Substring Matching in P2P Publish/Subscribe Data Management Networks

Ioannis Aekaterinidis
R&A Computer Technology Institute and
Dept. Computer Engineering and Informatics
University of Patras, Greece
aikater@ceid.upatras.gr

Peter Triantafillou
R&A Computer Technology Institute and
Dept. Computer Engineering and Informatics
University of Patras, Greece
peter@ceid.upatras.gr

Abstract

The content-based publish/subscribe (pub/sub) paradigm for system design is becoming increasingly popular, offering unique benefits for a large number of data-intensive applications. Coupled with the peer-to-peer technology, it can serve as a central building block for such applications deployed over a large-scale network infrastructure. A key problem toward the creation of large-scale content-based pub/sub infrastructures relates to dealing efficiently with continuous queries (subscriptions) with rich predicates on string attributes; in this work we study the problem of efficiently and accurately matching substring queries to incoming events.

1. Introduction

During the past few years a number of solutions have been presented dealing with data management in a large-scale distributed environment typically based on a p2p network. Related work covers the data management problem with two different points of view. The first one is mainly concentrated on applying traditional database functionality over (typically DHT[1]-based) p2p networks where the data is being appropriately indexed among peers and queries arrive at the system that trigger a mechanism for collecting the data matching them. The other is related to the publish/subscribe (pub/sub) approach where (continuous) queries (*subscriptions*) are indexed in the network and the data (*events*) are matched to the already stored queries.

In this work we concentrate our efforts on the pub/sub point of view and especially the content-based pub/sub model that dominates the area since it allows for greater user-query expressiveness by defining predicates on content values (of predefined attributes) and matching events based on content. Possible applications include auction services (such as eBay) matching buyers and sellers, delivery of software updates across wide areas, distributed computing of large data volumes (such as in computational Biology), distributing gaming, stock market services, etc.

Contributions

With this work we will provide solutions to the substring matching problem under two different environments: the one is a DHT-independent environment where our solutions may be applied on top of any DHT-based p2p network and the other is a DHT-dependent environment, which assumes the existence of a particular popular class of DHTs (and specifically DHTs that employ prefix- (or suffix-) based routing a la Pastry[13]). In the DHT-dependent environment, we shall utilize a specific distributed data structure, coined PastryStrings, developed to support prefix and suffix predicates and presented in [3].

Given these environments, we will propose and study two different suites of algorithms (for subscription and event processing). The first (simpler) one is based on producing the complete set of substrings of a given string and is coined *complete*. The second is based on the *ngrams* method [9] where strings are decomposed into a finite set of *n-characters* long substrings.

2. Background and related work

2.1. Distributed hash tables (DHTs)

Distributed Hash Tables (DHTs [1, 13, 14]) are becoming increasingly popular for structuring overlay topologies of large-scale data networks. Since one of the environments for which we will develop solutions is DHT specific, we overview here the functionality of Pastry. Pastry [13] is based on a distributed data structure introduced by Plaxton et. al. [11] optimized for routing and locating objects in a very large network with constant size routing tables. Assuming a static network, routing tables consist of multiple levels, where in each level i there are pointers to nodes

[1] Distributed Hash Tables

whose identifiers (or node ids) have the same i-digit long suffix with the current node's id. The routing of messages is achieved by resolving one digit of the destination id in each step i and looking at the $i+1$ level of the local routing table for the next node. This mechanism ensures that a node will be reached in at most $O(log_\beta N)$ logical hops, where N is the namespace size and β[2] is the base of ids. The size of the routing table is constant and equal to $\beta \times log_\beta N$ (with $log_\beta N$ levels/rows and β columns per level). Pastry [13] offers a robust, scalable, and self-organizing extension to Plaxton's approach under a dynamic environment.

2.2. The publish/subscribe paradigm

The main challenge in a distributed pub/sub environment is the development of an efficient distributed event processing (matching) algorithm and related efficient algorithms to store subscriptions in the network. Distributed solutions have been provided for topic-based pub/sub systems (Scribe [8], daMulticast [5]). Some attempts use routing trees to disseminate the events to interested users based on multicast techniques [7, 10, 15, 16, 18]. Typically, processing subscriptions and/or events in these approaches requires $O(N)$ messages in *N*-node networks. Additionally, there exist techniques for subscription summarization that significantly reduce this overhead [18].

Prior relevant research includes our previous work to support numerical-typed attributes over DHTs [17], string-typed attributes [2] and finally prefix/suffix/equality predicates on strings and ranges/equality predicates on numerical values over Pastry [3]. In [6] events and subscriptions are mapped to bit-strings which are then used to process them in multiple DHT nodes. The work in [12] proposed a distributed trie structure called *Prefix Hash Tree* (PHT) which is built on top of a DHT p2p network and can support range queries and prefix string queries. PHT, like [2, 17], and [6] enjoy universal applicability (as they are based solely on the DHT's lookup function). However, they suffer from a number of performance drawbacks and particularly the message complexity of processing range and string queries.

To our knowledge, the only prior work that has addressed the issue of substring queries in DHTs is [9], where a sketch was presented of how the ngrams method could be adopted in DHTs.

3. The model

The DHT broker network

The typical pub/sub architecture consists of broker nodes forming the p2p network and clients "attached" to them. Client nodes are producers or consumers, issuing events and/or subscriptions, respectively. For every subscription, there is a broker in the DHT network where the subscriber is attached and keeps metadata information about the subscription.

In a pub/sub system there are two main tasks to be accomplished. The matching of events to subscriptions and the delivery of matched events to interested clients. In this work we mainly concentrate on the former, while for the later we can borrow already known techniques found in the literature.

For simplicity of presentation, we will assume that events consist of a single string-typed value (e.g. abcd) and subscriptions consist of a single substring predicate on a string value (e.g. *bc*). An event matches a subscription *if and only if* the subscription's value is indeed a substring of the event's value. Each subscription, i is identified by its $SubID_i$ which encapsulates meta-data about the subscription and its origin (the broker to which the subscriber is attached).

The extension of this simple single-attribute scheme to the multi-attribute approach with alphabet base β is fairly straightforward (in [2] the reader can find alternative algorithms for doing so). Briefly, we process each attribute of the subscription, i, with the methodology presented in section 4, and store the $SubID_i$ in specific nodes. When an event arrives we process each one of its k attributes and collect k lists of stored $SubIDs$. A subscription, i, is considered to match the event if its $SubID_i$ is found in as many collected lists, as the number of attributes defined in the subscription.

PastryStrings functionality

PastryStrings ([3], which is adopted in one of our solutions) is a scalable pub/sub infrastructure able to handle subscriptions with equality/prefix/suffix predicates on string values and equality/ranges (and $\geq$, $\leq$, etc.) predicates on numerical values. The two primary design choices that best describe PastryStrings are (i) a tuned Pastry (or any other Plaxton-like DHT) network with an alphabet base β appropriately defined so as to map string values (every possible spoken word) to nodes and (ii) a tree structure (known as *string trees*) on top of Pastry dedicated for (a) storing subscriptions and (b) matching events to subscriptions using prefix-based routing a la Pastry.

There are two types of nodes in PastryStrings. Network (Pastry) nodes (referred to as simply "nodes", "brokers", or "peers") and *string tree* nodes (referred to as $Tnodes$). A node in general hosts several $Tnodes$. Nodes have ids (assigned by Pastry) while $Tnodes$ have labels and are responsible for storing subscriptions matching that label. Each $Tnode$ has a constant size routing table of size equal to β with entries pointing to the $Tnode$'s children. Each $Tnode_{lbl}$ with label lbl starting with the same character i, form a *string tree* T_i .

[2] β is the number of different digits in node ids

4. Algorithms and problem statement

We first show the DHT-independent approach (IND) and then the approach relying on the PastryStrings infrastructure (PS). The problem can be stated as follows: given that the maximum string length of the attribute value is l, and that a subscription defines a substring predicate with value $s[1...l_s]$ with $l_s \leq l$, we must locate and collect the subscription upon the arrival of an event, if $s[1...l_s]$ is a substring of the event's value $e[1...l_e]$ ($l_s \leq l_e \leq l$).

4.1. The complete approach : complete-IND

Suppose that a subscription arrives with a substring predicate with value $s[1...l_s]$. We hash the string value (using the DHT specific hash function) and store $SubID$ to the peer with id numerically closest to the hashed value. Each peer, along with the stored list of $SubIDs$, maintains a list V storing the value of each one of the stored $SubIDs$. When an event arrives with value $e[1...l_e]$ ($l_s \leq l_e$) we hash all possible substring values of e and query those nodes for possibly stored $SubIDs$ whose values are in the list V and are substrings of e. That procedure will require to contact $\frac{l_e \times (l_e+1)}{2}$ peers. Given that in almost any DHT-like p2p network contacting a peer requires $O(logN)$ messages, we conclude on a $O(l_e^2 \times logN)$ complexity per event. Subscription processing, on the other hand, requires only $O(logN)$ messages.

4.2. The ngrams approach: ngrams-IND

The main idea behind *ngrams* [9] is to split each string (related both to subscriptions and events) into n-character long substrings, and use those substrings for storing subscriptions and locating them when events arrive. Given a string value $v[1...l_v]$ and *ngrams* with $n = 2$ (that is *bigrams*), the i^{th} *ngram* of v, is $v_i[i...(n+i-1)]$, $i \leq l_v - n + 1$. For example the 3 *bigrams* of the string *abcd* are: *ab*, *bc*, and *cd*.

When a subscription arrives with value $s[1...l_s]$, we first break s to $(l_s - n + 1)$ *ngrams*. Then we store the $SubID$ into all nodes with ids numerically closest to the hashed value of each *ngram*. Now when an event arrives with value $e[1...l_e]$ we split e to $(l_e - n + 1)$ *ngrams* and contact those nodes whose ids are numerically closest to the result of hashing the event's *ngrams*, for stored $SubIDs$. The collected lists are grouped based on $SubIDs$ and the number of copies of each $SubID$ is counted. The event is considered to match the subscription if the number of $SubID$ copies returned, is equal to the number of the subscription's *ngrams* and is less or equal to the number of the event's *ngrams*. The matching of events to subscriptions under this simple approach needs attention since false positives may occur. This is the case where the event and subscription have the same *ngrams* however *ngrams* may occur in different order in the subscription. For example *aba* and *bab* have the same set of *ngrams*: *ab* and *ba*.

Algorithm	Subscription Processing	Event Processing
complete-IND	$O(logN)$	$O(l_e^2 \times logN)$
ngrams-IND	$O(l_s \times logN)$	$O(l_e \times logN)$
complete-PS	$O(l_s + logN)$	$O(l_e^2 + logN)$
ngrams-PS	$O((l_s \times n) + logN)$	$O((l_e \times n) + logN)$

Table 1. Message complexity overview

In order to avoid this case we need to store along with the $SubID$ three extra fields, resulting in the *extended* $SubID$ containing : (i) the $SubID$ itself, (ii) the order of appearance of the *ngram* in the string, (iii) the number of *ngrams* in the string, and (iv) the *ngram* itself. The first two extra fields are needed for avoiding false positives, while the third for identifying $SubIDs$ matching a specific value. A typical *extended* $SubID$ consists of the following fields:

$SubID$	order	number of ngrams	ngram value

Extended $SubID$

Given that the lists of *Extended* $SubIDs$ are returned by querying the appropriate nodes based on the event's *ngram* values, we first count the size of each *Extended* $SubID$ group returned. We conclude on event-subscription matching if: (i) the group size is equal to the 3^{rd} field, which means that all *ngrams* of the subscription's value were collected, and (ii) the reconstruction of the subscription's substring based on the 2^{nd} field (showing the order) results on an event's value substring (It is clear that this is a local substring matching procedure for which a number of related algorithms have been proposed and it is out of the scope of this paper).

The improvement here compared to the complete-IND approach is that we need to contact considerably fewer nodes during event processing. In fact we have to contact $l - n + 1$ nodes as opposed to $\frac{l_e \times (l_e+1)}{2}$. Considering that n is much smaller than the string length we conclude on $O(l \times logN)$ message complexity for both events and subscriptions. What we have to pay compared to the *complete* approach is (i) the extra state per $SubID$ resulting in storing the *Extended* $SubID$, (ii) the local processing for concluding on possible matching after we collect all $SubID$ lists for the given event, and (iii) an increased network traffic since the same $SubID$ will be collected multiple times ($l_s - n + 1$ times for a subscription with value $s[1...l_s]$).

4.3. The complete approach over PastryStrings : complete-PS

It is possible to take advantage of the PastryStrings infrastructure to further improve the performance of event and

subscription processing. Subscriptions are stored inside the *string tree* dedicated to the first character of the substring, while events are processed by visiting all *string trees* whose characters are found in the event string in order to discover subscriptions stored inside them, and match the event.

When a subscription arrives with value $s[1...l_s]$, we locate the appropriate *string tree* dedicated to the first character of the substring, $T_{s[1]}$, and then we follow the path towards $Tnode_{s[1...l_s]}$ by resolving one character at a time. That node is going to store the $SubID$. The message complexity for subscription processing is $O(logN)$ for locating the *string tree* and $O(l_s)$ for walking in the *string tree* towards the right $Tnode$.

Upon the arrival of an event with value $e[1...l_e]$, we remove one character at a time from the beginning of the string and treat the suffix derived as a separate event. For each suffix $e_i[i...l_e]$ we locate the appropriate *string tree* $T_{e[i]}$ and following the path towards $Tnode_{e_i[i...l_e]}$ we collect matching subscriptions from all intermediate nodes. This procedure is continued until we reach the last character.

What we achieve with this approach is to check all possible substrings of the event's value and conclude on possible matches, by jumping from one *string tree* to another. Since locating the root of a *string tree* would require $O(logN)$ message, we could extend PastryStrings routing state, so that each root node holds an extra table of pointers to all other roots. This would reduce message complexity to $O(logN)$ in order to find the first root plus $O(l)$ in order to find the rest (l is the string length) as opposed to $O(l \times log(N))$.

4.4. The ngrams approach over PastryStrings : ngrams-PS

In this approach we choose to split strings in *ngrams* and process them on top of the PastryStrings infrastructure. Subscription processing involves the breaking of the substring $s[1...l_s]$ into $(l_s - n + 1)$ *ngrams* and for each one of them locating the appropriate *string tree* (based on the first character of the *ngram*) reaching the right $Tnode$ inside the tree and storing there the extended $SubID$ (as defined in section 4.2).This procedure requires $O(logN)$ messages to reach the appropriate *string tree* for each *ngram* and $O(n-1)$ to locate the appropriate $Tnode$ inside the tree.

Event processing is achieved following the same procedure. Whenever we locate stored $SubIDs$ we collect them and perform the post-processing explained in section 4.2 in order to conclude on possible matching.

5. Performance analysis and evaluation

5.1. Message complexity analysis

Given a subscription with a substring predicate $s[1...l_s]$ and an event with value $e[1...l_e]$, the event and subscription processing complexity in terms of hop count (or message count) in an N-node network can be seen in Table 1.

As you can see the preferable algorithm when looking at subscription processing is complete-IND. However, event processing under this scheme costs a lot and since events are expected to arrive at much higher rates than subscriptions, event processing should be as fast as possible. The second approach, ngrams-IND, performs slightly better than complete-IND when looking at event processing which is our main concern. On the other end lies the encapsulation of these two techniques in the PastryStrings infrastructure with ngrams-PS being the preferred one among all.

5.2. Experimentation

We tested the proposed algorithms in an 800-broker simulated network with up to 100,000 stored $SubIDs$ and up to 560,000 generated requests for collecting them. The algorithms perform on a multi-attribute event-subscription schema of up to 5 attributes. We used a Zipfian popularity distribution for attributes (determining the actual number of attributes in an event or subscription) and values for each attribute.

Our experimentation results mainly involve subscriptions defining only substring predicates. However, we also tested how the proposed approaches perform on top of PastryStrings when all supported predicates (prefix, suffix, substring, ranges, equalities [3]) can be declared in each subscription.

Due to space limitation, we only present Table 2 where one can view all proposed algorithms and their expected performance in terms of: (i) the *DHT Selection Flexibility*: can we apply the algorithm on top of any DHT?, (ii) the *Extra Routing State*: do we have to maintain a specific data structure in order to process events and subscriptions?, (iii) the *Number of Messages* during event processing, (iv) the *Total Network Traffic* generated during the event matching phase, and (v) the *Result Set Sizes* of the returned $SubID$ lists during event processing. The table classifies the algorithms using the scale *Very Bad*, *Bad*, *Good*, and *Very Good* for each metric.

In general complete-IND has excellent performance in terms of result set sizes (since no duplicates $SubIDs$ are transmitted). However, it has poor performance in terms of the number of messages since one message is sent for each possible substring (which also introduces additional message traffic). On the other hand, ngrams-IND shows

	complete-IND	ngrams-IND	complete-PS	ngrams-PS
DHT Selection Flexibility	YES	YES	NO	NO
Extra Routing State	NO	NO	YES	YES
Number of Messages (events)	Very Bad	Very Good	Very Good	Very Good
Total Network Traffic	Good	Bad	Good	Very Bad
Result Set Sizes	Very Good	Bad	Very Good	Bad

Table 2. Performance overview of the proposed algorithms

very good performance in terms of number of messages but suffers from much higher result set sizes at each peer which introduces great network traffic overhead.

Complete-PS and ngrams-PS ameliorate some of the performance problems of complete-IND and ngrams-IND at the expense of introducing additional routing state (and associated maintenance overhead) and sacrificing the flexibility of choosing any underlying DHT.

6. Concluding remarks

Internet-scale data management systems with continuous queries and event-based communication are an interesting area of research and development. In this area we propose and extensively study a number of algorithms and optimizations aiming on efficient substring processing of string-typed queries. Specifically, we present the *complete* and *ngrams* techniques applied on (i) any structured DHT-based p2p network and (ii) specific DHTs performing prefix-based routing.

Our analysis and experimentation shows that *complete* and *ngrams* both offer advantages and disadvantages over different metrics. For detailed performance results with respect to the number of messages and bandwidth requirements as well as for a method offering very good performance in all metrics of interest of Table 2, see [4].

7 Acknowledgments

This work has been partially supported by the IST Programm of the European Union number IST-2004-001907 (DELIS) and the European Social Fund (ESF), Operational Program for Educational and Vocational Training II (EPEAEK II), and particularly the Program PYTHAGORAS.

References

[1] K. Aberer. P-grid: A self-organizing access structure for p2p information systems. In *CoopIS'01*, 2001.

[2] I. Aekaterinidis and P. Triantafillou. Internet scale string attribute publish/subscribe data networks. In *14th ACM Conference on Information and Knowledge Management (CIKM05)*, 2005.

[3] I. Aekaterinidis and P. Triantafillou. Pastrystrings: A comprehensive content-based publish/subscribe dht network. In *the 26th IEEE International Conference on Distributed Computing and Systems (ICDCS 06)*, 2006.

[4] I. Aekaterinidis and P. Triantafillou. Publish/subscribe information delivery with substring predicates. Technical Report TR2006/11/11/01, [URL: http://netcins.ceid.upatras.gr/publications.php], Research Academic Computer Technology Institute, Patras, Greece, 2006.

[5] S. Baehni, P. Th, and E. Guerraoui. Data-aware multicast. In *DSN '04*, 2004.

[6] R. Baldoni, C. Marchetti, A. Virgillito, and R. Vitenberg. Content-based publish-subscribe over structured overlay networks. In *ICDCS '05*, 2005.

[7] A. Carzaniga and A. Wolf. Forwarding in a content-based network. In *Proc. SIGCOMM'03, 2003*, 2003.

[8] M. Castro, P. Druschel, A. Kermarrec, and A. Rowstron. Scribe: A large-scale and decentralized application-level multicast infrastructure. *Journal on Selected Areas in Communication*, 2002.

[9] M. Harren, J. M. Hellerstein, R. Huebsch, B. T. Loo, S. Shenker, and I. Stoica. Complex queries in dht-based peer-to-peer networks. In *IPTPS'02*, 2002.

[10] R. Huebsch, J. M. Hellerstein, N. Lanham, B. T. Loo, S. Shenker, and I. Stoica. Querying the internet with pier. In *VLDB'03*, 2003.

[11] C. G. Plaxton, R. Rajaraman, and A. W. Richa. Accessing nearby copies of replicated objects in a distributed environment. In *ACM Symposium on Parallel Algorithms and Architectures*, 1997.

[12] S. Ramabadran, S. Ratnasamy, J. M. Hellerstein, and S. Shenker. Brief announcement: Prefix hash tree. In *ACM PODC*, 2004.

[13] A. Rowstron and P. Druschel. Pastry: Scalable and distributed object location and routing for large-scale peer-to-peer systems. In *ACM Middleware'01*, 2001.

[14] I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *SIGCOMM'01*, 2001.

[15] C. Tang and Z. Xu. pfilter: Global information filtering and dissemination using structured overlays. In *In Proc. of FTDCS*, 2003.

[16] W. W. Terpstra, S. Behnel, L. Fiege, A. Zeidler, and A. P. Buchmann. A peer-to-peer approach to content-based publish/subscribe. In *DEBS'03*, 2003.

[17] P. Triantafillou and I. Aekaterinidis. Publish-subscribe over structured p2p networks. In *DEBS 04*, 2004.

[18] P. Triantafillou and A. Economides. Subscription summarization: A new paradigm for efficient publish/subscribe systems. In *IEEE ICDCS04*, 2004.