of a new publication, the matching graph is updated by adding a publication node and the edges to all matching NWRs are added to the graph maintained by the broker. We constantly prune the graph by removing processed publication and NWR nodes. We also remove disconnected edges and vertices. Thus, the matching graph contains only publications which are involved in future aggregation decisions. As discussed earlier, the lack of complete knowledge causes any dynamic solution to generate more notifications than the static optimal solution. In this section, we provide the details of our two dynamic solutions.

A. Aggregation Decision, Optimal with Complete Knowledge (ADOCK) approach

The ADOCK approach makes decisions by computing the minimal vertex cover over a matching graph at runtime. When processing an NWR, Algorithm 1 is invoked for each matching outgoing link to decide whether to send individual matching publications or aggregate them first and send the result. This algorithm, inspired by [35], determines if the NWR node v is part of the vertex cover. First, the algorithm computes the maximum matching of graph G using the Hopcropft-Karp algorithm [33]. From the König's theorem [34], we know that the size of the maximum matching is equal to the number of vertices of the minimum vertex cover (which corresponds to the minimum number of messages that we must send). In the next step, we temporarily extend this graph to G_2 by adding two "dummy" publication nodes u_1 and u_2, and we connect them to v. The addition of these publications enforces v to be part of the vertex cover. Now after computing maximal matching again, if v is part of the vertex cover then the size of the maximum matching set of G_2 is the same as G. If not, then v is not in the minimum vertex cover. Therefore if the size of the two maximal matchings are equal, the algorithm returns true to aggregate v and false otherwise (i.e., to send all matching publications). The running cost of this decision algorithm is $O(\sqrt{V}E)$ where V and E are set of all nodes and edges in the matching graph. As shown in the description of the matching graph (see Section VI-C), the nodes are formed out of NWRs and publications. Hence, the running complexity can also be represented as $O(\sqrt{|N|+|P|} \times deg_A(N) \times |N|)$ where N and P are the sets of all NWRs and publications respectively, and $deg_A(N)$ is the average degree of an NWR node.

Input: v: node corresponding to the NWR;
G=(V,E): current matching graph
Output: *aggregate*: true if the NWR should be aggregated

$max_1 := |\mathit{MaxMatching}(G)|$
$G_2 := (V \cup \{u_1, u_2\}, E \cup \{(v, u_1), (v, u_2)\})$
$max_2 := |\mathit{MaxMatching}(G_2)|$
if *max1=max2* **then**
  aggregate := *true*
else
  aggregate := *false*
end

Algorithm 1: ADOCK decision algorithm for an NWR

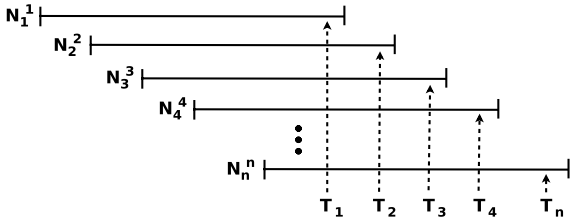


Fig. 4: Scenario with suboptimal decisions for ADOCK

We demonstrate the above process using our previous example used to describe the static approach. After receiving the two first publications, the resulting matching graph is represented in Figure 3(b). At this stage, the broker does not have information about p_3 and p_4, therefore the nodes for these publications and their edges (dotted lines) are unknown. When N_a^a expires, our decision algorithm instructs the broker to forward publications p_1 and p_2, as N_a^a is not part of the minimum vertex cover. The graph is then updated by removing the nodes for N_a^a, p_1, and p_2 as well as their associated edges. After the arrival of p_3 and p_4, the corresponding nodes and edges are added. At the expiry of N_b^b, the algorithm decides to forward publications p_3 and p_4 before deleting the associated nodes and edges. Finally, when the last N_c^c expires, as there is no matching publication left, no notification is sent and this NWR node is deleted. Note that this last NWR will be aggregated at subsequent brokers to which the relevant publications have been forwarded.

Suboptimal decision scenario: As presented earlier, ADOCK dynamically decides based on an incomplete matching graph. Using partial knowledge can lead to suboptimal choices. To illustrate this case, assume that one broker has n different NWRs registered as shown in Figure 4. At time T_1, let $n-1$ publications arrive which match all these NWRs. At the end of N_1^1, ADOCK chooses to forward publications (rather than aggregate) as the number of aggregation notifications is n compared to $n-1$ publications. In other words, this broker decides to send $n-1$ publication messages. After this decision, assume that $n-2$ publications arrive at T_2 which match the remaining $n-1$ NWRs (since N_1^1 has expired). For the reason stated earlier, $n-2$ publication messages are sent at the end of N_2^2. Let us assume that a similar scenario occurs for all remaining NWRs. In summary, this broker sends $n-1$ messages when we have n NWRs, $n-2$ for $n-1$

NWRs and so on. The total number of messages sent is $(n-1)+(n-2)+\ldots+1+1$, i.e., $(n^2-n)/2+1$. In contrast, the optimal solution consist of sending n aggregated messages. Therefore $(n-1)(n-2)/2$ additional messages are sent compared to the optimal solution.

However, our experiment results (see Table I) show that scenarios leading to incorrect decisions are rare and in practice, ADOCK performs close to the theoretical minimum. As observed in Table I, decisions taken by ADOCK are less accurate when the number of subscriptions is high, i.e., with a large number of NWRs. In such situation, the probability that a publication belongs to more than one NWR is high. Because ADOCK individually decides for each NWR at the time it expires, publications arriving later are not present. This missing information impacts ADOCK's ability to make optimal decisions as shown in the above scenario.

TABLE I: Communication cost comparison (stock-sliding)

# Subcriptions	90	180	270	360
Static vs ADOCK in %	3.53%	0.88%	4.29%	3.27%

B. Weighted Aggregation Decision (WAD) approach

The algorithm for making decisions based on the matching graph is computationally expensive and its performance does not scale with an increasing number of NWRs. We thus propose an alternative algorithm, called Weighted Aggregation Decision (WAD), which trades lower computation cost for somewhat higher communication cost. This novel dynamic solution makes aggregation decisions based on limited knowledge that consists only of publications matching an NWR.

The core idea of WAD is to leverage the following observation: the aggregation decision for one NWR depends on the number and degree of matching publications, where the degree of a publication is the number of NWRs which match the publication. If, for an NWR, the ratio of the number to the degree of publications is high, the optimal decision tends to be to aggregate.

The WAD algorithm functions by summing the weight of publications for an NWR and comparing it to a threshold of 1. A publication p is assigned the weight of $weight_p = 1/deg(p)$ where $deg(p)$ is the degree of p. This weight of a publication is therefore inversely proportional to the number of NWRs it is matching. For example, if a publication matches only one NWR, the optimal decision is to aggregate for this NWR, regardless of other publications. The overall algorithm is presented in Algorithm 2. Note that similar to previous approaches, the decision will be taken for each outgoing links for an NWR separately.

Figure 3(c) presents the graphs generated from our previous example. As shown in this figure, the weight of publication p_1 is $1/3$ as it is matching three NWRs. Similarly, the weights for p_2, p_3 and p_5 are $1/3$, $1/2$ and $1/2$ respectively. When making a decision for N_a^a, the algorithm computes a total weight of $2/3$ which is smaller that the trigger value of 1. Hence, publications p_1 and p_2 are forwarded. The computed weight for both N_b^b and N_c^c will be 1, which implies that p_3 and p_4 are aggregated.

Input: w: The window set of publications matching the NWR
Output: *aggregate*: true if the NWR should be aggregated

```
1 weight := 0
2 for p ∈ w do
3 |   weight := weight + weight_p
4 end
5 if weight ≥ 1 then
6 |   aggregate := true
7 else
8 |   aggregate := false
9 end
```

Algorithm 2: WAD's decision algorithm for an NWR

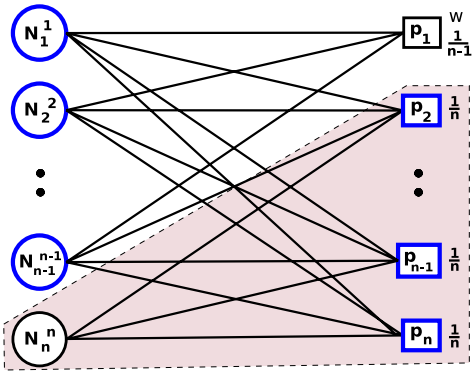


Fig. 5: Suboptimal decision scenario for WAD

Suboptimal decision scenario: The WAD algorithm employs a weight-based heuristic which can lead to suboptimal decisions, as shown in the following example. Assume that one broker has n NWRs and n publications (see Figure 5) such that first publication matches all NWRs except the last one. In this situation, the weight of this publication is $1/(n-1)$. The rest of $n-1$ publications match all NWRs which implies a weight of $1/n$ for each of them.

When processing the first $n-1$ NWRs, the weight of the first publication is $1/(n-1)$. Since the sum of weights of remaining $n-1$ publications is $(n-1)/n$, this leads to a total of $(n^2-(n-1))/(n^2-n)$ which is greater than 1. This causes the broker to aggregate and send $n-1$ messages for these NWRs. For the last NWR, the sum of publications weights computed by the broker will be $(n-1)/n$ (smaller than 1), and thus the broker decides to forward the matching $n-1$ publications. In total, we send $2(n-1)$ messages from WAD approach. For this matching graph the minimum vertex cover set has n elements, therefore optimal solution sends only n messages, that is $n-2$ fewer messages.

VIII. Experimental evaluation

This section presents experimental evaluation conducted for our proposed approaches. In particular, our experiments explore the trade-off between the communication and computation cost. We use the total number of messages exchanged among brokers as the metric for communication cost. We

measure the computation cost by summing the processing time of two components across all brokers: the matching time for each publication and the time to make an aggregation decision for each NWR. We also compare our approach with a solution presented in [16], referred to as "per-Broker Adaptive aggregation Technique" (*BAT*).

A. Setup

We use two different datasets to evaluate our solutions. The first dataset contains traces from real traffic monitoring data extracted from the ONE-ITS service [14]. The publications for this Traffic dataset are produced by 162 sensors located at major highways in Toronto. These publications contain 12 distinct fields[2]. We also use a dataset from a stock market application (fetched from daily stock data of Yahoo! Finance). This Stock dataset is commonly used to evaluate pub/sub systems [36]. In the experiments with this dataset, we use 62 stock symbols with each stock publication containing 8 distinct fields. Note that the high-dimensional traffic workload exhibits higher filtering cost than the Stock data. We also keep the selectivity of subscriptions [37] to around 1% for the Traffic dataset and 2% for the Stock dataset.

We implemented our approaches in Java using the PADRES pub/sub system [18]. The broker overlay is deployed on a cluster of 16 servers with Intel(R) Xeon(TM) 3.00 GHz CPUs. Each broker runs in a JVM on a separate machine with a minimum of 256 MB to a maximum of 2 GB memory allocation. In this setup, 4 brokers are arranged as a chain of core brokers where each core broker has 3 edge brokers attached. Among the 12 edge brokers, 3 brokers have publishers, 3 brokers have subscribers, and the remaining 6 brokers have a mix of subscribers and publishers attached.

We vary two parameters in our experiments: the publication rate ranges from 14 to 1080 publications per second for the Traffic dataset and from 465 to 2790 publications per second for the Stock dataset. The number of subscriptions varies from 9 to 450 for both datasets. Our subscription workload consists of a mix of aggregate and regular subscriptions at a ratio of 1:2. Our experiments include subscriptions for both tumbling and sliding window semantics. For tumbling window subscriptions in either dataset, we choose a duration of 10 seconds for both window and shift size ($\delta = \omega = 10$ seconds). For sliding window subscriptions, the window size is also 10 seconds, but the shift size is only 5 seconds ($\omega = 10$ seconds, $\delta = 5$ seconds). Each experiment runs for 1000 seconds, which represents a maximum of 100 NWRs for both settings. This allows us to observe the effect of using sliding windows without varying the number of NWRs.

B. Communication cost

In this section, we explore the impact of the publication rate and the number of subscriptions on the communication cost.

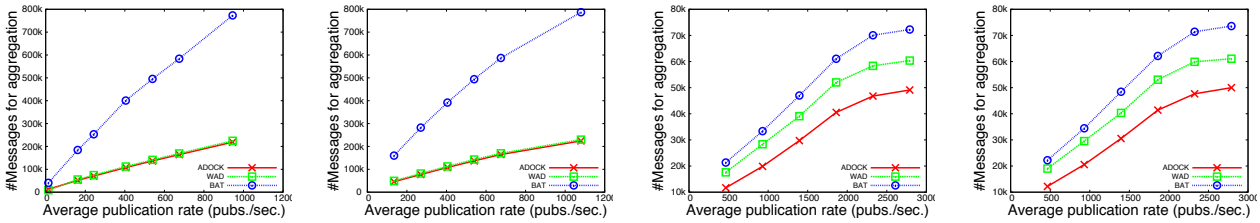
1) Publication rate: Figures 6 and 7 compare the communication and computation costs for the ADOCK, WAD and BAT aggregation techniques at different publication rates. Figures 6(a) and 6(b) show the communication cost for the Traffic dataset with tumbling and sliding windows, respectively, while Figures 6(c) and 6(d) show results for the Stock data.

In general, we can observe that ADOCK is the most efficient solution in terms of number of notifications across all experiments, and that the BAT baseline performs the worst across the board. Note that ADOCK performs close to the optimal static baseline (as shown in Table I, the maximum difference is only 4.29 % in the experiment). WAD performs similarly to ADOCK on the Traffic dataset but is noticeably less efficient on the Stock dataset. This is due to subscriptions in the Traffic dataset using a relatively larger number of fields, resulting in a reduced selectivity of publications (1%) compared to those of the Stock dataset (2%). By rendering publications less selective, the interconnectivity of NWRs decreases as well. As discussed in Section VII-B, WAD makes better decisions when NWRs have a smaller overlap, which allows WAD to perform close to ADOCK for the Traffic experiments. For instance, at 1080 pub/s WAD has only 2.7% more communication overhead than ADOCK (Figure 6(a)). In contrast, for the Stock dataset (Figure 6(c)), the increase in NWR interconnectivity when raising the publication rate negatively impacts the performance of WAD. However, the margin of error by WAD decreases as publication rates increase. For instance, at 2790 pub/s, WAD sends only 22% more publication than ADOCK compared to 50% at 465 pub/s. This happens when the average degree of NWRs is sufficiently large. Due to a high publication rate, WAD tends to always decide to aggregate NWRs rather than forward publications, which is in line with the decisions ADOCK makes at that publication rate. Such reduction is also present in the Traffic experiments where the overhead shrinks from 7% at 162 pub/s to 2.7% at 1080 pub/s (Figure 6(a)).

In the sliding window experiments for both datasets (see Figures 6(b) and 6(d)), all techniques produce a similar number of messages compared to their tumbling window equivalents (see Figures 6(a) and 6(c)). For example, the difference between total messages sent by ADOCK in the sliding and tumbling settings is at most 4.6% at 465 pub/s rate and at least 1.8% at 2790 pub/s (Figure 6(d)). Since we fixed the number of NWRs to be equal in both settings, the difference in communication cost is accounted by the increased interconnectivity of NWRs in the sliding window experiments. However, the increased overhead due to interconnectivity is offset by smaller gaps in time between consecutive NWR decisions which translates in a small overall impact on the communication load.

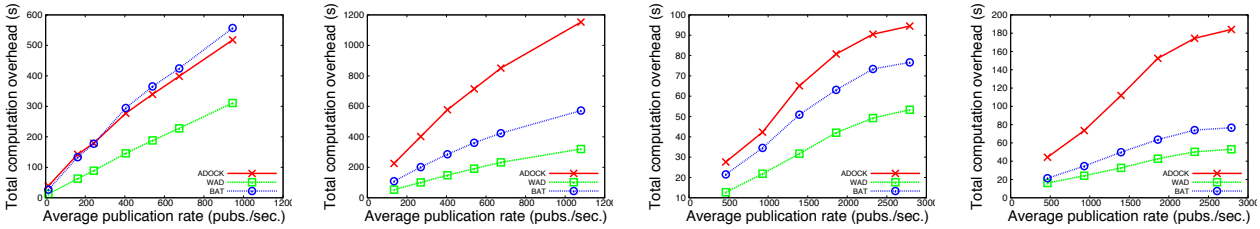
2) Subscription rate: Figure 8(a) shows the communication cost when varying the number of subscriptions. We can observe that the cost of communication increases for all techniques according to the number of subscriptions registered. This is because more messages are required to serve additional subscriptions. As expected, we find that ADOCK has the lowest communication cost while WAD generates slightly more messages, due to the suboptimal decisions it takes. However, the relative difference between the total number of messages sent by WAD compared to that of ADOCK shrinks as the number of subscriptions increases. For example, at 90 subscriptions, WAD has 28.67% additional overhead whereas for 360 subscriptions it costs only 12.7% more (see Figure 8(a)).

[2]http://msrg.org/datasets/traffic



(a) Comm. cost (traffic-tumbling) (b) Comm. cost (traffic-sliding) (c) Comm. cost (stock-tumbling) (d) Comm. cost (stock-sliding)

Fig. 6: Communication cost with variable rate of publications



(a) Comp. cost (traffic-tumbling) (b) Comp. cost (traffic-sliding) (c) Comp. cost (stock-tumbling) (d) Comp. cost (stock-sliding)

Fig. 7: Computation cost with variable rate of publications

With a large number of NWRs, the decisions taken by brokers using WAD are less prone to error.

C. Computation cost

This section presents a study of computation overhead while varying the publication rate and number of subscriptions.

1) Publication rate: The graphs in Figure 7 show the total computation time when varying the publication rate for our different workloads. Figure 7(a) shows the total computation cost for the Traffic experiments using tumbling windows. As expected, ADOCK exhibits a higher computation cost compared to WAD. Surprisingly, BAT has a computation cost similar to ADOCK which is due to the processing required to handle the extra communication (see Figure 6(a)).

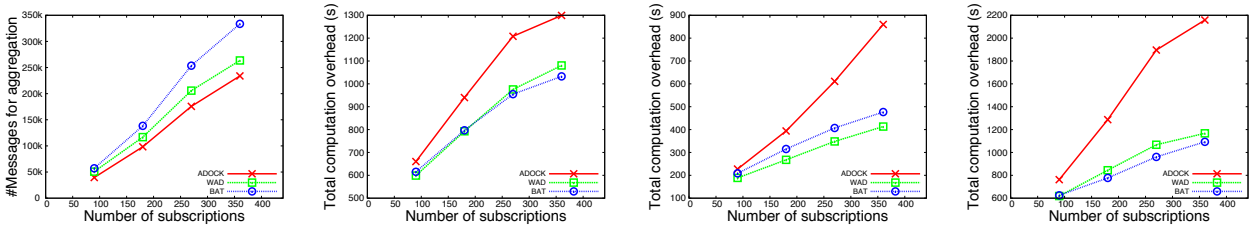
As stated earlier, in the sliding window setting, the size of the decision graph increases as NWRs are more likely to be interconnected. We observe in Figure 7(b) that the computation cost for ADOCK reaches almost 1200s for the sliding setting while it is only 500s for the tumbling setting. This difference is explained by the computation cost required to process bigger decision graphs. In contrast, we observe that WAD and BAT have a similar computation cost for tumbling and sliding settings. BAT is based on a heuristic with a cost independent of the size of the decision graph. For WAD, the cost of the decision effectively increases: However, it accounts for a negligible part of the overall cost which consists mainly of matching. Therefore, the increase in computation cost is not visible as shown in Figure 7(b).

For the Stock dataset, WAD and BAT also have a similar computation cost for both tumbling and sliding settings (see Figures 7(c) and 7(d)). For the tumbling setting (Figure 7(c)), BAT performs better than ADOCK which differs from the observation made in the Traffic dataset. The explanation lies in the difference in matching cost. For this setting, BAT is sending at most 20K more messages than ADOCK (Figure 6(a)), which results in a moderate increase in matching cost. In contrast, for the Traffic dataset, BAT sends nearly 600K more messages than ADOCK, which causes the matching cost to equal ADOCK's decision cost.

2) Subscription rate: We observe in Figure 8(c) that the computation time of ADOCK grows faster than a linear function when the number of NWRs increases. In contrast, the computation cost for WAD varies linearly relative to the number of NWRs. We can explain this observation by comparing the running time complexity of both approaches. From Section VI-C, we know that the complexity of ADOCK is $O(\sqrt{|N|+|P|} \times deg_A(N) \times |N|)$ per decision where N, P and $deg_A(N)$ denote the set of NWRs, set of publications, and the average number of publications per NWR, respectively. Therefore, the total computation cost for all NWRs is $O(\sqrt{|N|+|P|} \times deg_A(N) \times |N|^2)$. Whereas for WAD, the total computation cost for all NWRs is $O(deg_A(N) \times |N|)$.

Figure 8(d) shows that ADOCK's computation cost is greater than WAD's, but the gradient gradually decreases for larger number of subscriptions. This is because the number of NWRs does not necessarily increase linearly to the number of subscriptions: an NWR is created only if there is at least one matching publication. This phenomenon is slightly more visible with tumbling window subscriptions than with sliding window ones (see Figure 8(b)).

We can also observe that the WAD approach incurs a computation cost similar to that of BAT but with a more effective traffic reduction gain. As shown in Figures 8(b) and 8(d), we also notice that the higher interconnectivity of NWRs in the Stock dataset causes the difference in the computation cost



(a) Comm. cost (stock-tumbling) (b) Comp. cost (stock-tumbling) (c) Comp. cost (traffic-tumbling) (d) Comp. cost (stock-sliding)

Fig. 8: Dependency of communication and computation cost on the number of subscriptions

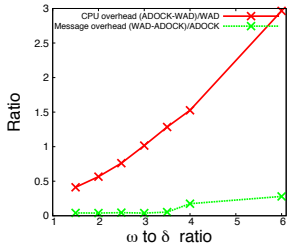


Fig. 9: Impact of sliding windows

between WAD and BAT to be more significant compared to the Traffic experiments.

D. Impact of sliding windows

To highlight the impact of sliding windows, we conduct a sensitivity analysis by varying the window size(ω) to shift size (δ) ratio. Figure 9 compares the relative performance of ADOCK and WAD for both communication and computation loads.

When the ratio increases, we observe that the difference between WAD and ADOCK also increases in terms of communication and computation cost. For example, at a ratio of 1.5, ADOCK requires about 41% more time than WAD to make a decision while WAD generates 4% more messages than ADOCK. At a ratio of 6, ADOCK requires around 300% additional computation time while reducing 27% in the communication overhead.

As we raise the ratio, the NWR interconnectivity increases and makes the decision graph bigger. Taking a decision based on a larger graph requires more computation cost which is disadvantageous for ADOCK. On the other hand, since WAD uses partial information about the graph, a decision on a larger graph becomes less accurate.

In light of our observations, WAD presents an interesting compromise as it trades a 25% higher communication cost for a 3 times lower computation time over ADOCK. We conclude that WAD is a better solution for workloads containing subscriptions with a high sliding ratio.

E. Summary

From our experimental results we conclude that, when varying the publication rate, the communication cost for ADOCK is up to 50% lower than that for WAD but at the expense of a computation overhead up to four times greater. However, as the publication rate increases, the relative difference in the communication cost between these two approaches decreases in some configurations. On the other hand, the computation cost difference increases for higher publication rates.

When varying the number of subscriptions, the communication cost for WAD is up to 29% higher than that for ADOCK. On the other hand, the computation cost of ADOCK is up to 2.1 times the computation cost of WAD. This shows that a trade-off exists between the computation and communication cost: While ADOCK performs slightly better in terms of the communication cost, this is offset by WAD showing a clear advantage in terms of the computation cost.

From the results in our setup we also conclude that for the Stock dataset, ADOCK is a better candidate as it has a lower communication cost than WAD provided that the application can afford a higher computation cost. On the other hand, WAD is more suitable for the Traffic dataset as it yields a significantly lower computation cost yet a similar communication cost compared to ADOCK.

As mentioned in Section VIII-B, the interconnectivity among the NWRs is the major source of the trade-off between communication and computation cost and hence between WAD and ADOCK. If the subscriptions have sliding windows or high selectivity, a high interconnectivity is expected. Overlapping among subscriptions [38] is also a prominent factor that increases interconnectivity. From the experiments with Stock and Traffic datasets we observe that, if the system expects a moderate amount of subscriptions with high selectivity, ADOCK is a better candidate. Otherwise, WAD is recommended.

IX. Conclusions

In this paper, we study the problem of minimizing the number of notifications for aggregation in pub/sub systems, which we refer to as *Minimum-Notifications-for-Aggregation*. Our theoretical study reveals that it is possible to achieve minimal communication cost under the unrealistic assumption of complete knowledge of future publications. We also demonstrate that applying this solution (ADOCK) at runtime leads to suboptimal, yet still efficient, results. We devise an

alternative solution, named WAD, which is computationally cheaper. We evaluate our proposed approaches using two real datasets extracted from our traffic monitoring and stock market application scenarios. Our experimental results highlight the existence of a trade-off between computation and communication cost: ADOCK exhibits a communication cost lower than WAD but at the expense of a computation cost increase.

REFERENCES

[1] 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." VLDB Endowment, 2004.

[2] S. F. Abelsen, H. Gjermundrd, D. E. Bakken, and C. H. Hauser, "Adaptive data stream mechanism for control and monitoring applications," in *Computation World*, 2009.

[3] M. Jelasity, A. Montresor, and O. Babaoglu, "Gossip-based aggregation in large dynamic networks," *ACM Trans. Comput. Syst.*, vol. 23, no. 3, 2005.

[4] P. Jesus, C. Baquero, and P. S. Almeida, "A survey of distributed data aggregation algorithms," University of Minho, Tech. Rep., 2011.

[5] G. Li *et al.*, "A distributed service-oriented architecture for business process execution," *ACM Trans. Web*, 2010.

[6] M. Sadoghi, M. Jergler, H.-A. Jacobsen, R. Hull, and R. Vaculín, "Safe distribution and parallel execution of data-centric workflows over the publish/subscribe abstraction," *IEEE Trans. Knowledge and Data Engineering*, 2015.

[7] Y. Tock, N. Naaman, A. Harpaz, and G. Gershinsky, "Hierarchical clustering of message flows in a multicast data dissemination system," in *Proc. of PDCS*, 2005.

[8] 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*, 2013.

[9] A. Demers, J. Gehrke, M. Hong, M. Riedewald, and W. White, "Towards expressive publish/subscribe systems," in *Proc. of EDBT*, 2006.

[10] J. Sventek and A. Koliousis, "Unification of publish/subscribe systems and stream databases: the impact on complex event processing," in *Proc. of Middleware*, 2012.

[11] B. Chandramouli and J. Yang, "End-to-end support for joins in large-scale publish/subscribe systems," *Proc. VLDB Endow.*, vol. 1, no. 1, 2008.

[12] R. van Renesse and A. Bozdog, "Willow: DHT, aggregation, and publish/subscribe in one protocol," in *Proc. of IPTPS*, 2004.

[13] T. Rabl, S. Gómez-Villamor, M. Sadoghi, V. Muntés-Mulero, H.-A. Jacobsen, and S. Mankovskii, "Solving big data challenges for enterprise application performance management," *Proc. VLDB Endow.*, 2012.

[14] "ONE-ITS Online Network-Enabled Intelligent Transportation Systems." [Online]. Available: http://one-its-webapp1.transport.utoronto.ca

[15] D. Qiu, K. Zhang, and H.-A. Jacobsen, "Smart Phone Application for Connected Vehicles and Smart Transportation," in *Middleware Posters*, 2013.

[16] N. K. Pandey, K. Zhang, S. Weiss, H.-A. Jacobsen, and R. Vitenberg, "Distributed event aggregation for content-based publish/subscribe systems," in *ACM DEBS*, 2014.

[17] A. Carzaniga, D. S. Rosenblum, and A. L. Wolf, "Design and evaluation of a wide-area event notification service," *ACM Tran. on Computer Systems*, vol. 19, no. 3, 2001.

[18] E. Fidler, H.-A. Jacobsen, G. Li, and S. Mankovskii, "The PADRES distributed publish/subscribe system," in *Proc. of ICFI*, 2005.

[19] 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.*, vol. 21, no. 2, 2003.

[20] P. Yalagandula and M. Dahlin, "Shruti: A Self-Tuning Hierarchical Aggregation System," in *Proc. of SASO*. IEEE Computer Society, 2007.

[21] T. Repantis and V. Kalogeraki, "Hot-spot prediction and alleviation in distributed stream processing applications," in *Proc. of DSN*, 2008.

[22] H. Shen, "Content-based publish/subscribe systems," in *Handbook of P2P Networking*. Springer, 2010.

[23] R. Baldoni, L. Querzoni, S. Tarkoma, and A. Virgillito, "Distributed event routing in publish/subscribe systems," in *MNEMA*. Springer, 2009.

[24] M. Wood and K. Marzullo, "The design and implementation of Meta," in *Reliable Distributed Computing with the ISIS Toolkit*, 1994.

[25] L. Golab, K. G. Bijay, and M. T. Özsu, "Multi-query optimization of sliding window aggregates by schedule synchronization," in *Proc. of CIKM*, 2006.

[26] N. Jain, D. Kit, P. Mahajan, P. Yalagandula, M. Dahlin, and Y. Zhang, "STAR: Self-tuning aggregation for scalable monitoring," in *VLDB*, 2007.

[27] S. Krishnamurthy, C. Wu, and M. Franklin, "On-the-fly sharing for streamed aggregation," in *Proc. of SIGMOD*, 2006.

[28] R. Huebsch, M. Garofalakis, J. M. Hellerstein, and I. Stoica, "Sharing aggregate computation for distributed queries," in *Proc. of SIGMOD*, 2007.

[29] L. Brenna, J. Gehrke, M. Hong, and D. Johansen, "Distributed event stream processing with non-deterministic finite automata," in *Proc. of DEBS*, 2009.

[30] A. Cheung and H.-A. Jacobsen, "Publisher placement algorithms in content-based publish/subscribe," in *Proc. of ICDCS*, 2010.

[31] A. K. Y. Cheung and H.-A. Jacobsen, "Green resource allocation algorithms for publish/subscribe systems," in *Proc. of ICDCS*, 2011.

[32] J. Chen, L. Ramaswamy, and D. Lowenthal, "Towards efficient event aggregation in a decentralized publish-subscribe system," in *ACM DEBS*, 2009.

[33] J. Hopcroft and R. Karp, "An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs," *SIAM Journal on Computing*, vol. 2, no. 4, 1973.

[34] D. König, "Graphs and matrices," in *Mat Fiz Lapok 38 (in Hungarian)*, 1931.

[35] M. Soltys and A. Fernandez, "A linear-time algorithm for computing minimum vertex covers from maximum matchings," McMaster University, Hamilton, Canada, Tech. Rep., 2012.

[36] A. K. Y. Cheung and H.-A. Jacobsen, "Publisher placement algorithms in content-based publish/subscribe," *Proc. of ICDCS*, 2010.

[37] K. R. Jayaram, C. Jayalath, and P. Eugster, "Parametric subscriptions for content-based publish/subscribe networks," in *Middleware*, 2010.

[38] Z. Liu, S. Parthasarthy, A. Ranganathan, and H. Yang, "Scalable event matching for overlapping subscriptions in pub/sub systems," in *Proc. of DEBS*, 2007.