A Policy Based Approach for Automated Topology Management of Peer To Peer Networks and a Prototype Implementation

Antonio Di Ferdinando
School of Computing Science
University of Newcastle
Newcastle Upon Tyne NE1 7RU
antonio.di-ferdinando@ncl.ac.uk

Paul Mckee
BTExact Technologies
Adastral Park
Ipswich IP5 3RE
paul.mckee@bt.com

Alessandro Amoroso
Dept of Computer Science
University of Bologna
Mura A. Zamboni, 40127 Bologna
amoroso@cs.unibo.it

Abstract

Peer-to-Peer (P2P) is a flexible architecture that let a network grow up in an arbitrary way by adding more and more peers providing resources to the whole system. If uncontrolled, however, this growth might lead to stability and reliability problems, due to the fact that any host might join the network, no matter whether it may provide guarantees or not. Another problem that might occur is the difficulty to administrate the network due to its possibly uncontrolled growth and its frequent topological changes. This paper focuses on the description of an approach to administration's automation based on the systematic use of policies. The goal is achieved by means of evaluation of the resources owned by each host, which addresses the problem of provision od reliable resources too. We believe this is a good solution to both problems. In this paper we will explain how our approach works and the benefits rising from its use. In order to better test our approach we developed and tested a working prototype of the system, also described in this paper.

1 Introduction

Peer to Peer (P2P) is a convenient and flexible way to share resources between hosts of typically different networks. The original aim of a P2P is to have more and more resources to share, and it is achieved by means of a continuous growth. An uncontrolled growth, perhaps, might lead to predictable problems. A network with virtually infinite growth possibilities could be hard or even impossible to manage, causing general stability problems. The frequent topological changes typical of a P2P network could lead to inconsistency problems. Moreover, the lack of guarantees on the resources brought by the participating hosts might represent a problem. In fact, typically there is no selection in the join phase on a P2P network, and a host with no resources can easily join the network. This could possibly lead to robustness and reliability problems.
This paper focuses on an approach to provide a higher degree of control in a P2P system. We make a systematic use of policies as a way for a runtime reconfiguration of the units forming the network. In order to fully test our ideas, we developed a prototype, named *NeToC (Network Topology Control*)[2] . Its goal is achieved by means of automation and evaluation of the nodes. The rest of the paper is structured as follows: section 2 describes the P2P system used to develop and test our prototype. Section 3 describes the structure of our system and the internals of *NeToC*. Section 4 briefly describes the dynamics of our approach. Section 5 shows the results of some test performed on *NeToC*. Section 6 plans some future work on *NeToC*, while section 7, finally, draws some conclusion.

2 Initial scenario

In order to exploit our ideas, we used *Hydra*, whose prototype implementation had been extended at *BTExact Technologies*. Hydra is a completely decentralized computation network whose fundamental unit is the *node*, defined as an autonomous entity which acts independently from the other nodes and decides everything related to a task to execute[1]. Structurally, an Hydra network is an acyclic graph, with the nodes representing fundamental computational units, and the connection edges representing the underlying communication network. A node can communicate by exchanging messages; a routing mechanism allows any node to communicate with only its neighbors. The only topological constraint present in the actual prototype specifies that each node can have up to three neighbors. Every subsequent join request is redirected to the node's smallest branch, allowing a balanced growth of the graph in each direction (Figure 1).
Hydra nodes are responsible for the distribution and the execution of the received tasks. When a node is asked

to execute a task, it checks its own resources. If they are enough, it executes the task locally, otherwise it distribute it to its neighbors. Hydra is subject to frequent topological changes, which represent both a *pro*, because of the scalability and flexibility of the topology, and a *con*, because of communication and management difficulties. Furthermore, it does not checks any guarantee on the nodes joining the network. It would be possible, for instance, to distribute a task to a “weak”, possibly unstable, node which will slow down the computation. In the case of a huge growth, furthermore, it could eventually be impossible to administrate or simply control an Hydra network, resulting in a possibly unstable and inconsistent network.

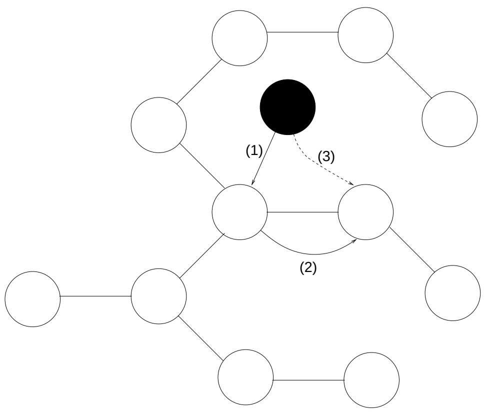

Figure 1. Typical Hydra network and redirection dynamics.

An approach to solve such problems consists in a mechanism able to evaluate each new node’s resources (guaranteeing resource sharing) and automate management (avoiding uncontrolled growth). Most P2P systems leave the node management to an *agent* running the session. In many of these contexts, an *agent* is an autonomous entity responsible to handle every task coming from the aggregation to the P2P network. This means that there is no concept of system management and, whereas present, it is static.
Our approach, on the contrary, aims to be dynamic by means of a runtime reconfiguration, triggered by the reception of a policy.
A policy is defined as a high level concept that is gradually refined into terms that relate more specifically to the actual infrastructure being managed[4, 5]. It can be seen as a constraint on how (part of) the system should work or the people can use the system. Many systems use policies o enhance trust within e-service frameworks[5], while others use them for QoS control on DiffServ and IntServ environments[6]. Our use of policies is meant to provide nodes with a reference the node can consult in determinate situations, and is structured as an XML subset, as seen in [3].

3 The *NeToC* solution

The solution we propose here to the aforementioned problems is the approach realized in *NeToC*. It provides for the construction of a *profile* of the resources (owned by the requesting node), to be matched against the ones required in the policy owned by the node receiving the join request. Logically, a carefully designed policy will require enough resources to contribute significantly to the whole system’s resources, and this will represent a guarantee on the resources that the joining node will share. The network management automation comes from the possibility to introduce in the network policies which will be adopted at runtime by the nodes, specifying the behavior of the nodes in determinate situations, specified in the policy as well.
The system could be divided in two main parts, the *Host Profile Subsystem* (HPS), and the *Network Policy Subsystem* (NPS). Both these parts take advantage of common auxiliary functionalities (like XML parsers) contained in another part of the system, outside the scope of the purposes of this paper. The HPS gains information on the local node and its point of view of the network, while the NPS is interested in policy management (therefore, in node’s behavior). HPS and NPS communicate by calling each other, and together form *NeToC*’s core system. They are coupled in a tight way and synchronized. Although the situations to manage are several, for the purposes of this work, and for the sake of brevity, we will take as an example the join scenario, since we believe it to be the most common situation to face. In our approach, we enclose a series of management directives in a policy object and distribute it throughout the whole network. The scenario to face, together with other information such as rules to apply and target nodes of the policy, are contained in the same policy object. The policies are injected in the network by the authority having the (previously checked) rights to do it. A node retains any policy that meets its characteristics, and retransmits all received policies, according to its routing rules. Since the target node of a policy is identified by its characteristics too, a policy injected in the network will possibly reach all the nodes that meet them, lightening the management efforts by handling the behavior of more than one node with just the systematic distribution of one policy. The distribution of a policy follows a very simple controlled flooding mechanism, starting from the local node. The control on such algorithms come from the fact that a node distributes the policy only to those neighbors that still have to receive it.
The HPS relies on the *meta data handler* entity and the node *profile* object, which contains an assessment of the set of resources owned by the node. The meta data handler is

mainly concerned with handling the profile, as well as the object exchange between nodes. It is responsible, therefore, for the creation and evaluation of the node profile and the I/O to/from neighbors. The profile object contains a minimal set of information about both hardware and software resources of the node. The NPS relies on the *policy* object which contains, apart from the aforementioned set of rules, information about the authority issuing it, the host sender and receiver, the rules the receivers must respect, and some topology information. Each host can adopt only one policy at a time. Policies may or may not expire, and can be replaced by other policies. An interesting feature of our approach, is that it is possible to introduce more than one policy in the network, since a policy object includes one or more separate target hosts. The possibility to specify such targets by an enumeration of hosts(IP addresses or names) or entire domains, makes possible and easy for a policy to reach a possibly wide number targets at the same time. Policies are mutually disjunct and does not influence other eventual policies circulating inside the network at the same time. This makes possible the contemporary circulation of more than one policy and the possible partition of the network in many dedicated subnetworks. Of course conflicts may happen, when a node is targeted from two contemporary policies. In such case the conflict is solved in the simplest way with the node adopting the last policy received.

4 *NeToC* in action

A new Hydra node can join an existing network or create a new network; in both the cases the aggregating process is automatically managed by the policy mechanism. As we mentioned above (Cfr. Section 2), if a node for some reason cannot fulfill the join request, redirects the request to its smallest branch. If a node is refused by *all* of the nodes in the network it meets along its redirection path it aborts. In the initialization phase, the new node evaluates its own resources and generates an XML object called *profile.xml*. Once created, the node asks to a specified Hydra node to join the network, by sending its profile. The Hydra node examines the profile of the new node and either accepts the connection by sending to the new node the current policy, or by refuses redirecting the connection request to its smallest branch. If the requesting node's profile meets the Hydra node's policy requirements, the connection can take place. The one mentioned is the classic example of scenario that is automatically handled by the system, without a direct user intervention. Since the rules governing the node's behavior are specified in the policy, and the policy is, in general, directed to more than one node, the same policy is useful to regulate the behavior of several nodes, automating node's behavior. By matching the resources specified in the profile object of the requesting node with the ones requested in the policy receiving node, we check whether the node is capable to share significant resources or not. By performing this check on *each* requesting node, we guarantee consistency and (possibly) robustness in terms of resources.

5 Tests

The tests performed were aimed to verify the respect of the specification and *NeToC*'s own performances, and have been conducted on a network provided by the University of Bologna composed by machines with heterogeneous configurations.
The respect of specification implies *NeToC* to have a predictable expected behavior when facing scenarios it has been designed for. The idea here is to build an Hydra network and introduce in it several different policies from the most general, with a few requirements, to the most particular, which may lead to a partitioned network. The latter represents the borderline case, and policies introduced aimed to build networks like, for instance, *data storage networks*, by including hosts with high storage facilities, *distributed calculus networks* by including hosts with very powerful and multiple CPUs as well as a large amount of transient space in the policy. All of the networks were correctly built, emphasizing a predictable behavior of *NeToC*, which resulted to be reliable.

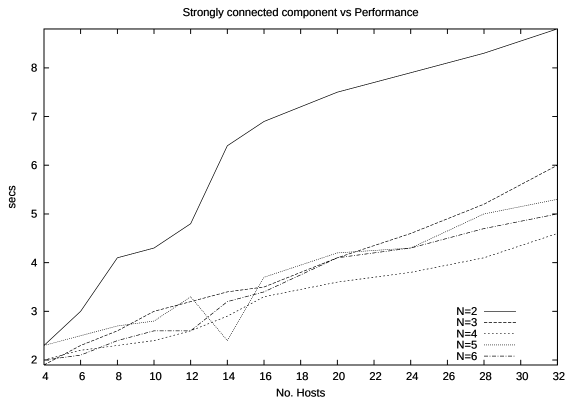

Figure 2. Performance comparison based on topological rules.

To investigate the performances of *NeToC* system, we examined mainly the efficiency of the distribution of the policy inside the network. Again, we built several Hydra networks but this time we differentiate them by varying the allowed number of neighbor for each node (named N). We let this parameter vary from 2 to 6. The results of this simulation is showed in Figure 2. The fact that the more N is higher, the more the policy distribution is faster is predictable and can be explained by saying that by increasing N we inherently increase the degree of parallelism of the network. What, instead, result interesting is the fact that for N=6 the line seems to be lightly swinging as you can note from Figure 3. This let us suppose some kind of “overload”, that could eventually lead the system to a “saturation point” in which the parallelism degree (and the performance) will start to decrease. This phenomenon could be explained by thinking that the policy distribution is a sequential operation, and a high number of nodes could slow it down, letting us argue that there will be a $N >> 6$ for which this process will start to became inefficient. From this comes the consideration that a high degree of parallelism does not necessarily means a high efficiency, and in our context a good value for N, i.e. the good number of neighbors, would be 3 (the default number) or 4. Obviously, this consideration must be related to the network we tested the prototype in, and could eventually be not true on a network made by computers with other equipment (for instance, multiprocessors). The performance tests showed, moreover, that the only factors able to slow down *NeToC* execution are external to *NeToC*, like the network latency of the passage through firewalls, and let us argue that *NeToc*’s integration in the Hydra environment is lightweight.

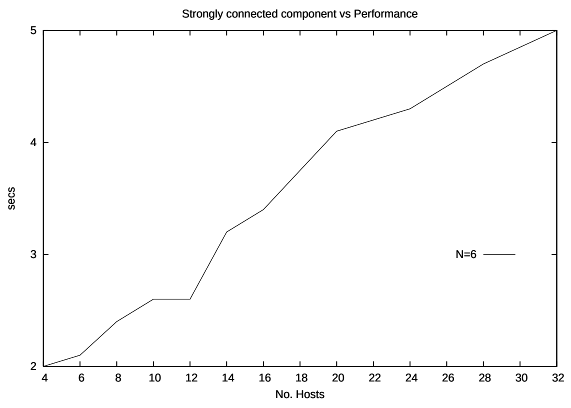

Figure 3. Swinging performance for N=6.

6 Future work

As stated in the abstract, *NeToC* is just a prototype. Therefore, many are the directions in which it can be improved. One should focus to an in depth study of other situation capable to be handled by means of policies. In particular, situations like leave (voluntary and not), rebuild (topological rebuilding) have been proven to be very important for the complete development of the prototype. Other directions could possibly focus on including Role Based Access Control (RBAC) techniques inside the policies to provide authority based access to design and distribution of a policy. Further improvements might involve an automatic peer discovery system (whose study has already began), and a graphic tool for policy design

7 Conclusions

We proposed an automated way to manage the topology of P2P networks, and presented a prototype implementation of our system, *NeToC*. The tests performed on it verified that it is possible to manage the topology of a highly scalable P2P network by means of policies distribution. We found out, moreover, an optimal topology structure which allowed us to take full advantage of this approach, letting the network grow in an arbitrary way but with some control on its growth and without slowing down the computation speed. Evaluation tests has proven the efficiency, correctness and robustness of *NeToC*, and we believe that the system is a promising instrument to the enlightenment of management efforts in P2P systems.

References

[1] P. Plaszczak,“Hydra: Decentralised Distributed Computing Environment. System Design and Prototype Implelentation”. MSc thesis, University of Mining and Metallurgy in Kraków, Poland, 2000.

[2] A. Di Ferdinando. “A policy based XML meta data system for a dynamic network topology management”, technical report, BTExact Technologies, July 2002.

[3] P. McKee and I. Marshall. “*Behavioural Specification Using XML”. British Telecom Technology Journal*, 2001.

[4] R. Wies,“Using a classification of Management Policies for Policy-Specification and Policy Transformations”, in *Proceedings of the IEEE/IFIP International Symposium on Integrated network Management*, Santa Barbara, CA, USA. 1995

[5] M. Cassasa Mont, A. Baldwin and C. Goh. “Role of Policies in a Distributed Framework”,in *Proceedings of POLICY 1999*, Bristol, UK, November 1999.

[6] R. Rajan, D. Verma and D. Kamat. “A Policy Framework for Integrated and Differendtated Services in the Internet”, in *IEEE Magazine*, September/October 1999.