ISSN (Print) : 2320 – 3765
ISSN (Online): 2278 – 8875

International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 2, Issue 10, October 2013

Analysis of Popular Heuristics for Event Matching for Publisher and Subscriber System in Cloud Platform

Subasish Mohapatra[1], Bhagyashree Mohanta[2], Subhadarshini Mohanty[3]

Lecturer, Department of CSA, CET, Bhubaneswar, India[1]

Research Scholar, Department of CSE, ITER, Bhubaneswar, India[2]

Lecturer, Department of CSE, CET, Bhubaneswar, India[3]

ABSTRACT: Publisher/Subscriber System is receiving increasing attention for providing customized information delivery. Context-based event matching is vexing issues in large-scale event-matching publish/subscribe systems. Major constraints in distributed environment such as location, behaviour that motivate for designing efficient algorithms for scalable communication system. Analysis of its difficulty and evaluation of its solutions is still a problem. This paper contributes towards survey of existing event matching algorithms in publisher/subscriber system.

Keywords: Event matching, publisher/ subscriber system, cloud computing, content based model, rapid match algorithm.

I. INTRODUCTION

Publish/subscribe system typically consists of an overlay network of brokers. Events originate from publishers and are delivered by the brokers to interested subscribers. Traditional publish/subscribe is topic-based where subscribers subscribe to a set of redefined topics such as “Apple news” or “American Idol” [1].

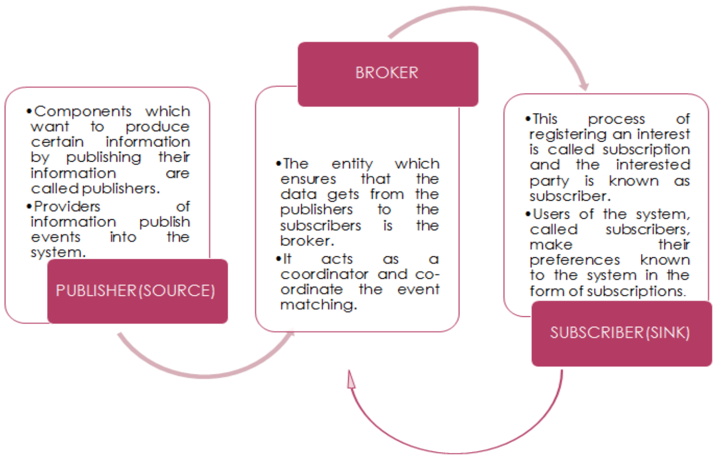
•Components which want to produce certain information by publishing their information are called publishers.
•Providers of information publish events into the system.
PUBLISHER(SOURCE)
BROKER
•The entity which ensures that the data gets from the publishers to the subscribers is the broker.
•It acts as a coordinator and co-ordinate the event matching.
•This process of registering an interest is called subscription and the interested party is known as subscriber.
•Users of the system, called subscribers, make their preferences known to the system in the form of subscriptions.
SUBSCRIBER(SINK)

Fig 1: Work-flow of Publish/Subscribe system

Content-based publish/subscribe, on the other hand, allows a subscriber to express an interest as a boolean predicate against values of attributes inside events [2][3]. For example, a subscriber may subscribe to eBay antique auctions with seller rating higher than 90 percent and starting bid between $100 and $200. Only events matching the predicate will be delivered to the subscriber [4]. Content-based publish/subscribe is of interest to both database and networking communities because it must address the dual challenges of subscription matching in an event space and event

ISSN (Print) : 2320 – 3765
ISSN (Online): 2278 – 8875

International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 2, Issue 10, October 2013

dissemination in the network space. Each subscriber needs to be assigned a broker which is responsible for forwarding matching events to this subscriber. Intuitively, we would like to assign subscribers with similar interests to the same broker, so that an event delivered to the broker could serve many subscribers. If all subscribers assigned to the broker have similar interests, then only a subset of all possible events needs to go through the broker. At the same time we may not want to assign a subscriber to a broker located far away in the network because by doing so increases delivery latency and communication cost. Finally, we should not assign too many subscribers to one broker, which creates a performance bottleneck and delays event delivery. Balancing these considerations—similarity of interests in the event space, proximity of locations in the network space, and balance of load across brokers—is a difficult optimization problem [4].

In the matching problem, we are given an events schema and a finite set of subscriptions (sub). Subsequently, we are given an event (e), and the goal is to determine all those subscriptions in sub that match e. We allow pre-processing of the set Sub before we are given e. A solution to the matching problem has two phases: pre processes and match pre processed data with the event. Initially in the pre-processing phase, the algorithm pre-processes the set of subscriptions into a data structure that allows fast matching [5]. Pre-processing makes sense in most pub/sub environments, where subscriptions tend to change infrequently enough that they can be considered approximately static, but where events are published at a fast rate. The first phase takes the set of subscriptions and outputs an internal representation of the subscriptions. The second phase match takes this internal representation an event and outputs of those subscriptions that match the event.

II. RELATED WORK

Various ways for specifying the events of interest have led to identifying distinct variants of the pub/sub paradigm. The subscription models that appeared in the literature are characterized by their expressive power. A highly expressive model increases the subscriber's possibility to precisely match their interest, i.e. receiving only the events they are interested in. In this section we briefly review the most popular pub/sub subscription models.

There are three principal types of pub/sub systems: topic-based, type-based and content-based systems.

A. *Topic-based Model*

Notifications are grouped in topics i.e., a subscriber declares its interest for a particular topic and will receive all events related to that topic. Each topic corresponds to a logical channel ideally connecting each possible publisher to all interested subscribers. For the sake of completeness, the difference between channel and topics is that topics are carried within an event as a special attribute.

Consider the example [6] of stock quotes disseminated to a large number of interested brokers. In a first step, we are interested in buying stocks, advertised by stock quote events. Such events consist of five attributes: a global identifier, the name of the company, the price, the amount of stocks, and the identifier of the selling trader. Fig 2 gives an overview of the resulting distributed interaction.

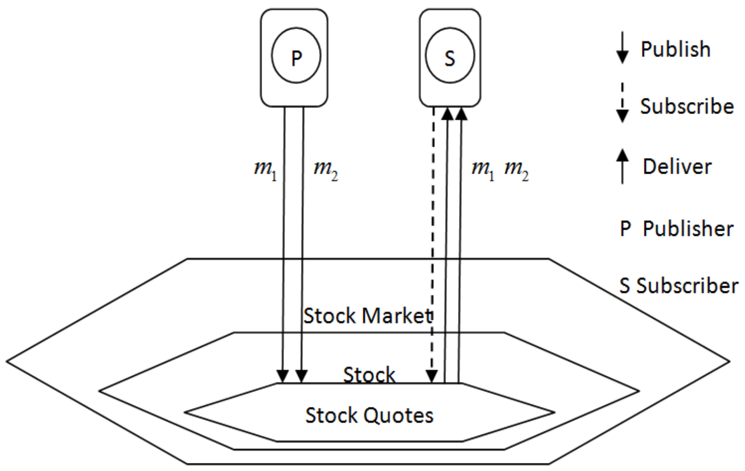

Fig 2: Topic-based Publish/Subscribe interactions

ISSN (Print) : 2320 – 3765
ISSN (Online): 2278 – 8875

International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 2, Issue 10, October 2013

Topic-based model has been the solution adopted in all early pub/sub structures. Examples of systems that fall under this category are TIB/RV [7], iBus [8], SCRIBE [9], Bayeux [10] and the CORBA Notification Service [11]. The main drawback of the topic-based model is the very limited expressiveness it offers to subscribers. A subscriber interested in a subset of events related to a specific topic receives also all the other events that belong to the same topic. To address problems related to low expressiveness of topics, several solutions are exploited in pub/sub implementations. For example, the topic-based model is often extended to provide hierarchical organization of the topic space, instead of a simple flat structure as in [7, 12]. Let A and B are two topics where topic B can be then defined as a sub-topic of an existing topic A. Events matching B will be received by all clients subscribed to both A and B.

B. Type-based

In type-based systems, a subscriber states the type of data it is interested in (e.g., temperature data). This type of systems is not very common. Producers publish message objects on a communication bus and consumers subscribe to the bus by specifying the types of the objects they are interested in. Message objects are considered as first class citizens and are classified by their types, instead of arbitrarily fixed topics. The type-based publish/subscribe is a new variant of distributed messaging based on the publish/ subscribe paradigm.

The example in Figure 3 illustrates type-based subscription. Stock events can be split into two distinct types: stock quotes (for sale) and stock requests. Brokers use stock requests to express their interest in buying stock [6].

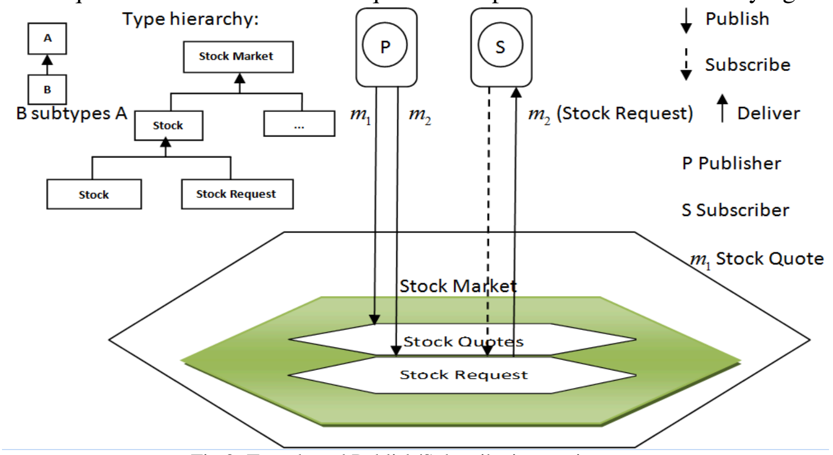

Fig 3: Type-based Publish/Subscribe interactions

C. Content-based Model

Content-based systems are the most versatile ones. The content-based publish/subscribe variant, which enables applications to describe runtime properties of messages the subscriber wish to receive. Here the subscriber describes the content of messages it wants to receive. Such a subscription could be for any messages like it containing both temperature and light readings where the temperature is below a certain threshold and the light is on. Possible constraints depend on the attribute type and on the subscription language. Most subscription languages comprise equality and comparison operators as well as regular expressions [13–15].

The subscriber would either have to filter out irrelevant events or topics that would need to be split into several subtopics—one for each company (And recursively several subtopics for different price “categories”, like price: 250 and price: 130) [6].

ISSN (Print) : 2320 – 3765
ISSN (Online): 2278 – 8875

International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 2, Issue 10, October 2013

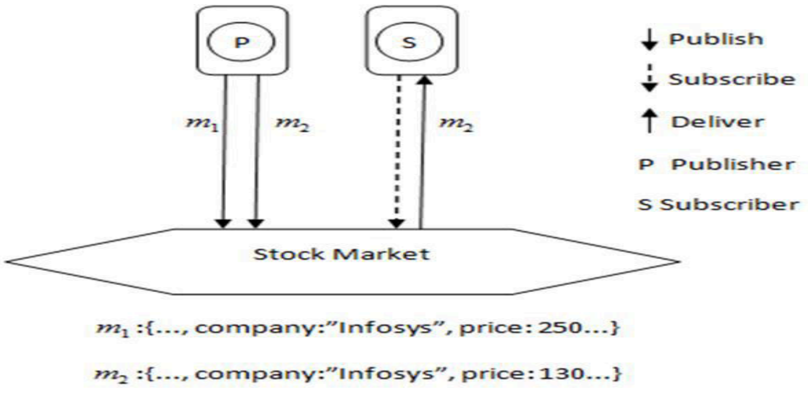
P
S
m_1
m_2
m_2
Publish
Subscribe
Deliver
P Publisher
S Subscriber
Stock Market
m_1 :{..., company:"Infosys", price: 250...}
m_2 :{..., company:"Infosys", price:130...}

Fig 4: Content-based Publish/Subscribe interactions

The complexity of the subscription language obviously influences the complexity of matching operation. It is not common to have subscription languages allowing queries more complex than those in conjunctive forms [16, 17]. A complete specification of content-based subscription models can be found in [18]. Examples of systems that fall under the content-based category are Gryphon [18], SIENA [19], JEDI [20], LeSubscribe [21], Ready [22], Hermes [23], and Elvin [24]. In content-based publish/subscribe, events are not classified according to some predefined criterion (i.e., topic name) but rather according to properties of the events themselves. As a consequence, the correspondence between publishers and subscribers is on an event basis.

III. HEURISTIC DESCRIPTIONS

For the publish/subscribe system, the following subsection describe the Analysis of some event matching algorithms.

A. *Naive Algorithm*

This is a simple algorithm for solving the matching problem consists of testing all subscriptions one by one against each incoming event [25]. Usually a matching algorithm may be used in environments with a large number of subscriptions (several hundreds of thousands or even more). In such environments, if the response time to process an event is an important factor or the rate of events to process is high, the naive algorithm is not a satisfactory solution. The main problem with this algorithm is the existence of a high redundancy in the evaluation of the predicates. In fact, the same predicate can be evaluated as many times as the number of times it appears in the set of subscriptions.

It evaluates the subscriptions one by one and for each subscription, evaluates its predicates until either one of them becomes false or all of them are evaluated.

```
Naive _match($S$, $e$)
1 // $S$ is the set of subscriptions, e is the event to
      match
2 matched $\leftarrow$ {}
3 for each $s \in S$ do
4     $m \leftarrow$ true
5     for each $p \in s.preds$ do
6        if ¬ predmatch ($p$, $e$) then
7           $m \leftarrow$ false
8           break // leave loop for
9        end if
10    end loop
11    if $m$ then
12       matched $\leftarrow$ matched $\cup$ s
13    end if
14 end loop
```

Fig. 5 Naive Algorithm

The limitation of the above proposed algorithm is the time taken to evaluate the matching process is too large as it evaluates all the subscription one by one. So to reduce the matching time we go for the tree match algorithm which is also less computationally expensive than Naive algorithm. This naive algorithm runs linearly to the number of subscriptions [5][26]. In practice, pub/sub systems may be deployed in environments with tens of thousands of publishers/subscribers, and in general pub/sub systems have been aimed at providing support for large-scale, widely distributed applications. Therefore, a linear time solution to the matching problem is not adequate.

B. *Tree Match Algorithm*

Tree based matching algorithm performs better than naive based algorithm. A tree-based algorithm [27] organizes subscriptions into a rooted search tree. Each node at level i of the tree represent a property p_i and have at most 3 successors corresponding to values $p_i \in \{0,1,*\}$. Each tree leaf contains a list of all subscriptions with predicate values specified by the path from the root to the leaf. Given an event $e \in \{0,1\}^k$, the search tree is traversed from the root to down. At every node, the value of the property for the event is tested and the satisfied branches are followed. All the matching subscriptions are obtained at the leaves of the tree that the algorithm hits.

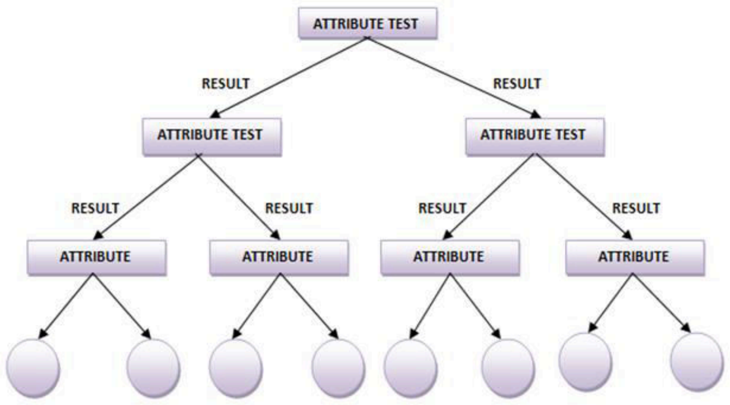

Fig. 6 Tree Match data structure

Here, in the pre-processing phase, our algorithm creates a matching tree [5]. In the matching tree each node is a test on some of the attributes and the edges are results of such tests. Each lower level of the tree is a refinement of the tests performed at higher levels and at the leaves of the tree we have the subscriptions. With such a tree, we can find the subscriptions that match an event by traversing the tree starting from the root at each node, we perform the test prescribed by the node and follow all those edges consistent with the result (there may be more than one edge). We then repeat these steps until we get to the leaves. The leaves that are finally visited correspond to the subscriptions that match the event.

C. *RAPID Match Algorithm*

RAPID Match is based on the tree-based matching approach. RAPID Match pre-processes subscriptions by partitioning them by the predicates corresponding to the "relevant" properties of events, so that for a given event, it can quickly eliminate large amounts of non-matching subscriptions and focus on a small subset of possibly matching subscriptions.

To validate the model, [27] the example of an online newspaper has been considered. The newspaper publishes articles periodically and wishes to notify subscribers when articles of their interest are published. Users specify their preferences in terms of the topics they are interested in, such as current affairs, sports, arts, etc. They may specify if they are interested, not interested, or neutral about each topic. For example, a subscription specifying the subscriber's interest in sports, a disinterest in politics and indifference about the rest may have a subscription like (sports = *), (politics = 1), (arts = 0) and … here, 1 indicates interest, 0 indicates disinterest, and * indicates "don't care". An event, such as an article concerning politics and arts but neutral about any other topic will be specified by (sports = *), (politics = 1) and (arts = 1) and other constraints. Formally, subscriptions consist of a conjunction of k predicates

ISSN (Print) : 2320 – 3765
ISSN (Online): 2278 – 8875

International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 2, Issue 10, October 2013

($p_1 = v_1$) and ($p_1 = v_2$) and ... ($p_k = v_k$) where the P_j are called properties and the v_i take values in {0, 1, *}. Events are represented in a similar fashion except that the v_i take values in {0, 1}. An event is said to match a subscription if its property values satisfy all predicates in the subscription, i.e., 0 and 1 in the subscription match 0 and 1 respectively in the event and * matches either. For conciseness, the property names are omitted and only the values are specified.

In this algorithm the terminologies used are,

e=Publishing event.

S=A set of subscriptions.

p_i = Properties of an event.

p_k =Predicates in the subscription.

k=No. of predicates.

P_j^t =No. partitions of a publishing event *e*.

T_j^t = Contains subscriptions which have either a 0 or a * in P_j^t .

S_t = Contains subscriptions with at most t 1's.

S= Any subscription that matches the event e, implies that $s \in S$

t=No. of 1's in event *e*.

1) *Subscription partitioning:* In this process first of all properties such as $p_1, p_2 ... p_k$ are divided into t+1 blocks of k/ (t+1) consecutive properties each. let r = k/ (t+1), and the partition is $| p_1 p_2 .. p_r || p_{r+1} p_{r+2} .. p_{2r} |...| p_{tr+1} p_{tr+2} .. p_{(t+1)r} |$. For clarity, the blocks are denoted as $P_1^t, P_2^t ... P_{t+1}^t$. Using this partition, the secondary partition are defined within RAPIDMatchSets as follows: S_t is partitioned into a disjoint and exhaustive collection of t+1 sets $T_1^t, T_2^t ... T_{t+1}^t$. The set T_j^t contains subscriptions which have either a 0 or a * in P_j^t . Note that since S_t contains subscriptions with at most t 1's and since there are t+1 blocks of predicates, at least one block of predicates must be totally devoid of 1's and so each subscription can be placed in at least one T_j^t . It is possible that a subscription may fit (and be put into) more than such sets. The partitions are illustrated in Fig. 7 [27].

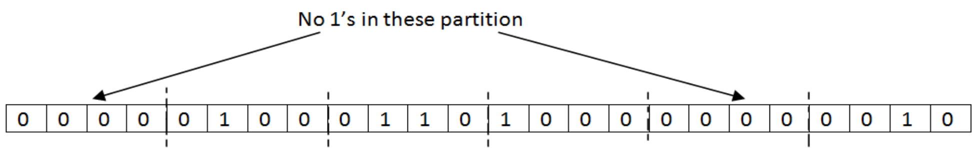

Fig. 7 Partitioning the properties of an exactly-5-light event. P (i, j) indicates P_j^i . partitions P_1^5 , P_5^5 have no 1's, so all matching subscriptions will be in either T_5^1 or T_5^5 .

ISSN (Print) : 2320 – 3765
ISSN (Online): 2278 – 8875

International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 2, Issue 10, October 2013

2) Data structures used in RAPID Match: Based on the partitions above, data structure is built in RAPIDMatch as follows [27]:

- Rapid Match partitions subscriptions into c+2 (not necessarily disjoint) RAPIDMatchSets $S_0, S_1, \ldots S_{c+1}$.
- For 0 ≤ t ≤ c, further partition S_t into t+1 sets $T_1^t, T_2^t, \ldots T_{t+1}^t$. For each such set, build a search tree, called RAPIDMatchTree for subscriptions in the set.
- Each RAPIDMatchTree T_j^t is a restricted search tree which handles all the properties in cyclic order starting from the property after P_j^t, namely p_{jr+1}, to p_k, then wrapping around to p_1 and ending at $p_{(j-1)r}$. Thus, the properties in block P_j^t are avoided.
- For the subscriptions in S_{c+1}, build a standard ($p_1, p_2, \ldots, p_k$)-restricted search tree, to test all the k properties.

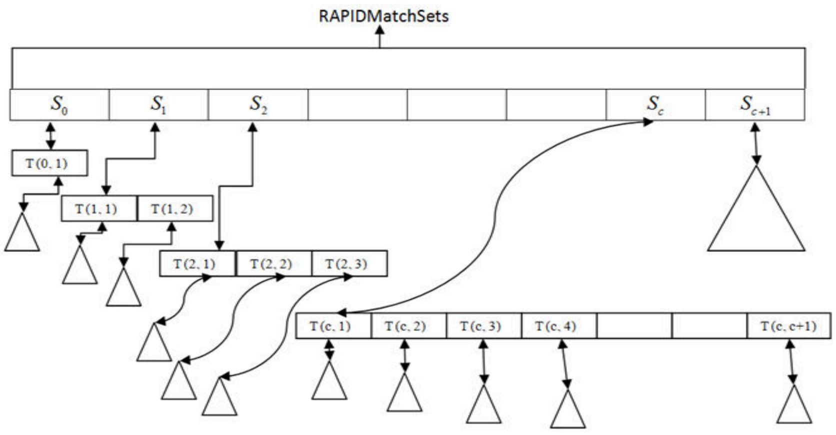

Fig. 8 RAPIDMatch data structure. T (i,j) denotes T_j^i.

- *Matching algorithm:* The RAPID Match matching algorithm runs as follows [27]:

  - Given an event e $\in \{0,1\}^k$, find t, the number of 1's in it.
  - If 0 ≤ t ≤ c, iterate over the $P_1^t, P_2^t, \ldots, P_{t+1}^t$ blocks of e to find an index i such that e has only 0's in P_i^t. i is guaranteed to exist, as e has only t 1's and there are t+1 blocks. Search RAPIDMatch Tree T_i^t, and return all matching subscriptions.
  - If t > c, search the tree of S_{c+1}, and return all matching subscriptions.

The limitation of the above proposed algorithm is, the algorithm runs efficiently only under the condition $t \le k/(t+1)$, but when that condition fails, the running time of the matching process increases and the efficiency of this algorithm decreases. So, we go for a modification of RAPID Match algorithm, named as Modified RAPID Match algorithm.

D. Proposed Modified Rapid Match Algorithm

In this algorithm the terminologies used that are,

e=Publishing event.

S=A set of subscriptions.

p_i = Properties of an event.

p_k =Predicates in the subscription.

k=No. of predicates.

P_j^t =No. partitions of a publishing event e.

T_j^t = Contains subscriptions which have either a 0 or a * in P_j^t .

S_t = Contains subscriptions with at most t 1's.

S= Any subscription that matches the event e, implies that s $\in$ S

t= No. of 0's or 1's in event e whichever is less in count.

The Subscription Partitioning and the Data Structures used in Modified RAPIDMatch Algorithm are same as used in RAPIDMatch Algorithm as stated above. But the modification of this algorithm is only in the matching process in 1^{st} point which is stated below.

1) Matching algorithm: The Modified RAPID Match matching algorithm runs as follows:

1. Given an event $e \in \{0,1\}^k$, find t, either less number of 1's or 0's in it.
2. If $0 \leq t \leq c$, iterate over the $P_1^t, P_2^t, \dots, P_{t+1}^t$ blocks of e to find an index i such that e has only 0's in P_i^t . i is guaranteed to exist, as e has only t 1's and there are t+1 blocks. Search RAPID Match Tree T_i^t , and return all matching subscriptions.
3. If $t > c$, search the tree of S_{c+1} , and return all matching subscriptions.

Instead of partitioning the subscriptions on the basis on number of 1's present in it, the algorithm can check for the Boolean value which occurs less number of times in the given event e. For example, if the event e is given as {0, 1, 1, 0, 1} and seven subscriptions are given as:

$$S_1 = \{0, 0, 1, 1, 1\}$$

$$S_2 = \{0, 1, 1, 0, 1\}$$

$$S_3 = \{*, 1, 1, *, 1\}$$

ISSN (Print) : 2320 – 3765
ISSN (Online): 2278 – 8875

International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering
(An ISO 3297: 2007 Certified Organization)
Vol. 2, Issue 10, October 2013

$$S_4 = \{0, 0, 0, 0, 0\}$$
$$S_5 = \{0, 1, *, 0, 1\}$$
$$S_6 = \{0, 1, 0, 0, 0\}$$
$$S_7 = \{1, 0, *, 0, 1\}$$

The given event has two 0's and three 1's. So, searching with respect to 0's would be easier than searching with respect to 1's. The subscriptions are partitioned on the basis of number of 0's. So, the groups are:

Zero 0's	One 0's	Two 0's	Three 0's	Four 0's	Five 0's
		S_1			
		S_2	S_2	S_2	
S_3	S_3	S_3			
					S_4
		S_5	S_5		
				S_6	
		S_7	S_7		

Table 1 Table of subscriptions

2) *Searching taking the positions into consideration.* Here, * is considered both as 0 and 1 and the subscription containing * appears in two columns first taking 0 and then taking 1. The algorithm doesn't consider the subscriptions containing all 0's or all 1's as they don't match the event. The algorithm then goes to the column containing two 0's (as the event has two 0's). We know that 0 is present in 1^{st} and 4^{th} position. Now, instead of linearly comparing the subscriptions with the event, it checks for the position of 0 in the subscription. It eliminates subscription S_7 as in the 1^{st} position, there is 1 and not 0. Considering the other two subscriptions, we find that 0's are present in both the required positions. Therefore, the subscriptions matching the event are S_2 and S_7.

E. *SGIM Algorithm*

SGIM algorithm operates in two stages. In first stage it pre- processes subscriptions by grouping them by the predicates corresponding to the relevant properties of events [28][29]. The basic grouping idea from statistics is employed to decide the number of. In the second stage matching subscriptions or predicates are derived sequentially. All predicates stored in the system are associated with a unique ID. Similarly subscriptions are identified with subscription id.

The pub-sub components workflow in our framework is as follows:

- Users register their information and subscriptions to various SaaS applications which then transfer all this information to pub/sub broker registry.
- When sensor data reaches to the system from gateways, event/stream monitoring and processing component (SMPC) in the pub/sub broker determines whether it needs processing or just store for periodic send or for immediate delivery.
- If sensor data needs periodic/emergency delivery, the analyser determines which SaaS applications the events belong to and then passes the events to the disseminator along with application ids.
- The disseminator, using event matching algorithm, finds appropriate subscribers for each application and delivers the events for use.

SGIM algorithm is a fast and scalable event matching algorithm called Statistical Group Index Matching. SGIM algorithm operates in two stages. In first stage it pre-processes subscriptions by grouping them by the predicates corresponding to the relevant properties of events. The basic grouping idea from statistics is employed to decide the

ISSN (Print) : 2320 – 3765
ISSN (Online): 2278 – 8875

International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 2, Issue 10, October 2013

number of groups. In the second stage matching subscriptions or predicates are derived sequentially. All predicates stored in the system are associated with a unique ID. Similarly subscriptions are identified with subscription id.

In case of normal range predicates or highly overlapping range predicates, it can greatly reduce the search space for fast event matching since it can directly choose or delete one or more group of subscriptions just by checking the index of groups. Then using sequential searching, when any predicate in a subscription is evaluated to false, there is no need to evaluate the remaining predicates. Furthermore, if this predicate is shared by many subscriptions, all remaining predicates in these subscriptions can be ignored, which can significantly reduce the evaluation overhead. The evaluation cost of SGIM is linear to the number of subscriptions.

The algorithm for inserting a new subscription in the system is very similar to the event matching algorithm. First the algorithm finds the group index which matches the subscription and insert into that group. If necessary, the group index may be updated for that subscription insertion. The cost of the inserting algorithm is close to the event matching cost. Deletion algorithm is very easy. Just we need to find the exact group index of the subscriptions to be deleted and then delete that from the group and if necessary group index is updated. If whole group needs to be deleted then we can search that group index and delete it.

IV. EXPERIMENTAL SETUP

In this experiment four matching algorithms were implemented for evaluation in Java. All experiments were conducted on a Pentium IV processor machine running at 2.0 GHz with 512 MB of RAM. We generate subscriptions ranging from 250-1000 predicates with 50 attributes. Each subscription predicate takes value 0, 1, *. The performance results are shown in seconds.

V. SIMULATION RESULTS

In the simulation we have plotted the graph between the runtime and the no. of subscriptions. Here, three matching algorithms were implemented for evaluation. It has been observed that, in comparison of Naive with RAPIDMatch and Modified RAPIDMatch algorithms, the performance of the Naive algorithm greatly reduces. But in the comparison of RAPIDMatch and Modified RAPIDMatch algorithms, Modified RAPIDMatch algorithms show the better result. In the other plot in Fig.11, we compare all four algorithms and conclude that when the no. of subscriptions is less, then all the three matching algorithms such as Naive, RAPIDMatch and Modified RAPIDMatch algorithms perform better than SGIM algorithm but the SGIM algorithm outperforms from other three algorithms, where the number of subscriptions is too large.

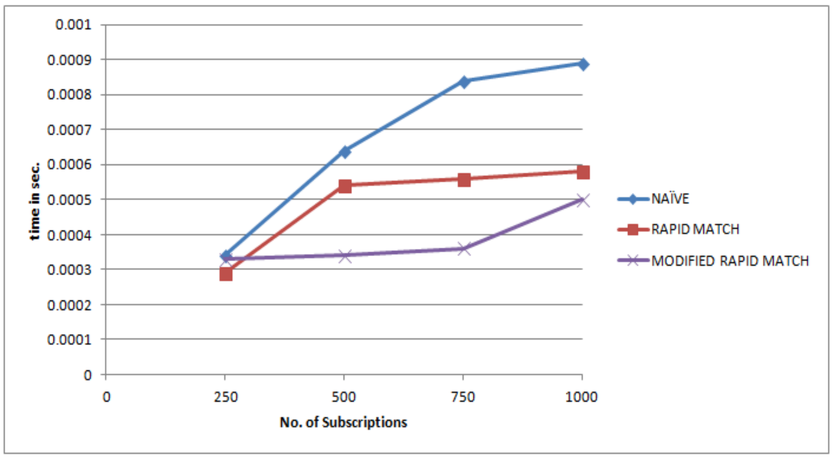

Fig. 11 Comparison of Naive, RAPIDMatch, Modified RAPIDMatch algorithms.

ISSN (Print) : 2320 – 3765
ISSN (Online): 2278 – 8875

International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 2, Issue 10, October 2013

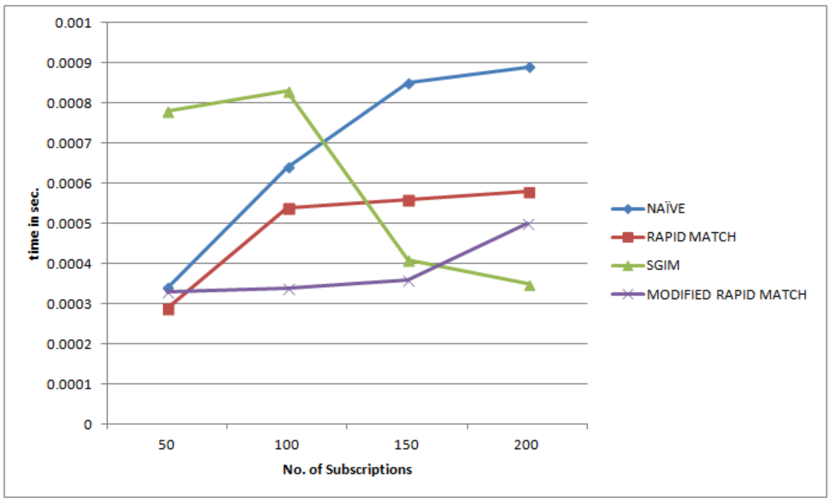

Fig. 11 Comparison of Naive, RAPIDMatch, Modified RAPIDMatch and SGIM algorithms.

XI. CONCLUSION

Three different event matching problems in content based publisher subscriber system are shown, such as Naive algorithm and Tree based algorithms in Fig.10. Tree based algorithms are more efficient than other algorithms. From the observations it has been concluded that the proposed/developed modified RAPIDMatch algorithm outperforms other algorithms. Our proposed Modified RAPIDMatch partitions subscriptions offline based on their predicate characteristics so that for given event we can quickly identify a small subset of relevant subscription and reduce its search space. But SGIM outperforms other three algorithms when the no. of subscriptions is too high. The efficiency of event matching algorithms is shown in the graph. Our future work is to develop deeper understanding and more comprehensive evaluation using realistic publisher subscriber information.

REFERENCES

[1] Yu, A., Agarwal, P. K., and Yang, J., "Content-Based Publish/Subscribe", IEEE, Transactions On Knowledge And Data Engineering, Vol. 24, No. 10, October 2012.
[2] Cheung, A.K.Y., Jacobsen, H. A., "Green Resource Allocation Algorithms for Publish/Subscribe Systems", Proceedings - International Conference on Distributed Computing Systems, DOI:10.1109/ICDCS.2011.82 In proceeding of: Distributed Computing Systems (ICDCS), 2011.
[3] Baldoni, R., Marchetti, C., Virgillito, A. and Vitenberg, R., "Content- Based Publish-Subscribe over Structured Overlay Networks", in the proceedings of ICDCS'05, 2005.
[4] Cheung, A.K.Y., Jacobsen, H.A., "Load Balancing Content-Based Publish/Subscribe Systems", University of Toronto, Volume 28, No. 9, Issue 4 ,2010.
[5] Aguilera, M.K., Strom, R.E. Stunnan, D.C. AsHey, M., Chandra, T.D., "Matching Events in a Content-based Subscription System", in the Proceedings of PODC '99 Atlanta GA USA Copyright ACM I-581 13-099-6/99/05, 1999.
[6] Eugster, P.T., Pascal, A. , Felber, Guerraoui, R., Kermarrec, A.M., "The Many Faces of Publish/Subscribe", ACM Computing Surveys, Vol. 35, No. 2, pp. 114–131, 2003.
[7] Oki, B., Pfluegel, M., Siegel, A., Skeen, D., "The information bus - an architecture for extensive distributed systems", In the proceedings of the 1993 ACM Symposium on Operating Systems Principles, 1993.
[8] Altherr, M., Erzberg, M. , Maffeis, S., "iBus - a software bus middleware for the java platform", In: Proceedings of the International Workshop on Reliable Middleware Systems. Pp, 49–65,1999.
[9] Castro, M. , Druschel, P., Kermarrec, A. , Rowston, A., "Scribe: A large-scale and decentralized application-level multicast infrastructure", IEEE Journal on Selected Areas in Communications ,2002.
[10] Zhuang, S.Q.,Zhao, B.Y.,Joseph, A.D. , Katz, R., Kubiatowicz, J. , "Bayeux: An architecture for scalable and fault-tolerant wide-area data dissemination", In: 11th Int. Workshop on Network and Operating Systems Support for Digital Audio and Video. 2001.
[11] Object Management Group: CORBA event service specification, version 1.1. OMG Document formal/2000-03-01, 2001.

[12] Baehni, S. , Eugster, P.T., Guerraoui, R., "Data-aware multicast", In: Proceedings of the 2004 International Conference on Dependable Systems and Networks , pp. 233–242, 2004.
[13] Carzaniga, A., Rosenblum, D. , Wolf, A. "Design and Evaluation of a Wide-Area Notification Service", ACM Transactions on Computer , pp. 332–383,2003.
[14] Fabret, F., Jacobsen, A., Llirbat, F. , Pereira, J. , Ross, K. , Shasha, D., "Filtering algorithms and implementation for very fast publish/subscribe", In: Proceedings of the 20th Intl. Conference on Management of Data (SIGMOD 2001),pp.115–126,2001.
[15] Campailla, A. , Chaki, S. , Clarke, E.M. , Jha, S. , Veith, H. , "Efficient filtering in publish-subscribe systems using binary decision diagrams", In: Proceedings of The International Conference on Software Engineering, pp.443–452,2001.
[16] Bittner, S. , Hinze, A. , "On the benefits of non-canonical filtering in publish/subscribe systems", In the Proceedings of the International Workshop on Distributed Event-Based Systems (ICDCS/DEBS'05), 2005.
[17] Muhl, G., "Generic Constraints for Content-Based Publish/Subscribe", In: Proceedings of the 6th International Conference on cooperative Information Systems (CoopIS), 2001.
[18] Gryphon Web Site: (http://www.research.ibm.com/gryphon/
[19] SIENA Web Site: http://www.cs.colorado.edu/users/carzanig
[20] Cugola, G., Nitto, E.D. , Fuggetta, A. , "Exploiting an event-based infrastructure to develop complex distributed systems", In the Proceedings of the 10th International Conference on Software Engineering (ICSE '98), 1998.
[21] Pietro,R.P. Pereira, J., Llirbat, F. , Fabret, F., Oss, K. , Shasha, D. , "Publish/subscribe on the web at extreme speed", In the Proceedings of ACM SIGMOD Conf. on Management of Data, Cairo, Egypt, 2000.
[22] Gruber, R.E., Krishnamurthy, B. , Panagosf, E. , "The architecture of the ready event notification service" In the Proceedings of The International Conference on Distributed Computing Systems, Workshop on Middleware, Austin, Texas, 1999.
[23] Pietzuch, P. , Bacon, J. , Hermes, "A distributed event-based middleware architecture", In the Proceedings of the International Workshop on Distributed Event-Based Systems (DEBS'02). (2003)
[24] Segall, B., Arnold, D. , Elvin Has Left the Building "A Publish /Subscribe Notification Service with Quenching", In the Proceedings of the 1997 Australian UNIX and Open Systems Users Group Conference, 1997.
[25] Hall, C. P., Carzaniga, A. , Rose, J., and. Wolf, A.L., "A content-based networking protocol for sensor networks", Department of Computer Science, University of Colorado, Technical Report, 2004.
[26] Fabret,F., Llirbat,F., Pereira,J., Shasha,D., "Efficient Matching for Content-based Publish/Subscribe Systems", Founded by "Instituto Superior T´ecnico" - Technical University of Lisbon and by a JNICT fellowship of Program PRAXIS XXI (Portugal).
[27] Kale,S., Hazan,E., Cao,F., Singh,J.P., "Analysis and Algorithms for Content-based Event Matching", Supported by Prof. Sanjeev Arora's David and Lucile Packard Fellowship, NSF grants CCR 0205594 and CCR 0098180.
[28] Hassan, M.M., Song,B., Huh,E.N., "A Framework of Sensor - Cloud Integration Opportunities and Challenges" ,The research was supported by the MKE (Ministry of Knowledge Economy), Korea, under the ITRC (Information Technology Research Centre) support program supervised by the IITA (Institute of Information Technology Advancement) (IITA-2009-(C1090-0801-002)).
[29] Hunkeler,U., Truong, H. L., and Stanford-Clark, A. , "MQTT-S – A Publish/Subscribe Protocol For Wireless Sensor Networks", In IEEE Conference on COMSWARE, 2008.

BIOGRAPHY

He received the B.E. degree in Computer Science & Engineering from the Biju Patnail University of Technology, Odisha, India, in 2003, and the M.Tech. degrees in Computer Science & Information Technology from Biju Patnail University of Technology, Odisha, India, in 2007 and currently pursuing Ph.D. degree in National Institute of Technology, Rourkela, India. Currently he is working as a lecturer in the department of CSA, College of Engineering and Technology, Bhubaneswar, India. His current research interests include distributed computing, Clod computing and sensor network.

She received the B.Tech. degree in Computer Science & Engineering from the Biju Patnail University of Technology, Odisha, India, in 2010, and the M.Tech. degrees in Computer Science & Information Technology from Siksha 'O' Anusandhan University, Odisha, India, in 2013. Her current research interests include Clod computing, Grid Computing and web Technology.

She received the B.Tech. degree in Computer Science & Engineering from the Biju Patnail University of Technology, Odisha, India, in 2010, and the M.Tech. degrees in Computer Science & Information Technology from Siksha 'O' Anusandhan University, Odisha, India, in 2013 and currently she is working as a lecturer in the department of Computer Science & Engineering, College of Engineering and Technology, Bhubaneswar, India. His current research interests include distributed computing, Clod computing and Bioinformatics.