Infrastructure Provisioning for Scalable Content-based Routing: Framework and Analysis

Raphaël Barazzutti, Pascal Felber, Hugues Mercier, Emanuel Onica, Jean-François Pineau and Etienne Rivière
Department of Computer Science
Université de Neuchâtel
Neuchâtel, Switzerland
Email: first.last@unine.ch

Christof Fetzer
Department of Computer Science
Technische Universität Dresden
Dresden, Germany
Email: first@se.inf.tu-dresden.de

Abstract—Content-based publish/subscribe is an attractive paradigm for designing large-scale systems, as it decouples producers of information from consumers. This provides extensive flexibility for applications, which can use a modular architecture. Using this architecture, each participant expresses its interest in events by means of filters on the content of those events instead of using pre-established communication channels. However, matching events against filters has a non-negligible processing cost. Scaling the infrastructure with the number of users or events requires appropriate provisioning of resources for each of the operations involved: routing and filtering. In this paper, we propose and describe a generic, modular, and scalable infrastructure for supporting high-performance content-based publish/subscribe. We analyze its properties and show how it dynamically scales in a realistic setting. Our results provide valuable insights into the design and deployment of scalable content-based routing infrastructures.

Keywords—Content-based publish/subscribe; scalable infrastructure; modularity.

I. INTRODUCTION

The publish/subscribe (Pub/Sub) communication model [1] is recognized as one of the approaches of choice for the construction of large-scale complex applications. By decoupling the communication between producers and consumers of information both in space and time, it hides the complexity of information routing between application components and presents a simpler, data-centric view to programmers. The publishers of information send publications, or events, to the system which is then responsible for dispatching them to the subscribers who previously expressed their interest in those events. One usually distinguishes between Pub/Sub systems based on the model used for expressing interests, by means of subscriptions sent to the system. Topic-based Pub/Sub, on the one hand, provides a set of predefined topics. Implementations are typically supported by a group membership protocol and application-level multicast within the group. Content-based Pub/Sub, on the other hand, allows expressing subscriptions as predicates on the content of events. This paper focuses on the content-based Pub/Sub model. This model, while more expressive, is also more complex to support. Unlike topic-based Pub/Sub, it is not possible to pre-compute the set of subscribers who will be interested in a given set of events: filtering of events against the subscriptions must be performed for each new event.

We assume in this paper that the content-based filtering is supported by a set of dedicated servers or brokers, as implemented for instance in Siena [2], PADRES [3] and XNet [4]. The brokers need to establish and maintain communication channels with the publishers in order to receive new events, and with subscribers to dispatch matching events and receive subscriptions and cancellations. Each broker implements the content filtering operation by matching all the events against its set of stored subscriptions.

Providing scalable support for content-based Pub/Sub requires being able to scale three operations: (1) communication from the publishers, (2) communication from and to the subscribers, and (3) matching the publications against the subscriptions. The resources of each individual server composing the infrastructure are obviously constrained, and the bandwidth for handling communications to and from publishers and subscribers is also limited. Moreover, the number of connections any single server can handle is constrained by OS-specific limitations, and the limited processing power for the matching operation affects the filtering delay. Achieving scalability requires splitting the load of the three aforementioned operations so as to seamlessly scale as the number of publishers, subscribers, publications and subscriptions varies.

Contributions. In this paper, we focus on the provisioning of a scalable infrastructure for content-based Pub/Sub systems. More precisely, we propose and describe a generic architecture that solves the scalability aspects of content-based routing systems and show how this infrastructure scales in a practical setting. We expect the results to help designers and engineers determine the requirements of the connection, routing, and filtering operators as the load varies. The work presented in this paper is conceptual in nature and part of a larger project whose goal is the implementation of a fast, scalable and secure content-based Pub/Sub system.
Outline. The remaining of this paper is organized as follows. Section II presents the infrastructure and defines its operators. Section III describes how to scale the components of our generic architecture for content-based routing. Section IV generalizes our approach to a stage-driven design. Section V presents experimental results, Section VI discusses related work, and finally Section VII concludes.

II. INFRASTRUCTURE MODEL AND ASSUMPTIONS

In this section, we describe our generic architecture as well as its basic operators required for a content-based system. Our Pub/Sub architecture model is organized with a collection of operators organized in stages, resulting in a directed acyclic graph traversed by every publication from a publisher to zero or more subscribers. We present our working assumptions in Subsections II-A and II-B, and describe the architecture starting from Subsection II-C.

A. Communication Assumptions

In order to analyze the communication constraints of the Pub/Sub architecture, we use the bounded multi-port communication model [5], which limits the total amount of data that a node can send and/or receive per time unit. This is a realistic assumption, and intuitively the bound corresponds to the bandwidth capacity of the node’s network card. The flow of data out of the card can be either directed to a single link or split among several links (likewise for the flow of data to the card) hence the multi-port hypothesis. Simultaneous sends and receives are allowed as long as the bound is satisfied. In the remaining of this paper, the multi-port constraint is denoted as the bandwidth cap.

We also assume that the communication capacity between the operators is homogeneous (i.e., same network links, but connecting machines with heterogeneous speeds and network cards). This assumption is realistic if the targeted deployment environment is within a single data center. The study of a wide-area system over the Internet will be considered in future work.

B. Workload Assumptions

We assume that the throughput of incoming subscriptions and cancellations is negligible compared to the throughput of publications. This is representative of Pub/Sub applications and workloads, where subscriptions remain unchanged for long periods of times while publications are constantly flowing through the system. We also assume that the machines doing the filtering have enough incoming bandwidth, in other words that the filtering process is costlier than the routing of the publications and subscriptions. Finally, we realistically assume that the stream of publications to any given subscriber does not exceed the bandwidth cap of our system components.

C. Interconnection Points

Publisher Interconnection Points (PIPs) and Subscriber Interconnection Points (SIPs), illustrated in Figures 1 and 2, are the connection points to the Pub/Sub system for publishers and subscribers, respectively. We assume that the connection between a user (either a publisher or a subscriber) and its corresponding interconnection point is persistent (e.g., TCP). One can note that this is necessary for the SIPs as subscribers may be located behind a NAT (Network Address Translation) or firewall, i.e., not reachable directly.

PIPs receive publications from connected publishers, which are then routed to the appropriate operators in the system. Each subscriber is connected to a single SIP to receive matching publications and send its subscriptions and cancellations. Moreover, SIPs are responsible for checking subscribers credentials for receiving publications, if any. It should be noted that the PIPs and SIPs could be merged into a single operator type maintaining persistent connections to and from the users. However, for simplicity and scalability purposes when the throughput from the publishers greatly differs from the throughput to the subscribers, we separate PIPs from SIPs.

D. Matching Operators

Matching operators (M), illustrated in Figure 3, receive streams of publications and match them against their local set of subscriptions. M operators output the matching publications and the list of subscribers who should receive them. The performance of an M operator depends on several factors like the size and complexity of events and subscriptions, the total number of subscriptions, and the relationships between subscriptions themselves (e.g., “containment” relations can be used for speeding up the filtering process). Such a complex behavior appears very difficult to analyze, but in practice with large systems we can model the processing speed of the M operators reasonably well as a function of the size of its dataset (set of stored subscriptions).
E. Dispatchers

Since we want to design a modular and scalable architecture, we want the PIPs, Ms and SIPs to scale independently. Therefore, we need intermediary operators, called dispatchers (D), to correctly route the messages between the interconnection points and the matching operators. Dispatchers ensure the transmission of messages independently of the number of operators, the number of transmitted messages, or the complexity of the matching operation. They can use different communication routines, such as Anycast and Multicast. Dispatchers are described in more details in the next section, and our entire framework, its operators, and its constraints when routing the publications from the publishers to the subscribers are summarized in Figure 4.

III. SCALING THE INFRASTRUCTURE

Provisioning a scalable filtering infrastructure requires taking into account a large number of parameters like the number of publishers and subscribers, the throughput of incoming publications and subscriptions, and the filtering load on the matching operators. Provisioning requires solving the following problems. First, we must determine when to scale up or scale down each part of the infrastructure. Second, for a given workload, we must find the configuration that can sustain the filtering and communication costs but use as few operators (and thus, machines) as possible. In this paper, we consider the throughput of the publications, the volume of the subscriptions, the memory of the matching operators, and the complexity of the filtering process. As mentioned in Section II, we assume that the throughput of the subscriptions and cancellations is negligible and that subscriptions and cancellations can be routed to the matching operators at a negligible cost.

Our system architecture and its operators are shown in Figure 5. In the rest of the section, we describe how to scale each of the system components.

A. Scaling with Publishers and Subscribers

As explained in Subsection II-C, the PIP and SIP operators are directly connected to publishers and subscribers, respectively. The number of open persistent connections from the users (publishers and subscribers) is limited to the number of TCP connections, typically a few hundreds on most systems. Thus, new Interconnection Points operators must be created if all the connections are taken, and Interconnection Points operators can be merged as the number of users goes down (assuming that the operators have enough available bandwidth).

B. Scaling with Publications (Matching Constraints)

In this subsection, we consider the scaling of the matching operators when they cannot accommodate all the incoming publications. Consider a single matching operator. As the publications throughput increases, it must be cloned if its queue increases and it can no longer filter the publications faster than they arrive. This is achieved by means of publication partitioning, as illustrated in Figure 6. The original matching operator is cloned, and all copies are connected by a dispatcher DP. We say clones because all the new matching operators must be provided with the full set of subscriptions managed by the original machine. The dispatcher splits the incoming stream of publications among the matching operators using anycast operations. Each event is filtered by a single matching operator, resulting in a filtering load reduced by a factor equal to the number of M...
operators (three in Figure 6) if they all have the same speed. It should be noted that subscriptions and cancellations must be multicast to all the matching operators in order to keep a consistent and complete view of the subscription set.

Let \( n_i \) be the number of publications that can be matched by the matching operator \( M_i \) per time unit for \( i \in \{1, 2, 3, \ldots\} \), and let \( T \) be the publication throughput of all the publishers in the system per time unit. Without loss of generality, we assume that \( n_1 \geq n_2 \geq n_3 \geq \ldots \), i.e., that the matching operators are sorted by speed in decreasing order. If \( n_i = n_j \) for all \( i \neq j \), then the number of matching operators required to split the publication load is at least \( \lceil \frac{T}{n_1} \rceil \). If the matching speed of the \( M \) operators is heterogeneous, each of them will receive a different share of the incoming publication throughput. To minimize the number of matching operators in such a scenario, we sort them by their speed and select the fastest ones. The number of matching operators required to split the stream of publications is at least the smallest \( j \) such that \( \sum_{i=1}^{j} n_i \geq T \).

**Dynamic scaling:** Scaling the number of matching operators as the publications throughput varies is done using a classical threshold mechanism. Since the creation of a new matching operator and the synchronization of its list of subscriptions can be a costly operation, one cannot wait for the saturation of an operator before setting up a new one. Therefore, if the incoming throughput exceeds the upper threshold (e.g., 75% of the achievable throughput), then one of the matching operators is cloned. Likewise, if the incoming throughput decreases below the lower threshold (e.g., 25% of the achievable throughput), then one of the matching operators is discarded.

**C. Scaling with Subscriptions**

As the number of subscriptions increases, the time needed to filter one publication increases. Furthermore, storing more subscriptions than can fit in the main memory of a matching operator can lead to a significant drop in performance. Thus, we limit the size of the subscription list kept at each matching engine using subscription partitioning, as illustrated in Figure 7. When a matching operator \( M \) reaches this memory limit, it is replaced by a new set of matching operators and the subscription set of the original operator is partitioned and split among the new operators. A dispatcher \( D_M \) must be used to multicast all the incoming publications to all the matching operators. Also, we point out that new subscriptions must be anycast to a single matching operator, or unicast to a specific machine if a clustering algorithm is used. Cancellations must be broadcast to all the matching operators, or unicast if their assigned operator is known.

Let \( |S| \) be the number of subscriptions in the system, and let \( m_i \) be the number of subscriptions that can be stored by the matching operator \( M_i \) for \( i \in \{1, 2, 3, \ldots\} \). Without loss of generality, we assume that \( m_1 \geq m_2 \geq m_3 \geq \ldots \). If \( m_i = m_j \) for \( i \neq j \), then the set of subscriptions must be partitioned on at least \( \lceil \frac{|S|}{m_1} \rceil \) machines. When simple linear filtering algorithms are used, partitioning the subscription set into \( k \) subsets of equal size will increase the filtering capacity by a factor of \( k \). However, with sub-linear filtering algorithms, filtering a publication over \( k \) subscription lists of size \( \alpha \) is in general more costly than filtering it over a single list of size \( k \cdot \alpha \), thus the filtering capacity is multiplied by a factor smaller than the number of \( M \) operators. If the memory size is heterogeneous, then the minimum number of matching operators for the partitioned set of subscriptions is the smallest \( j \) such that \( \sum_{i=1}^{j} m_i \geq |S| \).

**D. Dispatching Filtered Messages**

Once a publication is filtered by a matching operator, it is forwarded, along with its list of matching subscriptions, to one of several dispatchers \( D_S \) using an anycast operation. It is possible for the publication to be forwarded from more than one matching operator each choosing a different dispatcher \( D_S \). Nonetheless the lists of matching subscriptions are disjoint. The selected dispatchers \( D_S \) then partition the subscription lists using the subscribers assigned to each SIP operator and forward the publication and its smaller lists of matching subscriptions to all the appropriate SIPs. Finally, the SIPs multicast the publication to matching subscribers.

**E. Scaling with Publications (Communication Constraints)**

In a Pub/Sub system with a large publication throughput, it is possible to saturate the bandwidth of the dispatchers and interconnection points, in which case the number of operators must be increased. When a dispatcher of type \( D_P \) must be added, all the PIPs must be able to anycast publications to it. Furthermore, the new dispatcher must be able to anycast publications to all the dispatchers of type \( D_M \) and to all the matching operators containing the complete set of subscriptions. When a dispatcher of type \( D_S \) must be added, all the matching operators must be able to anycast publications.
the publications and the lists of matching subscriptions to it. The new dispatcher \(D_S\) must also be able to forward the publications and the lists of subscriptions to all the SIP operators.

The dispatchers \(D_S\) and \(D_P\) must scale independently because the incoming and outgoing throughput of the matching operators can be quite different. More precisely, although the outgoing throughput depends on the incoming throughput, it also depends on the characteristics of the subscription set: a few general subscriptions can generate more outgoing bandwidth than a large set of very selective subscriptions that only match a tiny subset of the publications.

F. Putting Everything Together

Figure 5 shows the path followed by a publication \(p\) from publisher \(P_1\) matching the set of subscriptions \(\{S_1, S_3, S_4\}\). The publication is first routed to the PIP assigned to \(P_2\). It is then forwarded to one of the dispatchers \(D_P\), which then transmits it to \(D_M\). The publication must then be filtered with the entire set of subscriptions, thus \(D_M\) multicasts it to the matching operators \(M_1, M_2,\) and \(M_3\). The next step is the filtering process: since \(p\) matches \(\{S_1, S_3, S_4\}\), \(M_1\) multicasts \(\{p, \{S_1\}\}\) to one of the dispatchers \(D_S\) and \(M_2\) multicasts \(\{p, \{S_3, S_4\}\}\). The top dispatcher \(D_S\) forwards \(\{p, \{S_1\}\}\) to the SIP assigned to \(S_1\). The bottom dispatcher \(D_S\) forwards \(\{p, \{S_3, S_4\}\}\), but since \(S_3\) and \(S_4\) are not assigned the same SIP, the dispatcher must split the subscriptions: it sends \(\{p, \{S_3\}\}\) to the top SIP and \(\{p, \{S_4\}\}\) to the bottom SIP. Finally, the top SIP conveys \(p\) to \(S_1\) and \(S_3\) while the bottom SIP conveys \(p\) to \(S_4\).

IV. A STAGE-DRIVEN ARCHITECTURE

In the previous two sections, we have seen how to provision the interconnection points, matching operators, and dispatchers independently. Both partitioning approaches for scaling the matching operators, i.e., publication and subscription partitioning, have advantages and drawbacks and should therefore be combined to split the load of either subscriptions or publications at different stages of the filtering pipeline as it increases. Conversely, a reduction of the system load can result in parts of the system collapsing into a smaller number of operators. Such a stage-driven architecture, when combined with a monitoring system, can support the dynamic scaling of resources according to the actual load experienced by the filtering infrastructure. This is similar to the elastic computing model associated to cloud computing: the amount of resources is dynamically adapted to the current application needs.

We now explain using a simple example how to scale all the resources optimally while respecting the various constraints of the system. Consider first the simplest Pub/Sub system composed of one PIP, one \(D_P\) dispatcher, one matching operator, one \(D_S\) dispatcher, and one SIP.\(^1\)

\(^1\)The dispatchers are not even necessary at the beginning.

Starting from this simple setup, the administrator of a new Pub/Sub system wants to scale it by adding machines (operators) so as to sustain a given number of connected publishers and subscribers. In some cases, this can be linked to QoS guarantees, for instance if subscribers pay for a minimum guaranteed throughput.

A. Interconnection Points Provisioning and Scalability

The first step is to provision the interconnection points to be able to scale up to the maximum number of expected users. When a new publisher wants to connect to the Pub/Sub system, the monitor searches for a PIP that has not reached its TCP limit nor its bandwidth cap. Once this has been found, the new publisher connects to it. Otherwise, a new PIP is started. Note that publishers with very large throughput requirements can be connected to more than one PIP. If the incoming publication throughput of a PIP increases to the point of saturation, a new PIP is also created, and the TCP connections of the saturated PIP are split with the new one.

When a publisher leaves the system, the monitor checks the usage of the PIPs (number of TCP connections and bandwidth). If possible, a PIP is removed and its TCP connections moved to other PIPs. As mentioned in the previous section, all the PIPs must be able to anycast the publications to any \(D_P\) dispatcher.

The subscription interconnection points are provisioned similarly. When a new subscriber wants to connect to the system, the monitor again looks for a SIP that can support the additional TCP connection and the additional bandwidth, and if none is available a new SIP is created. SIPs can also be created or removed as the incoming throughput coming from the matching operators varies.

B. Matchers and Dispatchers Provisioning and Scalability

We now describe how to scale the matching operators as the number of publications and subscriptions fluctuates.

Initialization: one matching operator.

At the beginning, only one matching operator \(M\) is activated and connected to the two dispatchers \(D_P\) and \(D_S\). As the number of subscriptions increases, \(M\) stores them into its database and its filtering performance decreases. Furthermore, as the publication throughput increases, \(M\) does the filtering load. At some point, \(M\) is no longer able to filter the incoming publications and partitioning is required.

First partitioning level: publication partitioning.

When matching operators using sub-linear filtering algorithms are saturated, it is usually better to reduce the number of messages processed per operator than to reduce the number of subscriptions stored on each operator (see Section V for more details). Thus, we use publication partitioning as much as possible.

Given certain thresholds, when a matching operator cannot filter the publications faster than they arrive and its memory is not full, then publication partitioning is initiated. The use of thresholds is important to allow enough time
for the deployment of the new operators. As the speed of the matching operators is not homogeneous, each operator receives a share of the incoming publications proportional to its computing power.

If the flow of incoming publications decreases, the system may become overused. Still given a threshold, the monitor compares the incoming throughput with the potential filtering speed of the matching operators and decides whether or not one of the operators can be removed.

**Second partitioning level: subscription partitioning.**

Publication partitioning alone cannot be done indefinitely, as even with complex filtering algorithms there exists a point after which the filtering performance stops to be sub-linear (again, see Section V for more details). Using a memory threshold (e.g. 75%), we thus monitor the memory of the matching operators and initiate subscription partitioning once one of them is saturated.

The subscriptions must be split so that all the partitioned matching operators have roughly the same filtering speed. For instance, if a naive linear filtering algorithm is used and all the operators have the same processing power, then the subscriptions can simply be split equally in random fashion. If more complex filtering algorithms exploiting relationships between subscriptions are used and if the set of subscriptions is split randomly, it is possible to obtain a bad split, forcing the next subscription partitioning to be done sooner. To overcome this, a clustering algorithm can be used for splitting subscriptions, and dynamic reconfiguration can be used to move subscriptions from the slowest matching operators to the fastest ones.

As subscriptions are canceled, the monitor checks if the memory of the matching operators is underused. If the remaining subscriptions can be stored on less machines (always given a threshold), then one of the operators is discarded and its subscription set is dispatched to the others.

Our final architecture, again shown in Figure 5, can scale with the number of publishers, the number of publications, the number of subscribers, the number of subscriptions, and the number of matched publications.

---

**V. EXPERIMENTAL RESULTS**

In this section, we study how the minimum number of required matching operators varies under different load scenarios, but first we discuss how the optimal partitioning depends on the filtering algorithm and the subscription set.

If a naive filtering algorithm testing each publication against all stored subscriptions is used, or more generally if the time required to filter a publication by a matching operator is linearly dependent on the number of stored subscriptions, then any partitioning will result in roughly the same number of required matching operators. However, subscription partitioning requires to duplicate the stream of publications and increases the communication load in the system. It is therefore preferable to store as many subscriptions as possible per matching operator and to partition the publications. Subscription partitioning is only done if absolutely necessary.

Interestingly, this is not always the case when more complex filtering algorithms are used, for instance algorithms exploiting containment relationships between subscriptions. Such algorithms are sublinear with a small number of subscriptions but can become increasingly inefficient as the subscription set increases. This occurs when the matching memory saturates due to the use of paging at the OS level, which forces the operator to swap. Adding the complexity of the filtering algorithm, this results in a significant performance degradation. For such systems, it can be preferable not to exploit the entire memory of the matching operators, even if this increases the number of cloned publications in the system.

For our simulations, we assume that our content-based Pub/Sub system has 1000 publishers, each of them transmitting 10 publications per second. The publication throughput is therefore $10^4$ publications per second. We also assume that the operators’ bandwidth is sufficient, thus we disregard the number of dispatchers and interconnection points, and only focus on the minimum number of required matching operators. We use the SIENA [2] content-based filtering algorithm. We generate subscription sets of various sizes and a collection of events using the workload generator provided by the authors of SIENA, and capture the average matching
speed through experiments on our own cluster. The results are shown in Figure 8.

Knowing the speed of the matching operators, Figure 9 presents the minimum required number of matching operators as a function of the number of subscriptions in the system. The four curves respectively use matching operators with subscription lists of size $10^2$, $10^3$, $10^4$, and $10^5$. Figure 10 shows the minimum number of matching operators as a function of the maximum number of stored publications per operator. The four curves use $10^3$, $10^4$, $10^5$, and $10^6$ subscriptions in the system. The main result is that for content-based Pub/Sub systems with non-naive filtering algorithms, designers should pay close attention to the number of stored subscription per matching operator. Using too few or too many subscriptions per operator can significantly increase the number of matching operators required to support the publication throughput.

VI. RELATED WORK

Content-based Pub/Sub systems have been widely studied by the research community in the past years. By design, they are conceived to support a large number of users. Discussing Pub/Sub systems in general is beyond the scope of this paper (see [1] for a survey). Instead, we focus on related work close to the objectives of our research, i.e., Pub/Sub infrastructures designed to dynamically scale with various parameters, and papers that study the behavior of such systems analytically.

Content-based Pub/Sub: SIENA [2] is one of the first content-based Pub/Sub frameworks. It uses a distributed network of dedicated event brokers and reverse path forwarding. Brokers assume all roles, serving as contact points for publishers and subscribers, matching events, and forwarding events to other brokers who may have at least one matching subscription. The matching operation is performed several times in order to determine the propagation of publications to brokers. The SIENA algorithms have become reference solutions for the problem of routing content-based events and subscriptions in an application-level network.

PADRES [3] is a Pub/Sub system with the capability to produce complex correlations among both past and future events. The system, using advertisement-based routing, includes algorithms to optimize the retrieval and correlation of heterogeneous data streams. It also exploits the relationships between subscriptions (equivalence, containment), supports multiple data partitioning schemes (each with its own trade-offs), and handles network failures. Recent work on PADRES, such as that by Cheung and Jacobsen [6], adds load-balancing to increase its scalability.

Scalable Pub/Sub architectures: The need for designing Pub/Sub infrastructures that can scale with large user and subscription populations has long been recognized. Classical broker-based systems such as SIENA [2] or PADRES [3] exploit similarities between subscriptions (in particular containment relationships) to improve the matching speed. Generally, such mechanisms produce higher gains when the considered subscriptions are similar or describe overlapping interests. By clustering the subscribers in a smart way, one can scale to large user populations with a reasonable number of brokers. These systems are designed to be deployed in large-area networks and do not explicitly consider deployments in clusters with a given scalability objective.

In the context of content-based routing of XML documents, Felber et al. [7] proposed several scalable strategies for parallel filtering that offer various trade-offs in terms of throughput, latency, resource consumption, and complexity. They can be combined together into a hierarchical routing architecture that can adapt to the specifics of individual services. These strategies explore similar trade-offs to those discussed in Section III, but at a much coarser level. There is no generalization of the model nor any analytical study.

Baldoni et al. [8] underlined the lack of self-organization capabilities of current content-based Pub/Sub systems, thereby requiring human intervention for set-up and management of the application-level network. Their solution describes a fully decentralized Pub/Sub system, where all nodes act as both publishers/subscribers and brokers. Self-configuration is achieved through the use of configuration middleware between the overlay supporting event propagation and the subscription layer that deals with subscription installation. The configuration middleware system decides on subscription-to-node and event-to-node mappings for better performance in this peer-to-peer context.

Baldoni et al. [9] also tackled the problem of event dissemination and proposed a novel approach by reorganizing the overlay topology. They presented a self-organizing overlay network based on the principle that brokers matching similar events should be located as close from each other as possible.

Farroukh et al. [10] also used event partitioning. They introduced a parallel matching engine on current chip multiprocessors to increase the throughput and reduce the matching time. Their engine exploits the characteristics of multicore computers to partition the events and perform the
processing.

Analytical frameworks for Pub/Sub: Raiciu et al. [11] studied the dimensioning and scaling of content-based Pub/Sub systems. They first analyzed a few applications suitable for content-based Pub/Sub and extracted their characteristics. They discovered that these applications exhibit significant diversity, from input loads (publishers bandwidth as well as publication frequency) to requirements (tolerable latency). They focused their study on throughput and latency, and analyzed their effects on the architecture parameters. However, they do make several unrealistic assumptions, such as using communication links with infinite bandwidth.

Wang et al. [12] studied the impact of subscriptions partitioning and existing routing algorithms on the throughput of Pub/Sub platforms. They mainly focused on the impact of subscription summaries on the routing algorithm rather than the impact of subscription partitioning.

In [13], Rose et al. proposed Cobra, a provisioning technique for aggregating blogs and RSS feeds. In a sense, it operates like a search engine, as one of its operators is a crawler which retrieves information. Its similarity with our work is its decoupled architecture between crawlers, filters, and reflectors as well as its off-line service provisioning technique to determine the minimal amount of physical resources required. However, while the general ideas are similar, their implementations are significantly different. The purpose of each stage is different, therefore leading to different analytics. Cobra’s filtering stage considers text matching algorithms, while we allow more general filtering algorithms and adapt our architecture accordingly.

Yoon et al. [14] also investigated provisioning techniques when SLAs need to be enforced for a Pub/Sub service. The provisioning is accompanied by broker topology transformations and only applies to reverse-path forwarding-based systems where each broker endorses all roles (contact point, matcher, dispatcher for other brokers).

VII. Conclusion

Content-based Pub/Sub architectures are becoming increasingly important for various reasons. Such architectures can easily be deployed on clusters or private clouds within companies, but the main challenge is to scale them with application-specific requirements: throughput, latency, number of publishers and subscribers, etc.

In this paper, we have proposed a flexible and generic stage-driven architecture for scaling a content-based Pub/Sub system combining dispatchers, interconnection points as well as publication and subscription partitioning. We have analyzed how to provision such an architecture in order to meet application-level constraints while minimizing the number of machines involved. Besides the framework itself, we have shown that when sub-linear filtering algorithms are used on large content-based Pub/Sub systems, developers must estimate the optimal size of the subscriptions lists as accurately as possible in order to minimize the total number of matching engines required. Our main goal for future work is to extend the architecture for the deployment of wide-area Pub/Sub systems over the Internet.

VIII. Acknowledgments

This work is partially sponsored by European Commission’s Seventh Framework Program (FP7) under grant agreement No. 257843. The authors would also like to thank the referees for their careful review of the manuscript.

References


