Publish/Subscribe on Top of DHT Using RETE Algorithm

Yan Shvartzshnaider[1], Maximilian Ott[2], and David Levy[1]

[1] School of Electrical and Information Engineering
The University of Sydney, Australia
yshv6985@uni.sydney.edu.au,
david.levy@sydney.edu.au
[2] National ICT Australia (NICTA)
max.ott@nicta.com.au

Abstract. This paper discusses the construction of a Global Semantic Graph (GSG) [1] to support future information- and collaboration-centric applications and services. The GSG is a publish/subscribe (pub/sub) based architecture that supports publication of tuples and subscriptions with standing graph queries. We believe that an implementation of an efficient pattern matching algorithm such as Rete [2] on top of a distributed environment might serve as a possible substrate for GSG's pub/sub facility. Rete operates on loosely coupled alpha, beta and join nodes and therefore has been chosen by us for implementation in a distributed setting.

In this paper, we propose a way to perform Rete's pattern matching over a DHT-based Structured P2P network to provide a scalable content-based publish/subscribe service.

Keywords: publish/subscribe system, distributed pattern matching, global semantic graph.

1 Introduction

The distributed publish/subscribe (pub/sub) interaction schema is viewed by many [3, 4, 5] as a more suitable communications paradigm for future Internet architecture as it better reflects the dynamic and asynchronous nature of today's Internet applications and services [6].

A typical pub/sub system offers a loosely coupled, event-based communication schema by offering a spatial, temporal, and synchronisation decoupling between publishers and subscribers, which is claimed to work well in a large-scale distributed environment [6, 5]. There are several variations of pub/sub schemas such as *topic-based*, *content-based* or *typed-based* schemas [6]. In this paper we focus on the implementation of a content-based (or property-based [5]) publish/subscribe schema where the subscription is based on the properties of the published information. To implement such schema on a large and distributed scale there is a need for an efficient and scalable pattern matching system that will support "*event filters*, which are predicates on the content of associated information, and *event patterns*, which are predicates on the relationships among event occurrences" [5].

A.J. Berre et al. (Eds.): FIS 2010, LNCS 6369, pp. 20–29, 2010.

The Artificial Intelligence (AI) community has widely studied the development of efficient matching algorithms in relation to *production rule systems*. Since their initial introduction in 1943 this work has greatly evolved [7] and is now is widely adopted by various AI systems, such as experts systems. A production rule system, in its simple design, comprises of three main parts: a *set of rules*, a *dataset* and *a rule interpreter* [8]. The rules are evaluated against the dataset by the interpreter and appropriate action is executed. For our purposes, we are primarily interested in the rules evolution part. In particular, our design is based on Rete – an efficient pattern matching algorithm – which is widely employed in production rule- based systems to match data against multiple productions. Rete operates on loosely coupled alpha, beta and join nodes and therefore has been chosen by us for implementation in distributed settings.

The emerging Peer-to-Peer (P2P) networks [9, 10, 11], in particular structured P2P networked overlays have been used as a building block for many global sharing and content distribution systems [12]. In such networks the core characteristics such as efficient routing, key-search, self-organisation, fault tolerance and good load balancing are offered by a Distributed Hash Table (DHT). Thus, in their basic form, they provide a relatively simple "hash-like" interaction interface (e.g., *get(key), put(key, value)*) which is not sufficient for a content-based publish/subscribe system [13] that requires support for more complex and expressive queries. Our system inherits support for such queries from the Rete algorithm.

The rest of the paper is organised as follows. Our motivation for the project is in Section 2. The design overview is covered and discussed in Section 3. In Section 4 we briefly review the Rete algorithm. Section 4.2 provides an overview of related work. Finally, Section 5 summarises our conclusions and gives some notes on future work.

2 Motivation

Our main motivation for this work is the development of a Global Semantic Graph (GSG) [1] to support future information- and collaboration-centric applications and services. The GSG is an Internet-scale tuple store that adopts approaches and methods from the Semantic Web to provide applications and services with the abilty to simply publish their internal state changes via simple tuple insertions, while a subscription is essentially a standing query to a specific pattern that keeps the internal state synchronised with any insertion of new information.

We strongly believe that a global infrastructure like the GSG will provide a convenient, powerful, and sound basis for building novel information-centric applications and services.

3 Design Overview

As mentioned in the introduction, in our approach we port the Rete algorithm onto a DHT-based overlay network to provide a scalable content-based publish/subscribe system.

Our system treats a tuple as a primitive – publications are tuples and subscriptions comprise of rule tuples which are bound by variables into a standing graph query. As a standing query, it remains in the system and continuously monitors the tuple-space to return a set of matched tuples. Each rule tuple is converted to tuple templates by dismissing the binding variables in the rule and replacing them with wildcards – i.e., only rules' constant attributes matter. Hence, for example, the (?a left-to ?b) rule template is represented with a (* left-to *) template. Every tuple template is associated with tuple-storage (also called alpha-memories) in the Rete network. It is worth noting that the *alpha memories* in a Rete network act as a tuple's local cache. Effective caching policies are an open research question in many fields and can significantly improve the performance of the overall system. Although all subscriptions are stored and managed locally by a single Rete network, we achieve scalability by distributing tuple templates among the DHT nodes. The classic DHT architecture is extended to support a separate storage for published tuples and tuple templates. This includes: mapping of every tuple and tuple template to unique DHT keys, similar to RDFPeers [14] and introduction of a new insert_condition(key, tuple pattern, subscribing node ID) method. Thus, as depicted by Figure 1, during publication, all tuples will be routed to a rendezvous node and matched against the tuple templates stored there. In case of a match, the tuple is forwarded to the subscribing node to complete the matching process.

3.1 Implementation

Subscription. As mentioned above, each subscription is comprised of rule-tuples that are converted to tuple templates and distributed over the DHT. In more

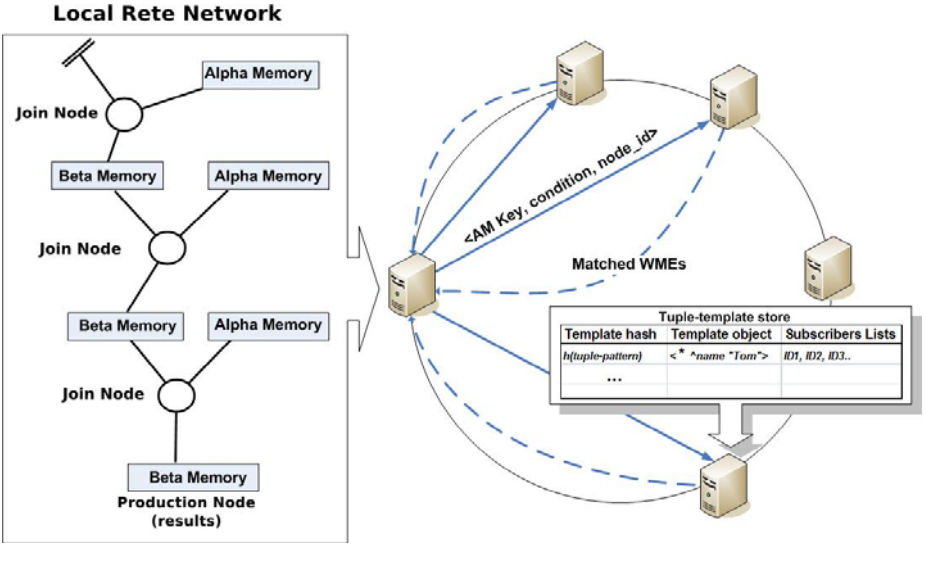

Fig. 1. An extremely simplified illustration of our implementation approach

practical terms, we pick any constant from a tuple template associated with the rule and hash it to create <key,tuple template>. The tuple template is then routed to a designated rendezvous node where it is stored in a "Tuple template Store". See Figure 1 for a simplified illustration of this process.

Publication. During the publication of a tuple, we hash each of the fields separately (e.g., *h(subject), h(predicate), h(object)*) to create three unique DHT keys for the same tuple, each key is used to route a copy of the tuple to a different rendezvous node. By using the same hash function for both subscription and publication, we guarantee that tuple and relevant tuple templates will rendezvous. The tuples matched at the *rendezvous node* are then delivered to relevant subscribers nodes.

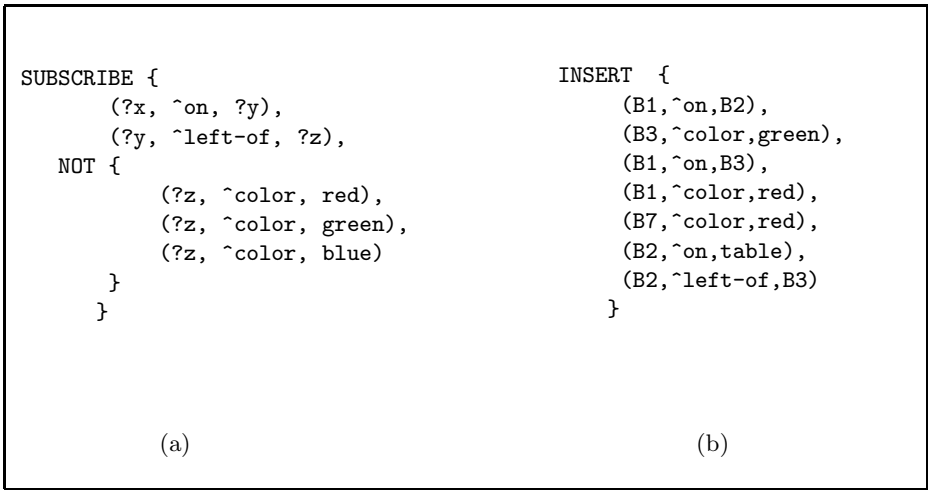

Fig. 2. Script Example (a) create a subscription, (b) inserts (publishes) collection of tuples

For our implementation of the Rete algorithm we have used the newly emerging Scala[1] language. Scala is a general purpose, object oriented and functional hybrid programming language. The Scala compiler produces Java binary code, and hence can be seamlessly integrated with existing Java solutions. This allows us to use our implementation with openChord[2], a Java implementation of the Chord [9] algorithm. Our implementation also includes a small Backus–Naur Form (BNF) grammar parser for a scripting language, similar to SPARQLs graph pattern query[3] syntax, to allow creation of more complex subscriptions and publication commands. See Figure 2 for a sample syntax of the script.

[1] http://www.scala-lang.org/
[2] http://open-chord.sourceforge.net/
[3] http://www.w3.org/TR/rdf-sparql-query/

3.2 Discussion

We have implemented our approach successfully and we are able to facilitate a distributed content-based publish/subscribe with support for expressive and complex querying over a structured overlay network. We recognise, however, that there is a place for future improvement.

One problem is *load-balancing*. We generate multiple key pairs for the same tuple, hence tuples with identical fields would be always routed to the same node and can cause significant overload when a particular tuple's attribute is shared by many (e.g., predicate: *suchAs* or object: *class* in RDFS scheme).

Our initial ideas is to tackle the *load-balancing* problem by introducing built-in time stamps with every tuple to indicate the time and date range of the publication, for example, *<10/3/2010-23/3/2010,subject, predicate, object>*. So, when hashing the tuple's fields we are insuring a different hash every time. Consequently, when creating a subscription, we need to indicate what is the tuple's valid query range – the range limit must be set in advance to insure proper matching. If a user specifies a range bigger than the limit it would be split into several ranges and subsequently create a separate subscription query per range.

The other issue that is open for debate is *push* versus *pull*-based approach. The current implementation is push-based system, that is, the matched tuples are pushed to the subscribing node. The benefit of this approach is that there is no the need to constantly query for updates, however the subscribing node can find itself overloaded with updates in a case of multiple and/or complex subscriptions. An alternative pull-based system avoids the overloading problem by giving control over the flow of updates back to the subscriber. That is, a subscriber node must periodically query the data store for updates. For our future work, we will explore the possibility of an hybrid querying model: by default the updates will be "pushed" to subscribing nodes, however, if a node becomes overloaded, some update streams will be blocked and queried later.

Finally, we are extending the current implementation and slightly changing the behaviour of a classical DHT-based overlay network. Our ultimate goal is work towards a "lightweight approach" [15] that is, to implement a pub/sub service without changing the DHT-generic algorithm. One option that we are considering is replacing the current hash function that is used to generated DHT keys with an alternative mapping function that better supports range and multi-attribute queries.

In Figure 3 above, subscription 1 returns matches of *all the Movies produced by Steven Spielberg* and Subscription 2 *matches all the Movies produced by Steven Spielberg with Tom Hanks as an actor*. The results of Subscriptions 1 are a subset of the total results returned by Subscription 2, therefore some of the *alpha memories* can be re-used and shared.

Ultimately, we envision the Rete network at the subscribing node utilising the DHT nodes as *alpha memories*, so that similar subscriptions from different subscribers will be able to reuse already available *alpha-memories* and also include other subscriptions as part of their Rete network. For example, as shown in Figure 3, the Rete network for Subscription 2 can comprise of new conditions in

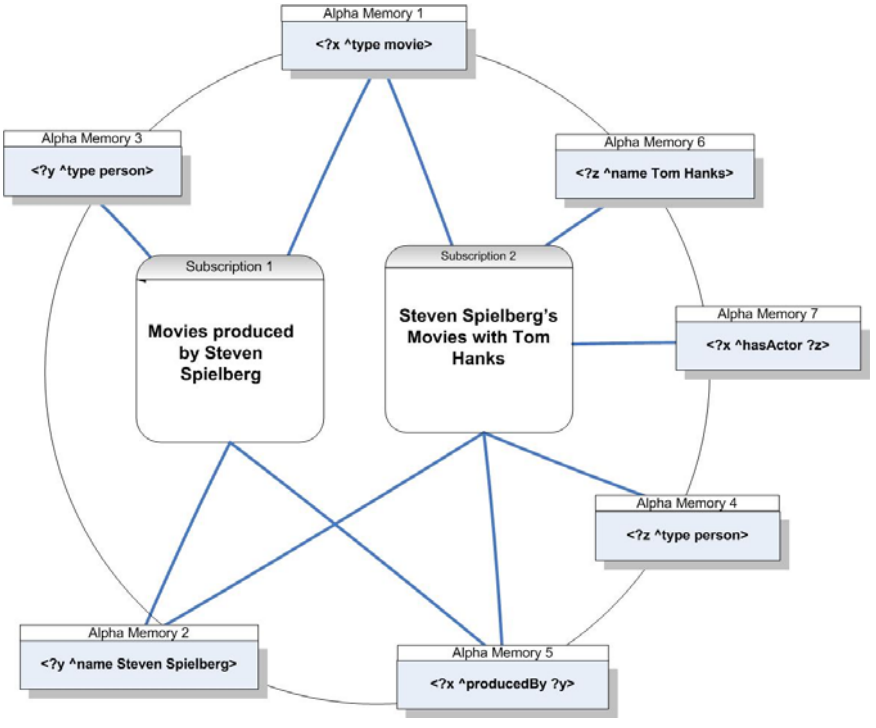

Fig. 3. An intuitive example showing the matching of subscription

conjunction with those previously defined Subscription 1. Such a configuration would allow more efficient utilisation of the Rete algorithm and provide a better platform for our future work [1].

Evaluation. As discussed in Section 3.2, the RETE-based pub/sub service allows the reuse and sharing of the results from past subscriptions. The system will perform better with time as more subscriptions are created. The GSG can take full advantage of this as it will deal with large number of simultaneous users, each with many complex and expressive subscriptions. To our knowledge, benchmarks available to date do not take into account these operational conditions. Hence, there is a need for a new benchmark to properly evaluate these types of systems. A proper evolution is part of our future work and is out of scope of this paper.

4 Rete Algorithm

This section provides a brief summary of the Rete algorithm from Chapter 2 of Doorenbos' [16] thesis and the motivation for using it in our implementation.

Rete [2] is an efficient and well-adapted pattern matching algorithm that is widely employed in production-based expert systems. It reduces the matching process time, by exploiting the "temporal redundancy" and "structural similarity" of the data; each state of a matching cycle is saved separately in the loosely coupled Rete dataflow network and can be reused in future pattern matches [17].

4.1 Rete Primitives

Rete operates on *productions* and *working memory elements* (WME). A *production* is defined by set of *conditions* (also called rules) that are evaluated against the WME dataset and *actions* which are executed when these conditions have been met.

4.2 Rete Dataflow Network

As depicted in Figure 4, the Rete-dataflow network comprises of *alpha memory*, *beta memory* and *join* nodes. The *alpha memory* (AM) node is part "alpha network" that acts as a predicate on the WMEs. For example, the AM node in Figure 4, that is defined by Condition 1 (<?x> ***on*** <?y>) contains WMEs that match the corresponding template (i.e., WME with attribute **on**). The AM treats *?x, ?y* as wildcards.

The *beta memory* (BM) nodes contain the result of *join* nodes. The input to "join" is the previous *beta node* and the corresponding *alpha* node. In other words, the *join* node filters through all the WME joined on variable values to the related *beta* nodes in a form of a join tuple; this process is called *left activation*.

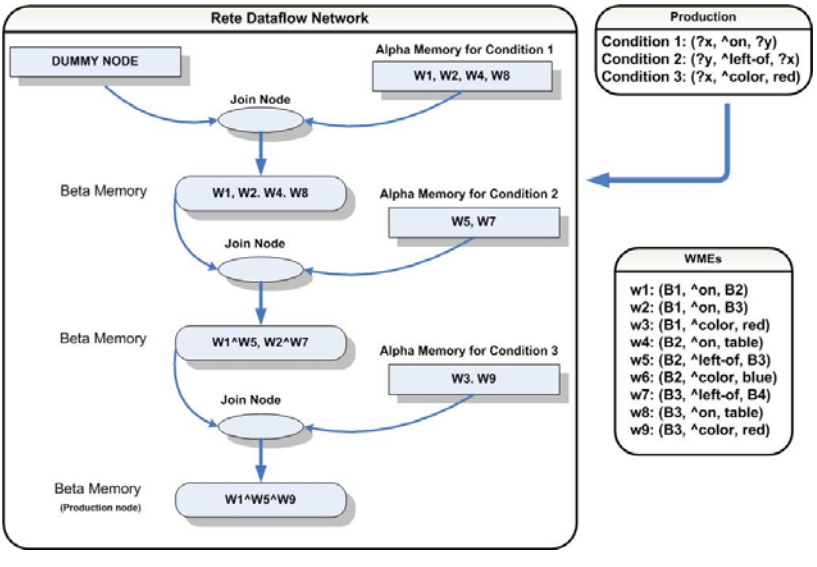

Fig. 4. Rete dataflow network acts a filter, inspired by [16]

On other hand, *right activation* is triggered by feeding a new WME into an appropriate AM node.

The matching process is a chain reaction of either *right* or *left* activations by the end of which all the matched WMEs are found in the *production node* which are essentially any last BM node in the chain flow.

5 Related Work

Several research efforts that have looked into implementing a content-based publish/subscribe communication schema over a DHT overlay.

In [18] the authors propose a rendezvous-based approach to publish/subscribe over the DHT overlay. The *rendezvous nodes* that are responsible for matching events, start the notification process. The system comprises of a three-layer design: *application*, *CB-pub/sub* and *overlay network*. The *CB-pub/sub* layer utilises the underlying overlay network, to provide generic applications that run on it with publish (pub), subscribe (sub) and notification (notify) functionalities. The system performs attribute-split, key space-split and selective-attribute mappings of subscriptions and published events onto the key space. The underlying overlay network is extended with an implementation of an additional one-to-many send primitive (*m_cast*) to optimise the overall system performance.

A distributed query evaluation is offered in [19]. The underlying DHT overlay is extended with *multiSend()*, *index_query()* methods and a new "extended matching" message format to support *Single Query Chain (SQC)* and *Multiple Query Chain (MQC)* algorithms. In SQC and MQC for each conjunctive multi-predicate query a single query and multiple query chains are created, respectively, in order to distribute the processing load between the nodes. Each node matches the received triples against its local queries and sends intermediate results to the next node in the query processing node chain. The final notification to the subscriber node is generated by the last node in the chain. The MQC and SQC, however, significantly increase the overall network traffic which in a large-scal environment will undermine the performance of the system.

More recently, in CAPS [15, 20] a content-based publish/subscribe architecture is proposed for event dissemination infrastructure on top of DHT. The system also employs a *rendezvous* model to facilitate meetings between events and subscriptions. The selection of the *rendezvous nodes* is governed by the internal DHT routing protocol and hence, the chosen node would be known by other nodes in the network. CAPS does not extend the underlying generic DHT distributing algorithm and this makes CAPS adaptable to other DHT-based overlays.

The above approaches put their focus on scalability and efficient retrieval capabilities of a distributed content-based pub/sub system. Much less attention is given to the ultimate expressiveness of the queries. Our approach addresses this problem, which we feel is equally important to the effective operation of the applications that will use our infrastructure.

6 Conclusion and Future Work

We have described our implementation of a content-based publish/subscribe system based on the Rete algorithm. Our design touches on new and less explored fields of combining AI and Peer-to-Peer related research approaches. We use Rete, an efficient matching algorithm that is widely used in AI's production rule based expert systems. The constructed Rete network exploits the "temporal redundancy" of the previously matched information to provide an efficient matching of data (working elements) to predefined conditions. In our case, a Rete network is generated to manage subscriptions. The network keeps matches for specific subscription's condition and shares them with other subscriptions containing the same condition. The efficiency of the Rete algorithm comes from "matching only the changed data elements against the rules rather than repeatedly matching the rules against all the data " [21]. In addition, the conditions' "structural similarity" (i.e., the n-tuples) makes the Rete approach even more effective because many of the condition tuples share same attributes.

The main motivation behind our work is the Global Semantic Graph project [1], which is an Internet-scale persistent publish/subscribe system. The applications and services "publish" their internal state changes via simple *n-tuple* insertion while a "subscription" is essentially a standing query that keeps the internal state synchronised with any new information provided by others. In our future work we plan to examine approaches mentioned in Section 3.2 to improve the the overall system and utilise it as substrate for GSG's pub/sub facility.

References

[1] Shvartzshnaider, Y.: Global Semantic Graph as an Alternative Information and Collaboration Infrastructure. In: The 7th Extended Semantic Web Conf. PhD Symposium (2010)
[2] Forgy, C.: Rete: A fast algorithm for the many patterns/many objects match problem. Artificial Intelligence 19(1), 17–37 (1982)
[3] Fotiou, N., Polyzos, G.C., Trossen, D.: Illustrating a Publish-Subscribe internet architecture. In: Future Internet Architectures: New Trends in Service Architectures (2nd Euro-NF Workshop) (2009)
[4] Demmer, M., Fall, K., Koponen, T., Shenker, S.: Towards a modern communications api. In: Proc. of HotNets-VI (2007)
[5] Rosenblum, D.S., Wolf, A.L.: A design framework for internet-scale event observation and notification. ACM SIGSOFT Software Engineering Notes 22(6), 360 (1997)
[6] Felber, P.A., et al.: The many faces of Publish/Subscribe. ACM Computing Surveys 35(2), 114–131 (2003)
[7] Post, E.L.: Formal reductions of the general combinatorial decision problem. American journal of mathematics 65(2), 197–215 (1943)
[8] Davis, R., King, J.: An overview of production systems.plus 0.5em minus 0.4em Stanford Univ Ca Dept Of Computer Science (1975)
[9] Stoica, I., et al.: Chord: a scalable peer-to-peer lookup protocol for internet applications. IEEE/ACM Transactions on Networking (TON) 11(1), 32 (2003)

[10] Maymounkov, P., Mazieres, D.: Kademlia: A peer-to-peer information system based on the xor metric. In: Druschel, P., Kaashoek, M.F., Rowstron, A. (eds.) IPTPS 2002. LNCS, vol. 2429, p. 2. Springer, Heidelberg (2002)
[11] Rowstron, A., Druschel, P.: Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In: Guerraoui, R. (ed.) Middleware 2001. LNCS, vol. 2218, pp. 329–350. Springer, Heidelberg (2001)
[12] Lua, E.K., et al.: A survey and comparison of peer-to-peer overlay network schemes. IEEE Communications Surveys and Tutorials 7(2), 72–93 (2005)
[13] Cai, M., Frank, M., Chen, J., Szekely, P.: MAAN: a multi-attribute addressable network for grid information services. Journal of Grid Computing 2(1), 3–14 (2004)
[14] Cai, M., Frank, M.: RDFPeers: a scalable distributed RDF repository based on a structured peer-to-peer network. In: Proc. of the 13th Int. Conf. on World Wide Web, p. 657 (2004)
[15] Ahull, J.P., Lpez, P.G., Skarmeta, A.F.G.: LightPS: lightweight Content-Based Publish/Subscribe for Peer-to-Peer systems. In: 2008 Int. Conf. on Complex, Intelligent and Software Intensive Systems, Barcelona, Spain, pp. 342–347 (2008)
[16] Doorenbos, R.B.: Production matching for large learning systems. Ph.D. dissertation, Citeseer (1995)
[17] Sohn, A., Gaudiot, J.: Performance evaluation of the multiple root node approach to the rete pattern matcher for production systems. In: FGCS, pp. 977–984 (1992)
[18] Baldoni, R., Marchetti, C., Virgillito, A., Vitenberg, R.: Content-based publish-subscribe over structured overlay networks. In: Int. Conf. On Distributed Computing Systems, vol. 25, p. 437 (2005)
[19] Liarou, E., Idreos, S., Koubarakis, M.: Publish/Subscribe with RDF Data over Large Structured Overlay Networks. In: Moro, G., Bergamaschi, S., Joseph, S., Morin, J.-H., Ouksel, A.M. (eds.) DBISP2P 2005 and DBISP2P 2006. LNCS, vol. 4125, p. 135. Springer, Heidelberg (2007)
[20] Pujol-Ahullo, J., Garcia-Lopez, P., Gomez-Skarmeta, A.F.: Towards a lightweight content-based publish/subscribe services for peer-to-peer systems. Int. Journal of Grid and Utility Computing 1(3), 239–251 (2009)
[21] Banares, J.A., et al.: Taking advantages of temporal redundancy in high level petri nets implementations. In: Ajmone Marsan, M. (ed.) ICATPN 1993. LNCS, vol. 691, pp. 32–48. Springer, Heidelberg (1993)