A Hierarchical Architecture for a Distributed Management of P2P Networks and Services

Guillaume Doyen, Emmanuel Nataf, and Olivier Festor

The Madynes Research Team, LORIA
615 rue du Jardin Botanique
54602 Villers-lès-Nancy, France
{doyen,nataf,festor}@loria.fr

Abstract. We propose a management architecture for the P2P model which respects its distributed nature while building a hierarchical structure. This architecture enables the distribution of management functions, avoids an excessive centralization of the manager role and fits the dynamic of the P2P model well. The architecture is evaluated through an implementation in the Pastry framework.

1 Introduction

Nowadays, P2P networking is an emerging model that extends the limits of the client/server approach. Indeed, applications built on top of it present better scalability, load balancing and fault tolerance. Enterprises, administrations or universities are interested in the deployement of P2P applications for purposes like the distribution of networked file systems, including data replication mechanisms, or the use of distributed collaboration tools for projects that count remote participants. Network and service providers also see a good opportunity in supporting P2P applications with service level agreements. In this context, the need for a management framework for these services is obvious in order to ensure service levels for value-added applications.

The power of the P2P model relies on the distribution of all resources, knowledge and load. We believe that the management of a P2P community cannot be achieved in a centralized way mainly because such a centralization can potentially strongly affect the advantages brought by the P2P model. It does not make sense for a P2P community to have a central authority which manages all the peers: all the efforts done to increase the service level by the use of a distributed model will be impacted by the addition of a centralized framework, which actually owns the same goal of service operating improvement. This is why, in the same way peers act both as client and server, they have to act both as manager and agents for their management plane. Thus, the management of P2P services should be achieved through a P2P approach and, in this paper, we present a framework which takes the advantages of both the P2P model for management task distribution and the centralized management approach with the use of the standard manager and agent roles, to build a hierarchical management architecture for P2P networks and services. This architecture fits the

P2P model characteristics, which are decentralization, dynamic of peers naming and presence, heterogeneous nature of involved devices, and behavior of participants. Moreover, it presents interesting properties concerning the load-control of manager nodes, the structure balance and the choice of nodes for crucial points.

The paper is organized as follows: motivations are given in section 2. section 3 deals with the current research works that address the management of P2P networks and services. Work on tree-based infrastructures for P2P networks is also addressed in this section. Section 4 presents the objectives we want to reach through our proposal and the general algorithm we designed for the tree construction. The way we distribute this algorithm among peers is shown in section 5 and deployment aspects are treated in section 6. Finally, some conclusions and directions for future works are given in section 7.

2 Motivation

The proposal of a hierarchical distributed management model that is aligned with the underlying P2P framework it manages is driven by the following motivations:

resist to scale: P2P infrastructures involve a large number of components often spread among multiple administrative domains. Only self-management capabilities built in these complex infrastructures can provide scalable and efficient management;

master the dynamics: the individual components of a large P2P infrastructure are expected to be very dynamic (i.e. versatile presence and contribution to a service). Traditionnal management systems in which all participating components and ressources are known in advance and registered cannot be applied there.

Our approach is based on a partial integration of the management plane in the service plane of the P2P infrastructure. Such an integration avoids developers and service operators to have to deal with two different worlds (naming schemes, access protocols, security issues, ...). JMX is a good example of such successful integration in the Java world. Moreover, the merging of dedicated management signaling with the existing inftrastructure signaling potentially reduces the management overhead.

Our architecture is hierarchical since it has proven efficient for many monitoring operations, i.e. those based on monotonic functions (e.g. Sum, Min, Max, $Count$ and $Average$) [1]. It is also very well adapted to the dynamics of the underlying environment.

3 Related Work

3.1 P2P Management

Currently, a lot of applications, built over different protocols, allow users of a community to share files. Besides the fact that shared data are copyrighted, the

major problem content sharing applications have to face concerns the free riding which consists, for a peer, in the use of other peers' resources without providing any themselves [2]. This phenomenon clearly shows the need for a management framework able to ensure service levels. From this point, several proposals have emerged. They are service-embedded and use incentive approaches which rely on economic models [3]. For example, the MMAPPS (Market Management of Peer-to-peer Services) project proposes a cost evaluation for resources that depends on their availability, interest and quality [4].

Concerning performance management, [5] proposes to use an active network framework dedicated to Gnutella-like applications. First, it enables the scattering of a community into sub domains, thus limiting the scope of messages which rely on a flooding method. Then, messages routing adapts itself to the traffic load between peers. Thirdly, the virtual topology is adapted to the physical underlying network, which increases the global overlay performance. This work presents interesting results and is deployed over a Gnutella like infrastructure.

The major work concerning the deployment of a management infrastructure for P2P networks and services concerns the MMP[1] project of Jxta [6]. Jxta is a generic platform for the development of P2P services. From a functional perspective, Jxta provides an abstraction of basic P2P mechanisms like routing, lookup, organization or communication. It makes the development of services easier and allows their interoperability. The MMP project aims at providing a management infrastructure for Jxta communities. To do that, it provides an instrumentation of Jxta peers, a remote monitoring service and a management console application. The idea of this work is very interesting but actually, the instrumentation of Jxta peers is incomplete and the MMP project is now abandoned.

3.2 Our Previous Work

One of our goals is to design and deploy a management infrastructure which can be independent from the underlying services and which relies on standard approaches of network management. A first instrumentation experiment of a instant messaging P2P application clearly expressed the need for a management framework for such a class of application [7]. Then, we designed a generic management information model for P2P networks and services [8]. The latter enables a manager to build an abstract view of a P2P community, participating peers, shared resources and deployed services. We used CIM (Common Information Model) [9] as a formalism to express our model. In a second step, we refined our model towards the performance management of DHTs[2] [10]. As a case study, we considered Chord [11] and we defined a set of metrics which feature the performance of this DHT. Then, we integrated these metrics into our model. By this way, we enable a manager to evaluate the global performance of a Chord ring.

Our current work concerns the architectural aspects for the management of P2P networks and services. We are working on a proposal for a management

[1] Metering and Monitoring Project - meter.jxta.org
[2] Distributed Hash Tables

architecture that is compliant with the characteristics of the P2P model, and we present it in this paper.

3.3 Existing Overlay Tree Proposals

We propose to use a tree structure to enable a root manager to aggregate management information provided by sub-manager and agents in order to build an abstract view of a P2P community. Nevertheless, there are many other use cases where building a tree structure is required.

El-Ansary et al. [12] propose the building of a broadcast tree for structured P2P networks. Despites it presents interesting properties, their algorithm is strongly dependent on specific components of Chord [11].

From a theoretical perspective, our approach presents many similarities with [13]. The authors propose to use a binary tree structure to build a DHT. Their building principle and simulations provide very interesting results, such as the cost for node insertion or removal which falls from $log^2(N)$, in [11, 14] to $log(N)$. For management purpose however, the use of a binary tree is not the best choice, mainly because the tree is too thin and deep and this can be problematic, for instance, to propagate alarms from a leaf to the root.

Current work which our work is the most closest to is proposed in [15]. Its objective is to build an aggregation tree over any DHT that enables the computation of aggregation functions. The definition of a *Parent* function allows any node to establish a link towards its parent in the tree. Such a function has to be well chosen, so that it ensures a good tree balance. Our work is very similar, but it achieves a broader objective in the sense that if our management tree enables the computation of aggregation functions, in a more general way, it defines the roles of manager and agent for nodes.

4 Foundations and Principle

4.1 Goal

There are several objectives we want our management architecture to reach. These are:

Optimal manager role distribution: the P2P model is a distributed model where there is potentially no central point; each peer acts as both a client and a server. From a management perspective, we want to distribute the manager role among most peers so that they act as agent and manager.

Structure balance: To fit the distributed aspect of the P2P model, we want our tree to be well balanced so that a node cannot act as an excessive central point of failure and be stressed more than others.

Manager election: The more managers are close to the tree root, the more their role is crucial to achieve management functions. This is why we want to be able to choose managers according to any application context criteria such as the hardware resources or the user behavior.

Depth constraint: To ensure a minimum performance level of the management architecture, we want to control the tree depth so that a manager can contact any agent in a controlled number of hops as well as an agent, located as a leaf to contact the root manager.

4.2 General Tree Construction Principle

In order to build our hierarchical structure, we define the following axioms:

1. Each node is an agent and eventually a manager (at most once);
2. Each leaf represents an agent;
3. Each intermediate node, up to the root, represents a manager;
4. Each node owns an identifier which is the one given at the DHT level.
5. Each node owns a metric, called $Weight$, which represents the quality of the node for the manager role. This metric is based on any relevant criteria such as the hardware capabilities, the behavior or the participation level of the node. It used to choose the managers so that nodes with the highest weight are the managers of the highest levels in the hierarchy.

Then, we define that: *each manager of level L is responsible for nodes of level $L+1$ that present a common prefix of L digits*. Moreover, *managers are chosen through an election process*. The construction principle used here is very similar to the one proposed in [16].

Figure 1 shows a simple tree example applied to a Pastry-like DHT. One can see that each leaf represents an agent and appears zero or one time in upper levels where the managers are placed. Moreover, each manager owns a common prefix with its children that depends on its level. For example, node 001 located at level 2 presents the common prefix 00 with each of its agents that are 001, 002 and 003.

4.3 Formal definition

Our structure follows a formal definition that is expressed using a first order logic statement. Consider the following construction parameters:

B	The identifiers' base
D	The number of identifier's digits
N	The set of nodes in the community

Then, we define the following sets and variables:

d_i	The digit of rank i of an identifier with $0 \leq d_i < B$ and $1 \leq i \leq D$
$d_1 \dots d_D$	A node identifier
L	The number of digits of a prefix
λ	The set of all levels present, that is the set of all L
$Q_{d_1 \dots d_L}$	The set of nodes that own the prefix $d_1 \dots d_L$
P_L	The set of set of nodes $Q_{d_1 \dots d_L}$ which owns a common prefix of L digits
G	The set of manager nodes.

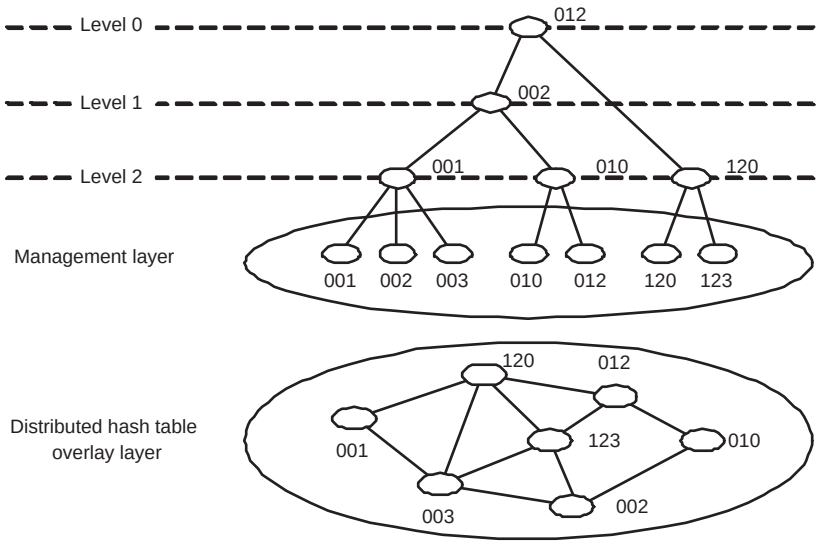


Fig. 1. Management tree example applied to a DHT

The statement below is always true with nodes that are involved in any management process. As we will see in the next section, only arriving nodes or temporarily disconnected nodes are not involved in management functions.

TREE DEFINITION ()
1 $P_{-1} \leftarrow \{\emptyset\}$
2 $\forall L \in \lambda$
3 $\forall n \in N$
4 $Q_{d_1 \ldots d_L} \leftarrow Q_{d_1 \ldots d_L} \cup \{n \mid n.\text{PREFIX}(L) = d_1 \ldots d_L\}$
5 $P_L \leftarrow \{Q_{d_1 \ldots d_L} \mid Q_{d_1 \ldots d_L} \neq \{\emptyset\}, 0 \leq d_i < B, 1 \leq i \leq L\}$
6 $\forall P \in P_L \setminus \{P_L \cap P_{L-1}\}$
7 $G \leftarrow G \cup \{n \mid n \in P, n \notin G, n.\text{WEIGHT}() = max(p.\text{WEIGHT}(), p \in P)\}$

where $n.Prefix(L)$ returns a list of L former digits from the node n identifier and $n.Weight()$ gives the node's quality according the metric defined in section 4.2.

In line 4, we build $Q_{d_1 \ldots d_L}$ the sets of nodes that own a common prefix of L digits. Then, in line 5 we gather all the non empty $Q_{d_1 \ldots d_L}$ sets in the P_L set of sets. Finally, in line 7, if needed, we elect a free manager which presents the highest weight. This way, we construct a tree that fulfills the goals presented in section 4.1.

5 Distribution of the Algorithm

We have designed two protocols for each event that can occur in the life of a P2P community. These events are: the arrival of a new node requiring an attachment

to the community and the departure of a node. Together with them, a regular maintenance process triggers structure update when the above mentioned events occur.

5.1 Node Insertion

The insertion process aims at adding a new node in the structure at its right location. It consists in looking for the manager that owns the longer prefix with the arriving node. In current DHTs, such an operation is not trivial since DHT functions do not enable semantic lookup. One solution could rely on a method of successive approaches: an arriving node looks for a node that owns a $D-1$ prefix, then a $D-2$ one, until it finds a manager. But, the method is costly in term of number of messages.

To overcome the above constraints, we propose to use the following method: when a node joins the tree, it generates a request with a random identifier. According to the DHT properties, a node with an identifier close to the required one responds. Then, the arriving node requests the manager of the latter node for its management. Thus, the new node is inserted, but the tree is not consistent regarding our formal description. This is not a problem because the new node will be involved, as either agent or manager, until it is correctly placed in the management architecture through the maintenance process.

5.2 Maintenance

The maintenance process is executed by manager nodes. It aims at maintaining the structure consistent. It is composed of two functions: the first consists in enforcing that the tree construction rules are effectively applied, and the second consists in verifying that referenced nodes are still alive.

This is why the maintenance process is executed in several contexts: (1) when a manager detects a new node insertion, to check that the arriving node is well located, and (2) at regular intervals, to check for any node departure.

The different operations executed by the maintenance process are:

1. **Presence checking:** For an L level manager, it consists in checking the presence of each of its children and its father. To do that, maintenance requests are sent to each child. Whenever a child doesn't respond, it is removed from the children list. Moreover whenever no maintenance request is received from its father, the manager is considered orphan and restarts the join process.
2. **Prefix checking:** For an L level manager, it consists in checking the children prefixes consistency. Two cases of reorganization are possible:
   - **Too short prefixes:** Whenever a child doesn't own a prefix of L digits with the considered manager, it transfers the child to its father;
   - **Longer prefixes:** Whenever two or more children share a prefix longer than L, two cases are possible:

- **Agent and manager children:** If agents and managers share a common prefix longer than L, then the manager with the highest weight will manage the other ones.
- **Identical children:** If children that share a longer prefix are exclusively a set of managers or agents, the child of higher weight will manage the other ones.

3. **Weight checking:** For an L level manager, it consists in checking that it does not reference any child, that is not a manager, and that owns a higher weight than its own weight. Each time this case occurs, the child of highest rank will take the place of the current manager and the latter loses its manager role.

5.3 Node Departure

When a node leaves the management structure and informs its father, all its children will be managed temporarily by the father, until the maintenance process reorganizes the structure. In case a node leaves the tree without informing its father, the maintenance process of neighbor nodes will detect its absence. The father will detect the absence of response from one of its children, and the children of the failing node will detect that they have not received any maintenance message for a given time. These orphan nodes will therefore use the insertion process to join the tree again.

6 Deployment

We have designed a prototype implementation of our architecture. It is built over the Java Free-Pastry[3] implementation of Pastry [14]. In this section, we first detail some implementation aspects. Then, we present the tests we have performed and the results they provide.

6.1 Node Architecture

The code we have deployed follows the functionnal architecture shown on Figure 2. Each Pastry node is composed of two different parts: an agent and a manager. The manager part is activated if the node endorses a manager role.

The agent part is composed of two main entities. The first one is a JMX MBean server. It hosts standard MBean objects coming from the instrumentation of Pastry presented below. The second entity of the agent part concerns core agent functionalities (requests processing, father soft state maintenance, ...).

The manager part is, as for the agent part, composed of two entities: the manager core and the maintenance process. The core entity is in charge of standard management functions of the tree, like requests forwarding or partial computations of a management function. The maintenance process is responsible for

[3] freepastry.rice.edu

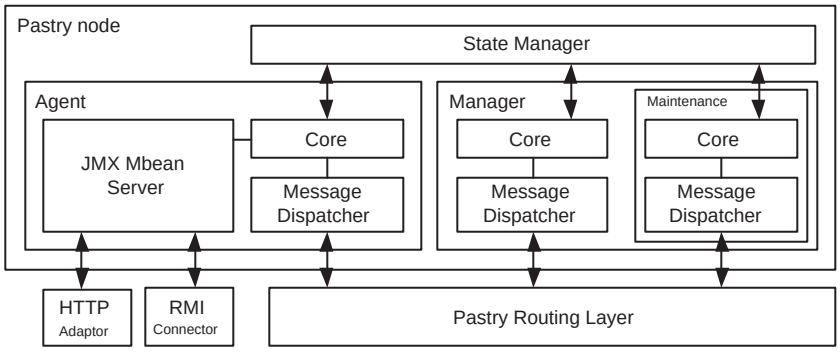


Fig. 2. Node architecture

ensuring the consistency of all meta-data stored in the state manager, represented in the upper part of Figure 2. This process periodically executes the operations described in section 5.2.

Concerning the communication, all the tree construction and maintenance related messages are exchanged through the Pastry routing layer. Management access to JMX MBean servers is achieved through a RMI as defined in JMX.

6.2 Node instrumentation

To design a manageable DHT community, we have instrumented Pastry nodes. Managed objects are CIM instances which follow the information model proposed in [10]; we collect information concerning routing tables, leaf sets and lookup and maintenance services. In fact, our management plane addresses two levels: a local one where managed objects stand for data related to their host node, and a global level, addressed by a manager (which can be centralized, hierarchical or distributed) which aggregates local information to provide an abstract global view of a community.

All the local and global managed objects are registered into a JMX MBean server as standard MBeans and we use RMI to enable the communication between these entities. To validate this instrumentation, we have designed a small application which draws a topological view of a managed Pastry community. We have chosen to use the leaf associations as a topological criterion because it respects the neighborhood semantic.

6.3 Evaluation

We present here the results of a small scale test that evaluates the construction cost of our management architecture according to the number of nodes. We have considered scenarios which involve from 1 to 20 nodes. Identifiers of nodes are set randomly using the Pastry factory. The nodes' weight, represented as a byte

value, is chosen randomly. Concerning timing aspects, the node arrival rate has been fixed to 1 node per minute. The maintenance process is executed every 15 seconds and a timeout for messages has been set to 30 seconds.

Fig. 6.3.a. Evaluation of the global tree construction cost

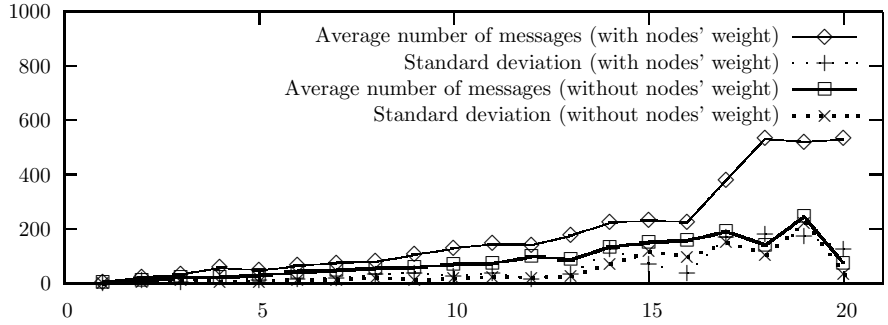
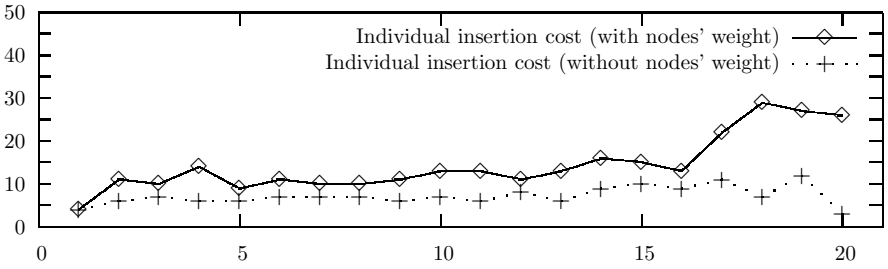


Fig. 6.3.b. Evaluation of the individual insertion cost



We did perform two tests. The first one considers the nodes' weight while the second one doesn't, i.e. step 3 of the maintenance process is executed in the first case only. Figure 6.3.a depicts the global construction cost. The metric we have considered is the number of messages exchanged between nodes to build the tree. For each of the two tests, we have represented the average value and the standard deviation. Figure 6.3.b represents the insertion cost for one node expressed in term of the number of messages exchanged. On this Figure, we have represented the average value of this metric for the two tests.

When considering the nodes' weight, one can see that from 1 to 16 nodes, the individual insertion cost is constant with a mean value of 12 messages per node. From 16 nodes, the insertion cost doubles. This phenomenon is due to the fact that statistically, up to 16 nodes with random identifiers, the tree contains only one level: a root manager in charge of agents; but from 16 nodes, the tree tends to present a second level; this is why the insertion cost increases. Then from

16 nodes, the standard deviation increases, because, the more nodes join the tree, the more different scenarios leading to different tree construction operating occur.

Concerning the second test, Figure 6.3.a and 6.3.b show that the tree construction cost is lower when the structure does not consider the nodes' weight. Moreover, the evolution of the construction cost is more regular than in the first test. Finally, one can remark that, in the latter case, the two cases which count 18 and 20 nodes are not statistically correct and show that the more nodes there are in the community, the more tests we have to perform to obtain meaningful results.

To conclude, this test shows that the consideration of a weight metric increases the construction cost strongly and may be removed to improve the performance of the tree structure.

7 Conclusion and Future Works

In this paper, we expressed the need for a management framework for P2P network and services; a management infrastructure is essential to enable a common use of the P2P model in sensitive value-added services. We proposed a hierarchical management architecture that fits the P2P model characteristics and that relies on the naming properties of peers. Our structure enables an strong distribution of the manager role, provides a balanced structure and a tree depth control. Moreover, the addition of a weight metric to peers ensure that critical places will be used by best participants. To implement our model, we have proposed a distributed algorithm which consists of three processes: insertion, removal and maintenance, responsible for enforcing the structure consistency.

Free Pastry was used to implement our model . We have instrumented nodes and integrated managed objects into a JMX MBean server. A prototype has been deployed and the tree structure building algorithm validated.

We plan to test our architecture in a cluster of five hundred nodes. In this context, we will be able to perform tests for communities containing up to 10000 virtual nodes and check the scalability of our proposal. Future tests will address (1) the tree resistance to nodes failures and (2) the way we can tune the prefix consideration to reach particular management objectives; for example, a management infrastructure which deals with fault management and alarm propagations requires a very short tree depth but the manager nodes will be strongly loaded. Besides this case, applications which want to spread management functions among peers will require a deep tree involving as most peers as possible.

References

1. Madden, S., Franklin, M.J., Hellerstein, J.M., Hong, W.: Tag: a tiny aggregation service for ad-hoc sensor networks. SIGOPS Operating System Review **36** (2002) 131–146

2. Saroiu, S., Gummadi, P.K., Gribble, S.: A measurement study of peer-to-peer file sharing systems. In: Proceedings of Multimedia Computing and Networking 2002 (MMCN '02), San Jose, CA, USA (2002)
3. Hellerstein, J.L.: A comparison of techniques for diagnosing performance problems in information systems (extended abstract). SIGMETRICS Perform. Eval. Rev. **22** (1994) 278–279
4. Antoniadis, P., Courcoubetis, C.: Market models for p2p content distribution. AP2PC'02, Bologna, Italy (2002)
5. deMeer, H., Tutschuku, K.: A performance management architecture for peer-to-peer services based on application-level active networks. In Stadler, R., Ulema, M., eds.: Networks operations and management symposium (NOMS 2002), IEEE (2002) 927–929
6. Oaks, S., Traversat, B., Gong, L.: Jxta in a nutshell. O'Reilly (2002)
7. Doyen, G., Festor, O., Nataf, E.: Management of peer-to-peer services applied to instant messaging. In Marshall, A., Agoulmine, N., eds.: Management of Multimedia Networks and Services. Number 2839 in LNCS (2003) 449–461 End-to-End Monitoring Workshop 2003 (E2EMON'03).
8. Doyen, G., Festor, O., Nataf, E.: A cim extension for peer-to-peer network and service management. In De Souza, J., Dini, P., eds.: Proceedings of the 11th International Conference on Telecommunication (ICT'2004). Number 3124 in LNCS (2004) 801–810
9. Bumpus, W., Sweitzer, J.W., Thompson, P., R., W.A., Williams, R.C.: Common Information Model. Wiley (2000)
10. Doyen, G., Nataf, E., Festor, O.: Performance management of distributed hash tables. In: To appear in The European summer School (EUNICE'2005). (2005)
11. Stoica, I., Morris, R., Karger, D., Kaashoek, M.F., Balakrishnan, H.: Chord: A scalable peer-to-peer lookup service for internet applications. In: Proceedings of the 2001 conference on applications, technologies, architectures, and protocols for computer communications, ACM Press (2001) 149–160
12. El-Ansary, S., Alima, L.O., Brand, P., Haridi, S.: Efficient broadcast in structured p2p networks. In: 2nd International Workshop on Peer-to-Peer Systems - IPTPS '2003. (2003) 304–314
13. Buccafurri, F., Lax, G.: Tls: A tree-based dht lookup service for highly dynamic networks. In: CoopIS, DOA, and ODBASE. Number 3290 in LNCS (2004) 563–580
14. Rowstron, A., Druschel, P.: Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In: IFIP/ACM International Conference on Distributed Systems Platforms (Middleware). (2001) 329–350
15. Li, J., Lim, D.Y.: A robust aggregation tree on distributed hash tables. In Sinha, V., Eisenstein, J., Sezgin, T.M., eds.: Proceedings of the 2004 Student Oxygen Workshop. (2004)
16. Plaxton, C.G., Rajaraman, R., W., R.A.: Accessing nearby copies of replicated objects in a distributed environment. In: ACM Symposium on Parallel Algorithms and Architectures. (1997) 311–320