On the Evaluation of Symmetric Publish/Subscribe

Anthony Tomasic
Carnegie Mellon University
Institute for Software
Research International
Pittsburgh, PA 15213, USA
tomasic@cs.cmu.edu

Charles Garrod
Carnegie Mellon University
Computer Science
Department
Pittsburgh, PA 15213, USA
charlie@cs.cmu.edu

Kris Popendorf
Carnegie Mellon University
Computer Science
Department
Pittsburgh, PA 15213, USA
krisp@cs.cmu.edu

ABSTRACT

Traditional publish / subscribe systems offer a range of expressive subscription languages for constraints, but restrict the publish operation to be a single published object that contains only constants and no constraints. We have introduced a novel generalization of publish / subscribe called symmetric publish / subscribe where both publications and subscriptions contain constraints in addition to constants, and published objects are matched to subscriptions by computing the intersection of their constraints. This paper describes the challenge of empirically evaluating symmetric publish / subscribe, a system for which traditional publish / subscribe benchmarking and evaluation tools do not completely apply. We introduce a number of workloads designed to elucidate the performance issues in our implementation, discuss the advantages and disadvantages of our evaluation methodology, and discuss how our experience generalizes to the evaluation of other novel publish / subscribe and database systems for which no established evaluation methodology exists.

1. INTRODUCTION

Current publish / subscribe systems support two key operations. The subscribe operation allows a client to register a subscription that contains a constraint. The publish operation allows a client to send a message to all clients whose constraints match a published object. Constraint languages for the subscribe operation include atomic matching (many are listed in [13]), comparison predicates over sets of attribute/value pairs [25], XPath expressions over XML documents [1], and vector space matches on documents [26]. In all of these systems, clients may only publish objects with fixed constants.

Symmetric publish / subscribe allows both publications and subscriptions to consist of constraints. Matches between publications and subscriptions are determined by computing the intersection of publication constraints with subscription constraints. This generalization leads to higher expressive power than classical publish / subscribe systems where publications consists only of constants, not constraints. The system is analogous to constraint databases [3, 4, 5, 20] which provide query processing over constraints.

To better understand symmetric publish / subscribe, consider an auction system. In this application domain of symmetric publish / subscribe, a seller of merchandise typically offers a range of options to buyers. In particular, pricing discounts often vary depending on the size of an order (lot size). Buyers are interested in simultaneously expressing price upper bounds and lot size ranges.

In a typical classical publish / subscribe system such as the Java Messaging System (JMS), an example auction implementation might assign sellers as publishers and buyers as subscribers. The seller publishes a lot size and price on a topic (channel) that represents a product category. The publication is a set of attribute/value pairs. For example, the publication "$topic$ = 'pencils' AND lot_lower = 1000 AND lot_upper = 10000 AND $price$ = 1.00" represents an offer to sell a lot of pencils.

Symmetric publish / subscribe enables the seller to express such an offer directly as a publication constraint and matches buyers' and sellers' constraints directly. In this case the additional expressive power of symmetric / publish subscribe allows the seller to describe the offer more naturally as "$topic$ = 'pencils' AND $1000 \leq lot \leq 10000$ AND $price = 1.00$."

Our previous work [23] is the first known investigation of symmetric publish / subscribe systems. That paper presented a preliminary investigation by reporting analytical and experimental results on several basic questions, e.g., the impact of the complexity of the constraint language, the performance penalty of symmetric publish / subscribe verses classical publish / subscribe, etc. In this paper, we describe the system of [23] and discuss the evaluation methodology used in that paper.

Section 2 describes symmetric publish / subscribe detail and Section 3 describes the design of of the system in more detail. These two sections are mostly reproduced from [23] as a convenience to the reader. Section 4 discusses the analytical and experimental results of [23] and discusses the advantages and limitations of those results. Section 5 evaluates the experimental evaluation according to the criteria described in [19, Chapter 2]. Section 6 surveys related work and Section 7 concludes the paper with a discussion of future benchmark work.

Proceedings of EXPDB 2006, June 30, 2006, Chicago, Illinois, USA

2. SYMMETRIC PUBLISH / SUBSCRIBE

A symmetric publish / subscribe system consists of:

- A *schema*of a set of attributes $A = \{a_1 \dots a_{|A|}\}$.
- A set of comparison operators, $O = \{o_1 \dots o_{|O|}\}$.
- A set of types $T = \{t_1 \dots t_{|T|}\}$ and an associated domain of values $D_{t_i} = \{d_1 \dots d_{|D|}\}$ for each type t_i.
- An assignment of a type to each attribute.
- A set of clients K.
- A set of constraints $C = \{c_1 \dots c_{|C|}\}$. Each constraint is a pair (k, e) of a client $k \in K$ and a boolean expression e. e is composed of the conjunction, disjunction, and negation of predicates $p \in P$, where P is set of type-safe predicates of the form $a_i \theta_i v_i$ where $a_i \in A$, $\theta_i \in O$, and $v_i \in D$. Each attribute must appear at most once in each disjunctive phrase within a constraint.
- A set of four functions:

  void* publish(c, k):** $C \times K \rightarrow$ ***void is a client function that publishes the constraint c of client k.

  handle* subscribe(c, k):** $C \times K \rightarrow$ ***handle is a client function that adds the subscription c for client k and returns a handle to manage notifications.

  ***pairset* match(C_1, C_2):** $C \times C \rightarrow C \times C$ is a server function that computes the match of constraint sets (publishers) C_1 and (subscribers) C_2. A match is a subset of $C \times C$.

  message* notify($pairset$):** $K \times K \rightarrow$ ***message is a server function that notifies pairs of clients (k_1, k_2) as determined by the match function.

The schema defines an underlying data model of attribute-value pairs. Each attribute has a single type of integer, float, date, string, point, region, etc. For example, {(x, integer), (y, float), (thing, string)} is a schema (that is, an instance of the data model) with three attributes and three different types. Publisher and subscriber constraints are type-checked against the corresponding type.

Publisher and subscriber constraints are conjunctions of comparison predicates where an attribute is compared to a constant. For example, a subscriber constraint may be "$x > 5$ AND $y < 5.5$ AND $thing =$ 'squirrel'". This constraint is also a legal publisher constraint. In fact, any constraint can be either a publisher or subscriber constraint; However, the interpretation of the constraint depends on its role as a published constraint or a subscription, as described at the end of this section.

In addition, each constraint may reference a variable only once per disjunctive clause. Thus, redundant constraints such as "$x < 1$ AND $x \leq 3$", tautologically false constraints such as "$x < 1$ AND $x \geq 3$", and complex constraints such as "$x < 1$ AND x $\neq 0$" are disallowed. Range expressions such as "$x > 1$ AND x ≤ 3" are explicitly supported with the range operators for each type. For example, for integers there are four range operators covering the four cases of open or closed intervals: $(\cdot, \cdot)$, $(\cdot, \cdot]$, $[\cdot, \cdot)$, and $[\cdot, \cdot]$.

Constraints describe *point-sets* [20], the (possibly infinite) set of points that satisfy the constraint. Thus, the constraint $0 \leq x \leq 1$ AND $0 \leq y \leq 1$, where x and y are floats, describes the set of all points of a unit square anchored at the origin. No type conversion is allowed; every value must be of the type of the attribute in a given predicate.

The match operation determines the set of publisher constraints that intersect subscriber constraints. A publisher constraint intersects a subscriber constraint if the point-set of a publication constraint intersects the projections of the point-set of the subscriber constraint. The constraint c_1 defined as "$x = 1$" contains the single value (1). The constraint c_2 defined as "$x = 1$ AND $y = 1$" has the point set containing the single point (1,1). The projection of c_2 onto c_1 is (1). The projection of c_1 onto c_2 is undefined. If c_1 is a publisher constraint and c_2 is a subscriber constraint, then the constraints match because the publisher point-set of (1) intersects projection subscriber point-set of (1,1). However, the opposite does not hold. If c_2 is a publisher constraint, then subscription c_1 does not match. Another consequence of the definition of matching is the treatment of half spaces. Consider the class of constraints $c_1 = x < i$ and $c_2 = x < j$, where i and j are integers. These two constraints match regardless of the association with publisher constraint or subscriber constraint, and regardless of the particular instantiated values of i and j.

3. ARCHITECTURE

Our previous work [23] showed that the system can be efficiently implemented as an application program executing transactions on a relational database management system (DBMS). The publish, subscribe, and match functions are implemented as application code that executes transactions on the database. The paper did not experimentally analyze the notify function.

3.1 Constraint Publication and Subscription

Data for each publication is stored in a publication relation, which for efficiency is partitioned by data type. For example, if the system supported constraints over integer, float and string types the database would contain the relations `integer_pubs`, `float_pubs`, and `string_pubs`.

The publish operation takes a constraint $\mathcal{C}$ as input and converts $\mathcal{C}$ into an equivalent constraint $\mathcal{C}'$ that is in disjunctive normal form. Negated atomic predicates are converted to equivalent predicates in a non-negated form if possible, and otherwise an error condition is returned to the publishing client (eg. "NOT integer x < 2" would be converted to "integer x ≥ 2"). For each disjunct in $\mathcal{C}'$ a tuple $(p\_id, p\_disjunct\_id, count)$ is inserted into a `pub_master` table which records the number of atomic predicates in the conjunctive clause of that disjunct. For each atomic predicate in $\mathcal{C}'$ a tuple $(p\_id, p\_disjunct\_id, field, value, operator)$ is inserted to the appropriate *`type`*`_pubs` table, where p_id is the unique identifier assigned to constraint $\mathcal{C}$ and attribute $p\_disjunct\_id$ specifies in which disjunct the atomic predicate occurs. The subscribe operation similarly takes a constraint as input and inserts such tuples into `sub_master` and *`type`*`_subs` tables. Essentially, the constraint is encoded into the relation by reifying the variables and embedding the values, and operators.

For example, consider a symmetric publish / subscribe system that implements less-than and equality over integers. The subscription constraints "$x < 6$ AND $y = 3$" and "$x < 4$ AND $y < 2$" generate the relation shown in Figure 1.

s_id	s_disjunct_id	field	val	op
1	0	x	6	<
1	0	y	3	=
2	0	x	4	<
2	0	y	2	<

Figure 1: Example instance of a subscription table, *integer_subs*

p_id	p_disjunct_id	field	val	op
3	0	x	5	<
3	0	y	1	=

Figure 2: Example instance of a publication table, *integer_pubs*

Similarly, the publication constraint "$x < 5$ AND $y = 1$" generates the relation shown in Figure 2.

3.2 Constraint Matching

A summary of the match operation is as follows. To compute the constraint intersections a query is issued for each possible pair of operators and type, grouped by pair of publish and subscribe constraint and disjunct identifiers. The query counts the number of matching conjuncts for the given pair of operators for each field and inserts the results into an intermediate answer table. An aggregate query is then executed over the intermediate answer table, determining the total number of conjuncts that matched for each disjunct. For each subscription disjunct this count is then compared to the total number of conjuncts for each disjunct in the `pub_master` table. If these counts are equal for a publication/subscription disjunct pair then the subscription and publication match, and the result is recorded for the notify function.

To improve the efficiency of matching, a multi-attribute index of `(id, disjunct_id)` is declared for the `pub_master` and `sub_master` tables. Similarly, a multi-attribute index of `(op, val)` is declared for each *type*`_pubs` and *type*`_subs` table where possible. For types where multi-attribute indexes are not supported, single attribute indexes on `op` and `val` are declared if possible.

Operators can be freely mixed between publication and subscription constraints for a particular field/type pair. Range and comparison operations on strings operate on the lexical ordering.

3.2.1 Generating the intermediate answer table

For each possible combination of operators and types, the system issues an explicit query that counts the satisfied intersections for a particular field and subscription. The results of this sequence of queries are inserted into a temporary table. Since operators can be freely mixed, a large number of queries may be generated. A rule-based query generator generates all possible combinations.

Constraint information				Answer	
p_id	p_disjunct_id	s_id	s_disjunct_id	field	count
3	0	2	0	y	1
3	0	1	0	x	1
3	0	2	0	x	1

Figure 3: Example intermediate answer table

Each query in the sequence has the same general form. For example, to generate matches between publications using integer less-than and subscriptions using integer greater-than the following query is issued:

```
INSERT INTO intermediate_answer
  SELECT p_id, p_disjunct_id,
         s_id, s_disjunct_id,
         p.field, COUNT(p.field)
    FROM integer_pubs p, integer_subs s
  WHERE p.op = '<' AND s.op = '>='
    AND p.field = s.field
    AND p.val > s.val;
  GROUP BY p_id, p_disjunct_id,
           s_id, s_disjunct_id
```

Figure 3 shows the intermediate answer table that would be generated for the example publication and subscription above after all such queries are issued. In this example, the first inserted row corresponds to the match of subscription 2's constraint "$y < 2$" with publication 3's constraint "$y = 1$." The second row corresponds to the match of subscription 1's constraint "$x < 6$" with publication 3's constraint "$x < 5$," while the third row corresponds to the match of subscription 2's constraint "$x < 4$"with publication 3's constraint "$x < 5$."

3.2.2 Counting the matches for each disjunct within the constraints

The intermediate answer table is then used to count the total number of matches for each disjunct within a publication/subscription combination. If the number of matches equals the number of conjuncts for some disjunct, then that publication/subscription combination is notified. This query is issued to compute the matching subscriptions from the intermediate answer table:

```
SELECT p_id , s_id
FROM intermediate_answer agg, pub_master pm
WHERE agg.p_id = pm.id
  AND agg.p_disjunct_id = pm.disjunct_id
GROUP BY agg.field, agg.p_id, agg.p_disjunct_id,
         agg.s_id, agg.s_disjunct_id, pm.count
HAVING sum(agg.count) = pm.count;
```

This query result is given to the notification system, which then notifies the appropriate clients.

3.3 Design Alternatives

The above section describes just one choice in a spectrum of design alternatives for the implementation of symmetric publish / subscribe, and in particular the encoding of constraints. One option would single subscription and publication relations that could accommodate the variety of data types. Since database schema definitions are well-typed, such a relation would require a distinct attribute column for each type (i.e., integer_val, float_val, etc.). The advantage of this design option is the simplification of the implementation; the disadvantage is the sparse nature of the encoding since each row of the table would contain mostly nulls.

Another option would be to avoid reification of variables and map each subscription variable into its own relation, e.g. "$x = 10$" would be encoded into either a relation

`subscription_integer_x` or perhaps just `subscription_x`. Such a choice has the advantage that matching constraints for a particular variable would be highly optimized since each variable is encoded as part of the schema, but the disadvantage of greatly increasing the number of queries that must be executed to compute the match function. Additionally, such a design would require the schema to adapt any time a new variable were introduced to the workload, a frequent occurrence for many applications.

Our choice to partition the subscription and publication tables by type is a compromise between these two extremes. This design obtains dense storage of constraints without requiring the explosive number of queries required to perform matching and the frequently-evolving schema of the latter alternative.

4. DISCUSSION OF EVALUATION

The evaluation methodology of [23] is based on the methodology described in [19, Section 2.2]. The goal of the evaluation was to explore the fundamental algorithmic issues of symmetric publish / subscribe. The goal implies a focus on the core matching algorithm. The performance of notification is clearly an important area of future work. Thus, the definition of the system under study is limited to a client workload generator and a server that processes the services of the system.

The strongest evaluation of any new system consists of both a collection of domain-specific benchmarks and synthetic benchmarks. The domain-specific benchmarks demonstrate the practicality of the system and the synthetic benchmarks demonstrate the breadth of the system. However, a review of DBMS benchmarks [17] did not reveal any implementations of publish / subscribe systems on a DBMS. Publish / subscribe benchmarks [7] provided some high level information about workloads and metrics but the evaluation of individual systems generally focused on the performance of notification operations (through the use of multicast, intelligent routing, etc.) Auction benchmarks [2, 22] provide a model for auctions, but these auctions were not complex enough to include the example given in the introduction. Thus, our search failed to turn up a benchmark that could be appropriately modified to evaluate symmetric publish / subscribe.

To work around these issues, [23] defined a synthetic set of workloads and metrics that directly explored the performance of the system.[1]

Metric selection posed little difficulty in the experimental design. Response time and throughput were the metrics chosen to match evaluation goals. Other metrics such as reliability, price/performance, and total cost of ownership may serve as future metrics.

The number of subscription constraints, the number of conjuncts, the number of (batched) publication constraints were the experimental factors studied. In addition, the application code was run on both a disk resident DBMS and a main memory DBMS. This experimental factor allowed the paper to draw conclusions about memory performance.

In general, system evaluation ranges across a variety of techniques: analytical results, queuing theory results, simulation (where software simulates all components), emulation (where hardware implements part of the systems), to prototypes. These techniques trade-off accuracy for effort.

In [23] both analytical and emulation performance measurements were reported. The analytical section reports the total worst-case time to compute all matches. In addition, the paper shows that in practical circumstances, the system operates in logarithmic time when processing a publication. These analytic results are valuable since they focus on (a) the interaction of the type system with operators and (b) a key use of indexes during match computation. In addition, these results are fairly independent of the particular details of the performance of existing hardware and software components. The emulation consisted of an implementation on a DBMS. The emulation functionality was very close to an actual prototype - only the notification functionality was missing.

A principle problem in the emulation experimental evaluation was the construction of synthetic workloads that produce reasonable result sizes for the number of matches and thus the number of notifications issued. Consider four experimental factors of (a) number of publisher constraints (1 or 100), (b) number of subscriber constraints (1,000 or 100,000), (c) number of variables (1 or 50), and (d) range of operators (the single equality operator "=" or the set of operators ($=, \neq, <, \not<$)). For the experimental workload of 100 publisher constraints, 100,000 subscriber constraints, 50 variables and equality comparisons, all constraints are of the form $var_i = val_j$ where i ranges from 1 to 50 and the distribution of val_j controls the result size. Uniformly selecting val_j from 1 to the number of subscribers will generate an expected result size of 2. However, consider the experimental workload with the inequality operator. No reasonable selection of val_j exists in this case, since many constraints are of the form $val_i \neq val_j$. The result size in this case will approach $\#publishers \cdot \#subscribers \cdot operators/variables$. This result size is unrealistically large.

To control for this problem, a variety of carefully crafted workloads were defined. These workloads explore the performance of the system as the number of publisher and subscriber constraints varies. The *fixed* workload has a constant result size independent of the number of subscribers. This workload models the case where new subscriber constraints are relatively independent from existing publisher constraints. The *proportional* workload has a result size that grows in linear proportion to the number of subscriber constraints. This workload models the case of a linear dependency between existing publisher constraints and new subscriber constraints. The *large-intermediate-results* workload explores the case where the intermediate result size grows with the number of subscription constraints, but the final result size is constant. This workload tests the performance of the intermediate result size generation step of the method. Finally, the *fixed-geometric* workload produces a fixed size result independent of the number of subscribers, but it is based on geometric types instead of primitive types. This workload tests the dependence of the system on geometric types to implement range constraints.

Each experiment essentially explores the modification of a single factor while holding other factors constant. Thus, the impact of a few factors is explored.

The experiments results measured response time performance for all workloads as the number of subscription con-

[1]These workloads and metrics are not a benchmark per se, but they could be packaged as a collection of synthetic micro-benchmarks.

straints scaled. This experiment demonstrated the impact of the optimizer on performance. As the number of subscription constraints grows, the optimizer successfully changes its plan to accommodate a shift in costs. The response time as the number of publication constraints are batched was also measured. In this latter case, the optimizer does not successfully accommodate the shift in costs and generates a poor plan.

5. EVALUATION OF EVALUATION

To systematically evaluate the evaluation described in [23], we qualitatively evaluated the paper based on a checklist of 23 common mistakes in performance evaluation [19, page 22]. This section lists the results of qualitative evaluation.

Although the goals of the performance evaluation were stated relatively clearly in [23], in some cases the paper left them as implicit and did not clearly state them. For example, the paper does not precisely define response time as the round trip time between the client machine and the server machine.

While the chosen workloads represented a variety of workload characteristics, the workloads were not based on the characteristics of an existing application or benchmark.

The use of two evaluation techniques for the paper provide an unusually broad set of results. Thus, we disagree with a reviewer of a previous version of [23] that stated "The analysis of Section 4 is meaningless to me. All that matters here are the timing numbers - especially in comparison with existing pub/sub systems." Precise measurements are valuable for a variety of reasons but analytical results concisely state the factors that have the largest impact on performance.

Since the implementation of [23] utilizes a complex DBMS as an underlying platform, there are a large number of potential parameters and factors that could be used in our experimental design. However, our experimental design was based almost solely on the characteristics of the publish / subscribe system and its workload, not based on the DBMS itself (since our goal was not to evaluate the DBMS).

Choosing the correct experimental factors is difficult because of the broad range of reported response times (from sub-second responses to tens of seconds). When measuring sub-second response time, parameters such as the network performance between the client and server or the performance of the database driver have a large impact on response time. However these parameters are irrelevant when the server processing time exceeds a few seconds.

The design did not perform a 2^k factor analysis but skipped directly to a detailed exploration to a few key parts of the experimental space.

With respect to the level of analysis, no tests for statistical significance were reported. However, the broad outline of results reported are all clearly statistically significant. Since the performance of the system was fundamentally related to the performance of the DBMS, a sensitivity analysis of the various parameters of the DBMS, such as the size of the buffer pool, etc. would reveal additional issues. One strong point of the analysis is the considerable time and effort spent on tracking down non-linear behavior in the reported results.

In summary, the evaluation described is [23] conforms to the current standard of performance evaluation in the database literature. Two areas for improvement are the study of variance in metrics and the introduction of a sensitivity analysis. Finally, the *reproducibility* of scientific results is a desirable goal and the paper carefully documents all aspects of the system and its evaluation. However, the testing of the claim of reproducibility would require an evaluation by an independent team of developers.

6. RELATED WORK

A general survey of publish / subscribe appears in [13]. This survey covers many distributed computing issues but does not cover content-based matching in depth, nor does it discuss the evaluation of publish / subscribe systems. Carzaniga and Wolf [7] describe in depth a list of model parameters and factors to consider when modeling publication records and subscription constraints.

The theory of constraint databases is outlined in Kanellakis et al. [20]. The particular class of constraints permitted in this paper does not directly map to the taxonomy of that paper since this system is based on operators. However, both papers use point-set constraints and overlapping subclasses of constraints. In particular, Kanellakis et al. describe the connection between constraint representation and spatial data structures. The work lists many analytical results.

CCUBE [4, 5] is a constraint database that combines database technology with in-memory linear constraint evaluation (via Simplex). LyriC [3] is the associated query language and object model. The overlap between symmetric publish / subscribe and constraint database functionality is an area of ongoing research.

Many works are concerned with aggregating subscriptions in classical publish / subscribe systems for more efficient content-based processing. Mühl [21] describes an algorithm for merging subscription constraints based on identical conjuncts. This paper does not contain a performance evaluation. Crespo et al. [9] explore optimization algorithms and cost models for subscription aggregation in a multicast environment. The paper includes a detailed performance evaluation featuring sensitivity analysis and price/performance computations.

The method of counting matched field and value pairs is similar to Yan and Garcia's [25] counting method for Boolean selective dissemination of information profiles. The technique of counting matched conjuncts appears in many works. The performance analysis is based on a mix of analytical and simulation results. Conjunctive predicate counting augmented with cache line analysis and other techniques is described in [14].

Our method of embedding multiple different types for a single generic value into a relation is similar to that of Yalamanchi, Srinivasan and Gawlick [24]. This work also describes a powerful generalization where expressions are treated as data and the evaluation of expressions can be combined with standard SQL processing. However, they do not appear to reify constraints, a key issue in the choice of a representation, nor do they use indexes, a key issue in performance.

Franklin et al. [1, 10, 12, 11] introduced and explored a method of compiling subscriptions into an in-memory finite state machine (FSM). The finite state machine represents common path prefixes of different subscriptions only once, thus providing a form of common sub-expression elimination. Matching a published document with subscriptions is implemented by traversing this FSM. The finite state machine methodology inspired several subsequent publications,

e.g. [18, 8].

Our previous work [23] is closely related to and inspired by Fink, Johnson and Hu's work [15] on an auction system that matches buy and sell orders. While auction systems and publish / subscribe systems differ in many details, they share some fundamental questions, such as index construction and its relationship to the complexity of the match operation. Fink et al. combine all constraints into a single large index. Each node of the index corresponds to a constraint attribute. Constructing this index requires choosing an attribute order and thus introduces a bias into the index search. The system described here does not exhibit this bias. However, auction systems and Fink et al. in particular compute the best match between a publisher and a set of subscribers. This problem is an open area of research for symmetric publish / subscribe.

Independently of our work, Fischer and Kossmann [16] analyze a variety of strategies for batching publications for the classical publish / subscribe case. Our batching results confirm that batching is an effective strategy for the symmetric publish / subscribe case. The various other strategies described by Fischer and Kossman probably apply.

Our previous work is similar in some respects to Chandrasekaran and Franklin's work on stream queries and data [6] that highly optimizes a particular transaction semantics of publish / subscribe, with the addition of support of query operations matches, maintenance of result sets, time windows, etc. However, this work does not consider constraints for publications.

7. CONCLUSION

Our previous work [23] is the first reported investigation into symmetric publish / subscribe systems. These systems permit publications as well as subscriptions to express constraints. The system computes the intersection of publisher constraints with subscriber constraints to determine matches.

In this paper we discussed the evaluation performed in our previous work and characterized its strengths and weaknesses. A major problem in our evaluation was that no existing publish / subscribe benchmark was well-suited to symmetric publish / subscribe, as all existing benchmarks implemented applications that require only publication of constants, not constraints as our system allows. Our evaluation instead relied upon a collection of synthetic workloads designed to mimic various properties of publishers and subscriber constraints that may be used with symmetric publish / subscribe in practice. In constructing these workloads and evaluating our system, we found that system performance was heavily dependent on the size of intermediate results in the computation of the publisher/subscriber matching algorithm.

For future work in evaluation, the additional expressive power of symmetric publish / subscribe provides a new area of research and new possibilities for applications of publish / subscribe systems. For us, one ongoing problem is to design and construct a benchmarking tool motivated by a real-world application (such as an auction system) that fundamentally utilizes the full expressive power of symmetric publish / subscribe.

8. ACKNOWLEDGMENTS

This material is based upon work supported by the Defense Advanced Research Projects Agency (DARPA) under Contract No. NBCHC030029.

9. REFERENCES

[1] Mehmet Altinel and Michael J. Franklin. Efficient filtering of XML documents for selective dissemination of information. *The VLDB Journal*, pages 53–64, 2000.

[2] Cristiana Amza, Emmanuel Cecchet, Anupam Chanda, Alan Cox, Sameh Elnikety, Romer Gil, Julie Marguerite, Karthick Rajamani, and Willy Zwaenepoel. Specification and implementation of dynamic web site benchmarks. In *WWC-5: IEEE 5th Annual Workshop on Workload Characterization*, 2002.

[3] Alexander Brodsky and Yoram Kornatzky. The LyriC language: Querying constraint objects. In *SIGMOD '95*, 1995.

[4] Alexander Brodsky, Victor E. Segal, Jia Chen, and Pavel A. Exarkhopoulo. The CCUBE constraint object-oriented database system. *Constraints*, 2(3,4):245–279, December 1997.

[5] Alexander Brodsky, Victor E. Segal, Jia Chen, and Pavel A. Exarkhopoulo. The CCUBE constraint object-oriented database system. In *Proceedings of SIGMOD 1999*, 1999.

[6] Sirish Candrasekaran and Michael J. Franklin. Streaming quereies over streaming data. In *VLDB*, 2002.

[7] Antonio Carzaniga and Alexander L. Wolf. A benchmark suite for distributed publish/subscribe systems. Technical Report CU-CS-927-02, Department of Computer Science, University of Colorado, 2002.

[8] Chee Yong Chan, Pascal Felber, Minos N. Garofalakis, and Rajeev Rastogi. Efficient filtering of XML documents with XPath expressions. In *ICDE*, 2002.

[9] Arturo Crespo, Orkut Buyukkokten, and Hector Garcia-Molina. Query merging: Improving query subscription processing in a multicast environment, 2003.

[10] Y. Diao, P. Fischer, M. Franklin, and R. To. YFilter: Efficient and scalable filtering of XML documents. In *The 18th International Conference on Data Engineering*, pages 341–342, 2002.

[11] Y. Diao and M. Franklin. Query processing for high-volume xml message brokering, 2003. Technical Report, University of California, Berkeley, http://citeseer.ist.psu.edu/diao03query.html.

[12] Yanlei Diao, Mehmet Altinel, Michael J. Franklin, Hao Zhang, and Peter Fischer. Path sharing and predicate evaluation for high-performance XML filtering. *ACM Trans. Database Syst.*, 28(4):467–516, 2003.

[13] Patrick Eugster, Pascal Felber, Rachid Guerraoui, and Anne-Marie Kermarrec. The many faces of publish/subscribe. *ACM Computing Surveys*, 35(2):114–131, 2003.

[14] Francoise Fabret, H. Arno Jacobsen, Francois Llirbat, Joao Pereira, Kenneth A. Ross, and Dennis Shasha. Filtering algorithms and implementation for very fast publish/subscribe systems. In *Proceedings of the 2001 ACM SIGMOD International Conference on Management of Data*, pages 115–126. ACM Press, 2001.

[15] Eugene Fink, Josh Johnson, and Jenny Hu. Exchange market for complex goods: Theory and experiments. *Netnomics: Economic Research and Electronic Networking*, 6(1):21–42, 2004.

[16] Peter M. Fischer and Donald Kossmann. Batched processing for information filters. In *ICDE*, 2005.

[17] Jim Gray, editor. *The Benchmark Handbook for Database and Transaction Processing Systems*. Morgan Kaufmann, 1993.

[18] Ashish Kumar Gupta and Dan Suciu. Stream processing of XPath queries with predicates. In *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, pages 419–430. ACM Press, 2003.

[19] Raj Jain. *The Art of Computer Systems Performance Analysis*. John Wiley, 1991.

[20] Paris C. Kanellakis, Gabriel M. Kuper, and Peter Z. Revesz. Constraint query languages. In *Proceedings 9th ACM PODS*, 1990.

[21] Gero Mühl. Generic constraints for content-based publish/subscribe. In *Proceedings of the 6th International Conference on Cooperative Information Systems (CoopIS '01)*, 2001.

[22] A. Schmidt, F. Waas, M. Kersten, M. Carey, I. Manolescu, and R. Busse. Xmark: A benchmark for XML data management. In *VLDB*, pages 974–985, 2002.

[23] Anthony Tomasic, Charles Garrod, and Kris Popendorf. Symmetric publish/subscribe via constraint publication. Technical report, Carnegie Mellon University, Department of Computer Science, 2006. CMU-CS-06-129.

[24] A. Yalamanchi, J. Srinivasan, and D. Gawlick. Managing expressions as data in relational database systems. In *Conference on Innovative Directions in Research*, 2003.

[25] T. W. Yan and H. García-Molina. Index structures for selective dissemination of information under the Boolean model. *ACM Transactions on Database Systems*, 19(2):332–334, 1994.

[26] Tak W. Yan and Hector Garcia-Molina. Index structures for information filtering under the vector space model. In *Proceedings of the Tenth International Conference on Data Engineering*, pages 337–347. IEEE Computer Society, 1994.