Subscription Partitioning and Routing in Content-based Publish/Subscribe Systems

Yi-Min Wang, Lili Qiu, Dimitris Achlioptas, Gautam Das, Paul Larson, and Helen J. Wang
Microsoft Research, Redmond, WA, USA

***Abstract*— *Content-based publish/subscribe systems allow subscribers to specify events of interest based on event contents, beyond pre-assigned event topics. When networks of servers are used to provide scalable content-based publish/subscribe services, we have the flexibility of partitioning existing subscriptions and routing new subscriptions among multiple servers to optimize various performance metrics including total network traffic, load balancing, and system throughput. We propose two approaches to subscription partitioning and routing, one based on partitioning the event space and the other based on partitioning the subscription set, and discuss their trade-offs. Finally, we collect and analyze a set of real-world stock-quote subscriptions and use that as the basis for our simulation study to demonstrate the effectiveness of the proposed schemes.*

I. Introduction

The World Wide Web use today predominantly follows the polling model: a Web user either visits a page using its URL or goes through a search engine to find pages of interest; if the information sought is not yet available, the user needs to periodically poll again in the future for that same information. As an example, consider a user who is interested to know when a new game console becomes available for a reasonable price at an online store or auction site. If the Web can additionally support a standard event notification model through a publish/subscribe (pub/sub) mechanism that stores users' subscriptions and notifies them when information of interest becomes available, it will greatly relieve burdens on the users and further enhance the Web's utility. Such event notification services, usually known as "alerts", have recently begun to emerge; Yahoo! Alerts and MSN Mobile are two examples. So far, the popularity of such services has not reached the point where scalability is a major concern.

We envision a future where the event notification model of Web access will parallel the polling model both in terms of usage and universality. Just as Content Distribution Networks (CDNs) have been deployed to provide scalable Web information dissemination, we propose building *Event Distribution Networks (EDNs)* to provide scalable event dissemination. In contrast with a centralized pub/sub system where all events and subscriptions are sent to a single server, an EDN will be built as a self-configuring overlay network [15] of servers, which provides content-based event and subscription routing to optimize various performance metrics such as total network traffic and overall system throughput. A geographically distributed subset of nodes, called edge servers, are deployed to provide a low-latency interface to geographically distributed subscribers and publishers. Other servers reside inside the network and may host subscriptions or route traffic or both. Any subscription submitted to an edge server may be hosted locally or forwarded to another server as needed. Any event published through an edge server will be routed to all server nodes that host at least one subscription matching the event.

In this paper, we consider the following EDN service model: an end user submits her subscriptions through a Web interface by specifying a device-independent ID as the notification address, for example, a Yahoo! ID or a .NET Passport ID. Every notification generated by an event match is first sent to a *notification routing service* that recognizes the address. The routing service resolves the address and forwards the notification to the user, based on her preference, via instant messaging, email, telephone, cell phone SMS, etc. [22], [26]. The EDN and the various notification routing services are often operated by different service providers. The routing services themselves can be scalable services based on dynamic request redirection mechanisms, and their internal network architectures are typically not exposed to the EDN.

This service model is quite different from the operational model implicitly assumed in most of the existing pub/sub systems [1], [7], [30]. In these systems, a subscriber typically submits a subscription to the "closest" server and essentially joins the event distribution tree as a leaf node: the notification address is an IP communication endpoint on the subscriber's machine, to which the server directly sends a notification when an event matches the subscription. In contrast, in the above EDN service model, the subscribers as well as the notification routing services are architecturally decoupled from the EDN. This gives the EDN the flexibility to freely move subscriptions around to optimize its internal network performance. In particular, we focus on two issues resulting from this flexibility in this paper: subscription partitioning and subscription routing.

An event service provider would typically start with a single-server, centralized architecture and then consider an upgrade to a multi-server, distributed architecture when the service is successful and the single server can no longer keep up with the demands. At this point, a large number of existing subscriptions are available and *subscription partitioning* refers to the offline process of partitioning these subscriptions onto the multiple servers. Once the existing subscriptions are partitioned and the servers start providing service, *subscription routing* refers to the process of forwarding each incoming new subscription to a server; it is essentially an online, incremental version of the subscription partitioning process.

In this paper, we investigate two dual approaches to subscription partitioning and routing: *Event Space Partitioning (ESP)* and *Filter Set Partitioning (FSP)*. The former approach partitions the event space among the servers and replicates subscriptions if necessary, and the latter approach partitions the set of subscription filters among the servers and may need to forward some events to multiple servers. We show that the ESP approach is well-suited for applications that perform content-based routing for equality predicates. We use simulation results based on actual stock-quote subscription data to demonstrate that our scheme can achieve good systems and network performance in terms of total network traffic, load balancing, and system throughput. We also briefly discuss the traffic-reduction effectiveness of applying the FSP approach to range predicates.

II. Network Architecture

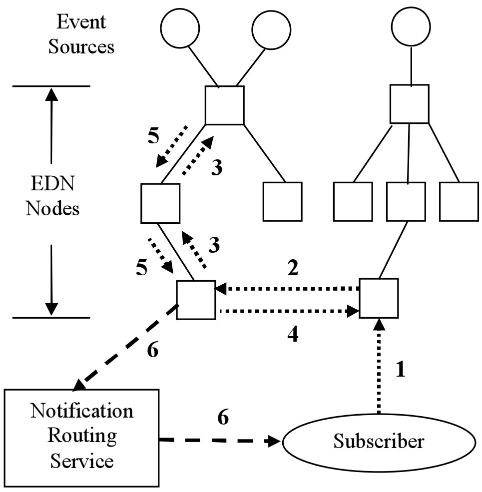
The following terminology will be used throughout the paper: an event source publishes events and a subscriber submits subscriptions for events of interest. A filter f is any expression that defines a set of events E_f, and an event e matches the filter f if $e \in E_f$. A filter f' is said to cover another filter f if $E_f \subseteq E_{f'}$. A similar covering relationship is defined between two sets of filters. A subscription consists of a notification address and a subscription filter that conforms to a pre-defined schema. When a server receives a published event that matches a subscription filter, it sends a notification through a routing service to the corresponding notification address.

To allow the event notification model of Web access to be as universally useful as the current browsing/polling model, the EDN must be designed to accommodate a large spectrum of pub/sub models, including (1) channel-based pub/sub, where events and subscriptions are identified by flat channel names or IDs; (2) topic-based pub/sub, where events and subscriptions are specified with topic names from a hierarchical topic namespace; (3) attribute-based pub/sub (commonly referred to as content-based pub/sub [1], [7]), where events are identified by a set of attribute-value pairs in a well-defined metadata block and subscriptions specify equality/range/prefix predicates on a subset of the attributes; (4) keyword-based pub/sub, where events are documents and each subscription specifies a list of keywords to be matched against the entire document contents; (5) similarity-based pub/sub, where each event is a document associated with a feature vector, each subscription is a query with a feature vector, and the matching is based on a threshold on the similarity between the two vectors [29]; and (6) pattern-based pub/sub, where subscriptions can specify predicates on sequences of events [13], [7].

To support scalable operations under diverse pub/sub models, EDN provides an extensible content-based routing mechanism to achieve efficient and effective event dissemination. Figure 1 illustrates the EDN network architecture. For ease of presentation, we assume in the following discussions that the servers are statically configured into a tree rooted at an edge server directly connected to event sources. The same concept can be applied to systems that dynamically construct trees through either "advertisement forwarding" [7] or "SUBSCRIBE message forwarding" [23] in a peer-to-peer routing system.

Instead of distributing events to all servers, which then perform local subscription matching operations, EDN allows a *content-based router* at each server to construct *content-based route updates* and propagate them upstream to affect event routing, as shown in Step 3 of Figure 1. Essentially, such updates indicate to the upstream server the subset of events that need to reach the subtree rooted at current node. The updates can be based on a dynamic partitioning of the event space or *summary filters* that cover all currently hosted subscription filters. Optionally, each server can periodically exchange route updates with other peers, as shown in Step 4. This would allow a new subscription submitted to any of the servers to be routed to the server whose content-based routes most closely "match" (and hopefully already cover) the new one, thus minimizing the additional event traffic handled by each server.

Two key points distinguish EDN content-based routing from existing systems in which each node submits some subscriptions to its parent and draws from the parent those



1. Submit subscription
2. Subscription routing
3. Content-based route updates
4. Peer exchange of route updates
5. Content-based event routing
6. Notification delivery

Fig. 1. EDN network architecture

events that match any of its subscriptions [7], [25]. First, since the EDN content-based router is architecturally decoupled from the main pub/sub filtering engine, the format of route updates need not be restricted to that of subscription filters. This allows the construction of compact route updates for efficient routing. Bloom filters [10], as discussed later, are such an example. Second, the EDN route updates need not precisely capture the set of required events. This allows a practical trade-off between wasted event traffic and routing efficiency, which can be made to satisfy different performance metrics such as total network traffic and overall system throughput. In particular, the use of "imprecise route updates" includes the following two approaches as two extremes of the spectrum: at one end, the route updates are so imprecise that every event is delivered to every server, but the routing overhead is minimum; at the other end, the route updates are so precise that none of the servers ever receives any unnecessary event traffic [7], [25], although this usually comes at the expense of more heavy-weight processing for routing.

III. Subscription Partitioning

As a first step towards understanding subscription partitioning and routing issues, we restrict our attention to a flat, dispatcher-based model: all subscription servers are connected directly to a dispatcher, which receives events from a source and forwards each event to a selective subset of servers. In this paper, we focus on the attribute-based (i.e., content-based) pub/sub model. Let d_t be the total number of attributes. Every event e must specify a value for each of the d_t attributes, and can therefore be represented as a "point" in the d_t-dimensional space, in which each dimension corresponds to an attribute. Every subscription filter f is a conjunction of d_f predicates ($d_f \leq d_t$), each specifying a constraint on a different attribute.

We consider two types of predicates: *equality predicates* on unordered, discrete value domains with constraints like "StockSymbol=GE", and *range predicates* on ordered value domains with constraints like "35 $\leq$ Stock-Price $\leq$ 40". Conceptually, we can impose an ordering on every unordered value domain and view each subscription filter as a "rectangle" [21] in the d_t-dimensional space, except that the rectangle projects into a single point along any dimension associated with an equality predicate. However, since considering ranges in an artificially ordered value domain may not be the most effective approach for constructing summaries, our solutions give separate treatments to equality and range predicates and do not attempt to combine them.

For routing efficiency, it is often desirable to make routing decisions based on only a subset of the d_t attributes. In the remainder of this paper, we assume that subscription partitioning, summary construction, and event/subscription routing are all based on a given set of d attributes, $d \leq d_t$. In practice, the set of d attributes to use is either obvious (as in the case of stock-quote subscriptions) or can be determined by entropy-based analysis for identifying most distinguishing attributes.

In general, there are two approaches to partitioning the overall pub/sub operations among multiple servers: we either partition the event space or partition the set of subscription filters. In the *Event Space Partitioning (ESP)* approach, given the number of servers N_s, the d-dimensional space is partitioned into N_s disjoint subspaces, each assigned to a different server. A subscription is hosted by a server if its filter (i.e., rectangle) intersects the server's associated subspace. An event is forwarded only to the server whose subspace contains the point representing the event. Summary filters can be used to further eliminate event traffic belonging to the part of a subspace that does not currently intersect any subscription filters. The main advantage of the ESP approach is that it minimizes event traffic by forwarding each event to at most one server. The main disadvantage is that, if a subscription filter intersects multiple subspaces, the subscription needs to be replicated on multiple servers, thus increasing the total number of subscriptions hosted by the network, as well as complicating the task of subscription management for subscription deletion or subscription state updates.

Filter Set Partitioning (FSP) is a dual partitioning approach which always assigns each subscription to a single server. Similar subscriptions, i.e., subscriptions likely to be matched by the same events, are grouped together and assigned to the same server to allow for the construction of compact and effective summary filters. Summary filters from multiple servers may overlap and an event is forwarded to every server whose summary filter contains the event. Figure 2 illustrates the difference between the two approaches for 2-dimensional subscription filters.

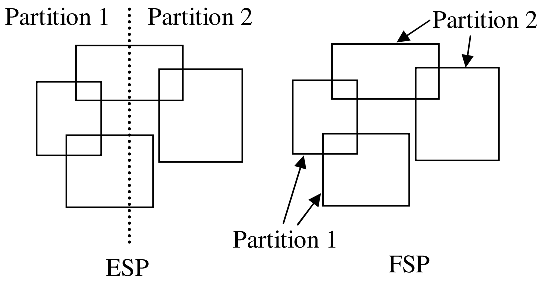


Fig. 2. (a) Event Space Partitioning and (b) Filter Set Partitioning.

A. Event Space Partitioning for Equality Predicates

We first consider equality predicates. For each event and subscription, we use the concatenation of the d predicates as its signature string. We hash the string to obtain a more uniform distribution in the hash domain, which is then used as the event space. We chose the ESP approach because each subscription is represented as a point in the hash domain and does not intersect multiple subspaces. So we gain the advantage of sending each event to at most one server without the drawback of managing replicated subscriptions.

To achieve a more fine-grain load balancing, we overpartition the hash domain into $N_s \cdot R$ buckets, where R is the *overpartitioning ratio* [16]. (Recall that N_s is the number of servers.) The dispatcher maintains an *indirection table* that dynamically maps each bucket to a server for event routing. It also makes the table available to all the servers for subscription routing. There are several scenarios where run-time re-partitioning is desirable. For example, actual run-time server load may deviate significantly from the estimated values; new subscriptions with unanticipated distributions may create load imbalance; new servers may join and existing servers may leave. By maintaining the indirection table, the system can support dynamic load balancing by adjusting some of the table mapping entries at run time and migrating the corresponding subscriptions.

Given a large set of existing subscriptions, we hash all their signature strings into appropriate buckets and assign to each bucket a weight that is the product of the number of unique strings and the total number of subscriptions falling into that bucket. The basic idea is that the first term gives an estimate of the aggregated event rate under the assumption of uniform event distribution, and the second term corresponds to the per-event processing overhead (for example, the overhead of preparing and sending a notification for each matching subscription).

The problem of the optimal load-balancing assignment of buckets to servers (i.e., minimizing the maximum load among all servers) is NP-complete. We use a simple greedy algorithm from the existing literature [12]: we sort the buckets in the order of decreasing weights, and at each step assign the next bucket in the sequence to the server that currently has the minimum load. It has been shown that this greedy algorithm has a good performance bound of within $(2 - \frac{1}{N_s})$ factor of the optimum [12]. We also use the same algorithm to perform dynamic load balancing by redistributing buckets owned by the most heavily loaded server to other less loaded ones.

To further reduce event traffic, the content-based router at each server can optionally maintain a Bloom filter [3], [17], [10] for every bucket it owns and report them to the dispatcher to block most of the events with unmatched signature strings. A Bloom filter is a bit vector V with length m, which represents a set E of n elements to support membership queries of the form: *"does the given element e belong to the set E?"* The key idea is to choose k independent hash functions, $h_1, h_2, ..., h_k$, each with range from 1 to m. The bit vector V is initialized to all 0's and selective bits are set to 1 as follows: for each $e_i \in E$, $1 \leq i \leq n$, the bits at positions $h_1(e_i), h_2(e_i), ..., h_k(e_i)$ are set to 1. Given an element e, we check the bits at positions $h_1(e), h_2(e), ..., h_k(e)$ and conclude that $e \notin E$ if any of them is 0; otherwise, we conjecture that $e \in E$ with some false-positive probability. Analyses [3], [10] have shown that this probability is minimized for $k = ln(2) \cdot m/n$ and, under this optimal k, the false-positive probability drops exponentially as m increases.

B. Simulation study with stock-quote subscription data

To evaluate the above scheme, we obtained a snapshot of actual end-user subscription filters from a major stock-quote alert service provider. (To our knowledge, this is the first study based on real data in the content-based pub/sub literature.) The data contains approximately 1.48 million subscriptions with 0.29 million unique filters involving 21,741 stock symbols. In Figure 3, we plot the number of subscriptions for stock symbols versus their popularity ranking on a log-log scale. As we can see, the middle part of the curve fits quite well with a straight line with slope -1.07, which suggests that the number of subscriptions for a stock symbol is proportional to $\frac{1}{i^\alpha}$ (i.e., Zipf-like distribution). However, the head and the tail of the curve deviate from the straight line. The deviation at the head is quite common, as reported by several Web document analyses, such as [4], [20]. A possible explanation for the deviation at the tail is that the number of unique symbols is limited, whereas Zipf-like distributions tend to exhibit in a relatively large data set.

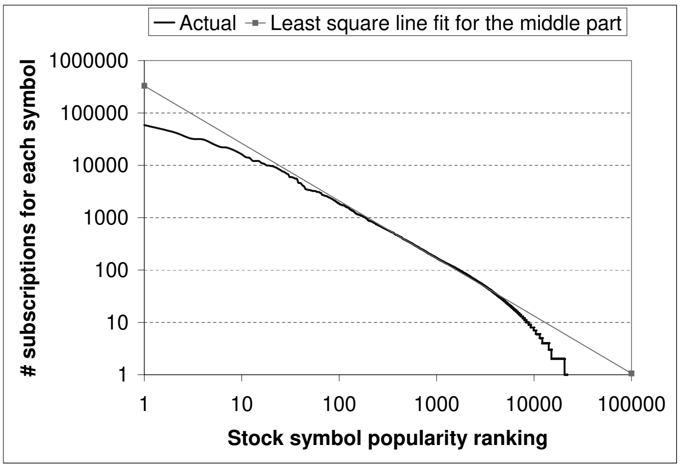


Fig. 3. Zipf-like distribution for number of subscriptions of stock symbols.

From the various sources on the Web, we obtained a list of 43,734 stock-related symbols and treat the additional 43,734-21,741=21,993 symbols as less popular ones that may appear in future subscriptions. In our simulation, events corresponding to these additional symbols would be generated, but they might not be forwarded to any server until new matching subscriptions entered the system. To generate new subscription sequences that cover all 43,734 symbols, we "scaled up" the Zipf-like distribution in Figure 3 and used the new distribution as the basis for generating up to 100 million new subscriptions in our experiments. We also introduced permutation and perturbation on the distribution to study the performance of our algorithms in the presence of imperfect knowledge. These experiments were intended to simulate the changes in relative popularity among the stocks as business and market situation changes.

In traditional systems without content-based partitioning, each subscription would either be hosted by the server that received it, or be forwarded to some other server strictly for load-balancing purposes without regard to the content. In the worst case, every event needs to be sent to every server. With Event Space Partitioning, each event only needs to be forwarded to at most one server, so it is clearly that it can significantly reduce event traffic. In the following discussion, we will focus on the load balancing aspect of the partitioning algorithm and the use of Bloom filters to further reduce event traffic and possibly increase system throughput.

Load balancing: We first discuss the offline partitioning of the existing 1.48 million subscriptions. Figure 4 illustrates the load-balancing performance of the offline greedy algorithm by showing the load imbalance – defined as the maximum load divided by the minimum load among all servers – as a function of the overpartitioning ratio R with $N_s = 50$. As R increases, the performance first improves significantly but then saturates around $R = 10$. Additional experiments showed that $R = 10$ can achieve good load-balancing performance by bounding the imbalance within 1.20 across a wide range of N_s. For the remainder of this subsection, we use $R = 10$ and $N_s = 50$ for all the experiments unless otherwise noted.

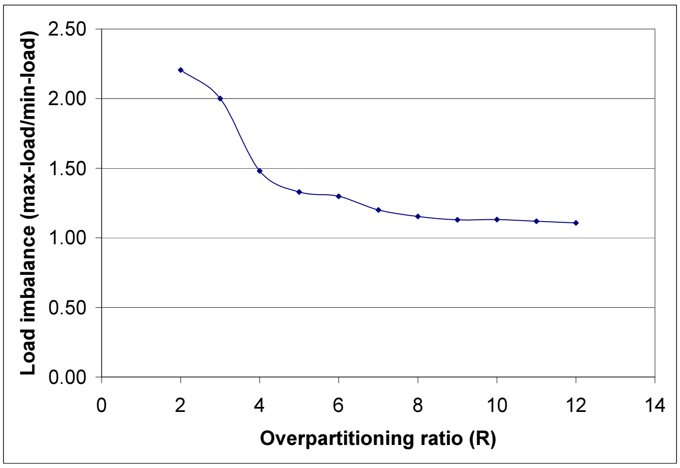


Fig. 4. Load-balancing performance of the greedy algorithm as a function of the overpartitioning ratio.

We next discuss the online routing of the 100 million simulated, new subscriptions. In our experiment, we first partition the existing subscriptions among the 50 servers, and then we generate new subscriptions, which follow various distributions. Figure 5 shows the potential load imbalance caused by new subscriptions. The "Actual" and "Anticipated" curves correspond to the cases where the new subscriptions follow the original (i.e., 21,741-symbol) and scaled-up (i.e., 43,734-symbol) Zipf-like distributions, respectively. Not surprisingly, the "Actual" curve stays flat because the new subscriptions proportionally increase the loads across all servers; the "Anticipated" curve stays within a small imbalance of 1.41 because the numbers of subscriptions of those less popular new symbols are dominated by those of the existing popular ones.

We also studied the effect of distributions that deviate from the anticipated distribution. For the "Perturb100" curve, we started with the "Anticipated" distribution and perturbed the value (i.e., number of subscriptions) of each symbol by up to $x\%$ in either direction where $x = 100$; that increased the imbalance to 2.83. For the "Per-

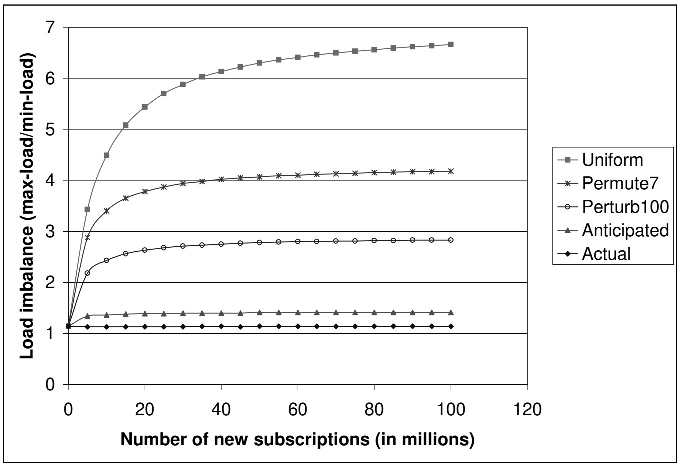


Fig. 5. Load imbalance as a function of new subscriptions.

mute7" curve, we divided the sorted set of symbols into y chunks where $y = 7$ and randomly permuted the symbols within each chunk; that increased the imbalance to 4.18. The "Uniform" curve corresponds to the case where the new incoming subscriptions follow a uniform distribution, which is unlikely to occur in practice but is included as a test case for our online repartitioning algorithm. The load imbalance increases to 6.66 in this case.

Figure 5 demonstrated that the load imbalance can be significant if the distribution of new subscriptions deviates significantly from that of existing subscriptions due to, for example, changes in relative popularity of certain stocks. Further experiments showed that the simple online repartitioning algorithm works well: even for the "Uniform" case and with a low imbalance threshold of 2.0 for triggering repartitioning, the algorithm was invoked only five times during the entire course of adding 100 million new subscriptions; each time it involved reassigning only three buckets from the maximum-load server to three other servers and the number of migrated subscriptions was around 0.7% of the total number of subscriptions at repartitioning time.

Bloom-filter summary: We next evaluate the performance of Bloom filters as a summary mechanism for traffic reduction. With R = 10 and N_s = 50, the average processing time per event at the dispatcher is approximately 3.6 microseconds on a dual-proc 2.2GHz Pentium machine with 1GB memory. The false-positive rate across all servers was less than 1%. Since the true positive rate with a precise summary is 21,741 / 43,734 = 49.7%, the use of Bloom filters reduced the event traffic by more than 100%-(49.7%+1%) = 49.3% at the end of the offline partitioning algorithm, compared to the case without Bloom filters. The memory usages for Bloom filters at the server and the dispatcher are 4.3KB and 215KB, respectively.

Note that we could have instead used a per-bucket symbol dictionary as a precise summary to eliminate all false-positive event traffic. Since the number of symbols per bucket is small in this case and so the dictionary lookup time is unlikely to become the bottleneck with respect to the dispatcher's outgoing link, minimum event traffic can be achieved without sacrificing system throughput. This may not be the case as the number of per-bucket symbols increases. For example, we have measured the dictionary lookup time involving 100,000 random symbols and 55% true positive rate to be 653 microseconds; the corresponding Bloom filter testing time ranged from 3.1 to 4.7 microseconds, with an extremely low false-positive rate. Assuming a 100Mbps link and an average message size of 1KB, the dispatcher's outgoing link is capable of consuming one message approximately every 100 microseconds. By using the dictionary, the dispatcher's CPU becomes the bottleneck and the system throughout is calculated as $1/(653{*}10^{-6}) = 1,531$ messages per second. In contrast, by using the Bloom filters, the link is the bottleneck and the throughput becomes $1/(0.55 * 100 * 10^{-6}) = 18,181$ messages per second, which is more than an order of magnitude higher.

C. Filter Set Partitioning for Range Predicates

In the case of range predicates, every event is a point and every subscription is a rectangle. We choose the Filter Set Partitioning approach to simplify subscription management. We group together rectangles that are close to each other in the event space by using an offline bulk-loading R-tree algorithm [11]. It constructs an R-tree [2] with all subscription rectangles residing at leaf nodes. We force the number of children of the root node to be equal to the number of servers, and assign each of the sub-trees at the root node to one server. Each server picks a level of its sub-tree, and uses the bounding rectangles residing in the nodes at that level as the summary filter. The content-based router at the dispatcher collects such bounding rectangles from all servers and builds another R-tree for routing events. (Alternatively, the dispatcher can use a separate R-tree for each server.) Using more bounding rectangles per server allows more precise summaries with lower false-positive rates, but incurs higher load on the dispatcher. So the maximum number of bounding rectangles per server is often dictated by the throughput requirement at the dispatcher. Each server also periodically exchanges bounding rectangles with other servers and maintains an R-tree for similarity-based routing of incoming subscriptions.

Our experimental results showed that offline R-tree-based partitioning can cut down the amount of event traffic by 20% to 60%, compared to random partitioning.

Moreover, R-trees of bounding rectangles are an efficient content-based routing mechanism: with $N_s = 50$ and 10 bounding rectangles per server, the average processing time per event at the dispatcher is approximately 3.0 microseconds.

IV. Related Work

A large number of prototype pub/sub systems have appeared in the literature, including Echo [9], Elvin [24], Gryphon [1], Herald [5], Hierarchical Proxy Architecture [30], Information Bus [18], JEDI [8], Keryx [28], Ready [13], SCRIBE [23], and SIENA [7], [6]. The EDN content-based routing was inspired by the quench expressions in Elvin, the hybrid matching schemes in Ready, the vector annotation in Gryphon, the filters poset in SIENA, and the subscription merging in Hierarchical Proxy Architecture. Similar ideas have also appeared outside the pub/sub literature. For example, each sensor node in the Directed Diffusion work [14] establishes sufficient "gradients" with its neighbors in order to draw events at required rates; each router in the XML router work [25] forwards to its parent the disjunction of its "link queries" received from the child nodes. None of the above related work, however, has explored the additional degree of freedom to partition and route subscriptions, and studied the performance benefits.

There have been several recent papers on using similarity-based clustering in the pub/sub setting. But all of them are concerned with reducing the total number of required IP multicast addresses for notification delivery, while the EDN subscription partitioning algorithms aim at enabling compact summaries for event traffic reduction, before notifications are generated. The Group Approximation Algorithm described in [19] tries to combine actual multicast groups into approximate groups, while introducing the least amount of false-positive traffic. For multi-party applications, [27] proposed using the k-means method to group subscribers with similar sets of publishers that they are interested in, so as to minimize overall wasted event traffic. The grid-based clustering framework in [21] partitioned the event space into cells, and associated a feature vector with each cell to indicate the set of subscribers interested in events falling into that cell. The cells are then clustered to minimize the expected waste of event traffic.

V. Summary

We have described the motivation for building Event Distribution Networks (EDNs) to provide scalable event dissemination. At the core of EDN is a content-based routing mechanism that allows both events and subscriptions to be routed based on their contents to optimize various systems and network performance metrics. We have proposed two subscription partitioning and routing approaches, Event Space Parititoning (ESP) and Filter Set Partitioning (FSP), and discussed their relative advantages. We have argued that the ESP approach eliminates the need to forward any event to more than one server and is well-suited for content-based routing with equality predicates. Through simulation study based on actual stock-quote subscription data, we have demonstrated that an overpartitioning approach could achieve good load balancing both statically and dynamically, while significantly reducing network traffic, and using Bloom filters as an additional summary mechanism could further reduce event traffic with little performance overhead. We also briefly discussed the FSP approach to partitioning subscriptions based on range predicates, and used preliminary experimental results to demonstrate that R-trees of bounding rectangles could be an efficient and effective summary mechanism.

References

[1] G. Banavar, T. D. Chandra, B. Mukherjee, J. Nagarajarao, R. E. Strom, and D. C. Sturman. An efficient multicast protocol for content-based publish-subscribe systems. In *International Conference on Distributed Computing Systems*, pages 262–272, 1999.

[2] N. Beckmann, H. Kriegel, R. Schneider, and B. Seeger. The R*-tree: An efficient and robust access method for points and rectangles. In *Proc. of the ACM SIGMOD Intl. Conf. on Management of Data*, May 1990.

[3] B. H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426, 1970.

[4] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker. Web caching and Zipf-like distributions: Evidence and implications. In *Proc. of INFOCOM*, 1999.

[5] L. Cabrera, M. Jones, and M. Theimer. Herald: Achieving a global event notification service. In *Proc. of HotOS VIII*, May 2001.

[6] A. Carzaniga, J. Deng, and A. L. Wolf. Fast forwarding for content-based networking. In *Technical Report CU-CS-922-01, Department of Computer Science, University of Colorado*, November 2001.

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

[8] G. Cugola, E. D. Nitto, and A. Fuggetta. Exploiting an event-based infrastructure to develop complex distributed systems. In *International Conference on Software Engineering*, pages 261–270, 1998.

[9] G. Eisenhauer, F. Bustamante, and K. Schwan. Event services for high performance computing. In *Proc. of Ninth High Performance Distributed Computing (HPDC-9)*, August 2000.

[10] L. Fan, P. Cao, J. Almeida, and A. Z. Broder. Summary cache: A scalable wide-area web cache sharing protocol. *IEEE/ACM Transactions on Networking*, 8(3):281–293, 2000.

[11] Y.J. Garcia, M.A. Lopez, and S.T. Leutenegger. A greedy algorithm for bulk loading R-trees. In *Univ. of Denver Computer Science Tech. Report #97-02*, 1997.

[12] R. L. Graham. Bounds for certain multiprocessing anomalies. *Bell System Technical Journal*, 45:1563–1581, 1966.

[13] R. Gruber, B. Krishnamurthy, and E. Panagos. The architecture of the READY event notification service. In *Proc. of the 19th IEEE International Conference on Distributed Computing Systems Middleware Workshop*, 1999.

[14] C. Intanagonwiwat, R. Govindan, and D. Estrin. Directed diffusion: A scalable and robust communication paradigm for sensor networks. In *Proc. of Mobile Computing and Networking*, pages 56–67, 2000.

[15] J. Jannotti, D. Gifford, K. Johnson, M. Kaashoek, and W. James. Overcast: Reliable multicasting with an overlay network. In *Proc. of OSDI*, October 2000.

[16] Hui Li and Kenneth C. Sevcik. Parallel sorting by over partitioning. In *ACM Symposium on Parallel Algorithms and Architectures*, pages 46–56, 1994.

[17] J. Marais and K. Bharat. Supporting cooperative and personal surfing with a desktop assistant. In *Proc. of ACM Annual Symposium on User Interface Software and Technology (UIST)*, October 1997.

[18] B. Oki, M. Pfluegl, A. Siegel, and D. Skeen. The Information Bus — An architecture for extensible distributed systems. In *Proc. of ACM SOSP*, 1993.

[19] L. Opyrchal, M. Astley, J. S. Auerbach, G. Banavar, R. E. Strom, and D. C. Sturman. Exploiting IP multicast in content-based publish-subscribe systems. In *Proc. of Middleware*, pages 185–207, 2000.

[20] V. N. Padmanabhan and L. Qiu. The content and access dynamics of a busy web site: Findings and implications. In *Proc. of ACM SIGCOMM*, 2000.

[21] A. Riabov, Z. Liu, J. L. Wolf, P. S. Yu, and L. Zhang. Clustering algorithms for content-based publication-subscription systems. In *Proc. of ICDCS*, 2002.

[22] M. Roussopoulos, P. Maniatis, E. Swierk, and et al. Personal-level routing in the Mobile People Architecture. In *USENIX Symp. on Internet Technologies and Systems*, 1999.

[23] A. I. T. Rowstron, A. M. Kermarrec, M. Castro, and P. Druschel. SCRIBE: The design of a large-scale event notification infrastructure. In *International Workshop on Networked Group Communication (NGC)*, pages 30–43, 2001.

[24] B. Segall and D. Arnold. Elvin has left the building: A publish /subscribe notification service with quenching. In *Proc. of AUUG*, 1997.

[25] Alex C. Snoeren, Kenneth Conley, and David K. Gifford. Mesh-based content routing using xml. In *ACM Symposium on Operating System Principles*, October 2001.

[26] Y. M. Wang, P. Bahl, and W. Russell. The SIMBA user alert service architecture for dependable alert delivery. In *IEEE Int. Conf. on Dependable Systems and Networks (DSN)*, 2001.

[27] T. Wong, R. H. Katz, and S. McCanne. An evaluation on using preference clustering in large-scale multicast applications. In *Proc. of INFOCOM*, pages 451–460, 2000.

[28] M. Wray and R. Hawkes. Distributed virtual environments and VRML: An event-based architecture. In *Proc. of the 7 th International WWW Conference*, 1998.

[29] T. W. Yan and H. Garcia-Molina. The SIFT information dissemination system. *ACM Transactions on Database Systems*, 24(4):529–565, 1999.

[30] H. Yu, D. Estrin, and R. Govindan. A hierarchical proxy architecture for internet-scale event services. In *Proc. of WETICE*, 1999.