# A Performance-oriented Management Information Model for the Chord Peer-to-peer Framework

Guillaume Doyen, Emmanuel Nataf, and Olivier Festor

The Madyne Research Team
LORIA, 615 rue du Jardin Botanique
54602 Villers-lès-Nancy, France
Guillaume.Doyen@loria.fr

**Abstract.** In this paper we propose an abstract model dedicated to the performance management of the Chord peer-to-peer framework. It captures information about a Chord community, the associated resources, the participating elements, as well as the global performance of the instantiated services. Our model enables the evaluation of the global health state of a Chord community and to act consequently.

## 1 Introduction

Chord [1] is a lookup protocol dedicated to Internet applications that need to discover any type of resources (files, CPU, services, ...) maintained by users that form a Chord community. It provides an elementary service: for a given key, Chord returns a node identifier that is responsible for hosting or locating the resource. Chord is deployed in several applications: CFS (Collaborative File System) [2] which is an Internet scale distributed file system, ConChord [3] which uses CFS to provide a distributed framework for the delivery of SDSI (Simple Distributed Security Infrastructure) security certificates and finally DDNS (Distributed DNS) [4] that proposes a P2P implementation of the DNS (Domain Name Service).

Chord offers performance levels in terms of load balancing, consistency and average path length for requests routing. In order to ensure such a performance, it executes dedicated services. However, the performance, as perceived by applications, highly relies on the conditions of the underlying network and the behavior the various and heterogeneous participants. To include these constraints in the evaluation, a performance measurement framework needs to be in place.

In this article, we propose a performance-oriented model for the Chord frameworks. We have chosen this particular P2P system among others (e.g. CAN, Pastry, Tapestry, D2B, Viceroy, ...) mainly because an implementation is available. In a general way, the information model we propose provides a manager with an abstract view of a Chord community, the participating nodes, the shared resources and the different instantiated services. Our model focuses on the performance aspect of the framework and, to reach this objective, we specify a set

of metrics that feature the Chord framework in terms of service performance, ring consistency and node's equity. This model is built on a previous work that provides a general information model for the management of P2P networks and services [5]. In addition to the embedded features of the Chord framework, having an abstract view of a Chord community, being able to evaluate its global performance, and having the possibility to react consequently, is a major step toward providing a P2P infrastructure aware of Quality of Service (QoS) constraints.

The paper is organized as follows: section 2 provides an overview of the P2P discipline and its management. Then, section 3 presents the Chord P2P framework and its properties. The definition of performance metrics and criteria is addressed in section 4 and the proposed information model is presented in section 5. A summary of the contribution is given in section 6 and future work is outlined.

## 2 Related Works

Peer-to-peer (P2P) networking is built on a distributed model where peers are software entities which both play the role of client and server. Today, the most famous application domain of this model concerns the file sharing with applications like E-mule, Napster, Gnutella and Kazaa among others. However, the P2P model also covers many additional domains [6] like distributed computing (Seti@Home [7], the Globus Project [8]), collaborative work (Groove, Magi) and instant messaging (JIM [9]). To provide common grounds to all these applications, some middleware infrastructures propose generic frameworks for the development of P2P services (Jxta, Anthill [10], FLAPPS).

While some applications use built-in incentives as a minimal self management feature [11], advanced management services are required for enterprise oriented P2P environments. The latter are the focus of our attention and deserve the availability of a generic management framework.

The first step toward this objective has consisted in designing a generic management information model for P2P networks and services that can be used by any management application as a primary abstraction. The work we have done in this direction has led to a model [5] that aims at providing a general management information model, that addresses the functional, topological and organizational aspects for such a type of application.

We have chosen CIM [12], [13] as the framework for the design of our generic P2P information model because of its richness in terms of classes covering several domains of computing that can be easily reused and extended.

The model we have designed covers several aspects of the P2P domain. First it deals with the notion of peer and its belonging to one or several communities. A particular association class allows the link of peers together in order to establish a virtual topology. One may note that, according to the context, different criteria can be considered to link two peers; for example, it can be based on knowledge, routing, interest or technological considerations. Then, our model features the available resources in a community and especially the ones

shared by its composing peers. We particularly address the fact that a resource can be spread in a community and thus (1) we differentiate owners and hosts of shared resources and (2) we split resources into physical (e.g. the chunk of a file on a file system) and logical ones (the aggregation of the different chunks). Moreover, these latter are consumed or provided in the context of services that is the fourth aspect of our model. Indeed, a P2P service is a basic functionality that is distributed among a set of participating peers; thus our model enables the global view of a service as well as the local one. Finally, we have identified particular basic services offered by any P2P framework; it concerns, for example, the way packets are routed in a P2P environment of an overlay type. We have thus modeled routing and forwarding services and the routing tables they generate or use.

In this way, our CIM extension for P2P networks and services provides an abstract view of a P2P network as well as the deployed services located in a manageable environment.

## 3 Chord

Chord is a framework that aims at providing routing and discovery mechanisms in a P2P environment. It is built on a ring topology in which nodes know their predecessor and successor. A consistent hash method generates a key for each node from its IP address. Then, each node is located in a ring in order to arrange keys in an increasing order. Each Chord node $n_i$ is responsible for the $]predecessor(n_i),n_i]$ range of keys.

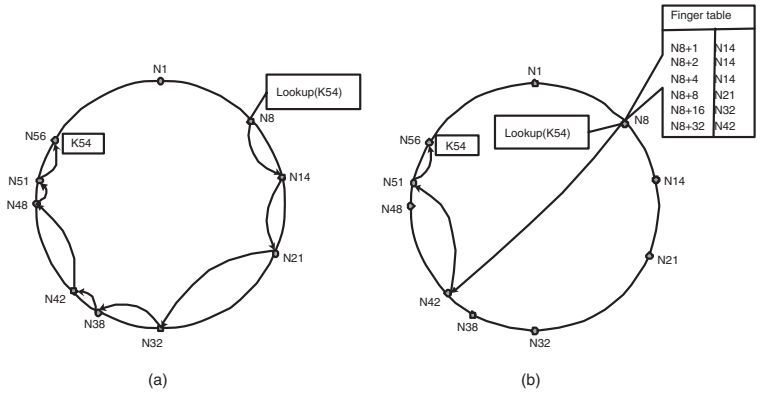


**Fig. 1.** *Extract from [1].*(a) Request routing by following the ring topology. (b) Request routing through *fingers*.

Consider a $N_T$ nodes ring. The routing tables maintained by each node $n_i$ will contain about $O((\log_2 N_T)^2)$ entries. Indeed, if the simple knowledge of

its predecessor and next node enables the construction of a ring topology, it has poor performance in terms of number of nodes that enter the routing of requests; Figure 1.a illustrates this kind of problem on a ring containing 10 nodes that uses a $[0, 64[$ range of addresses. One can see that a request for $k_{54}$ initiated by node $n_8$ will be routed in 8 hops.

In order to solve this problem, for a key domain comprised in the $[0, 2^m[$ interval, each node $n_i$ maintains a routing entry towards nodes, called *fingers* that own the $successor(n_i + 2^{k-1})$ identifier (with $1 \leq k \leq m$). Thus the number of node involved in a request routing is about $O(\log_2 N_T)$. Figure 1.b shows that with finger tables, the same request for key $k_{54}$ is routed in 3 hops.

## 4 Metrics for the Chord Performance Monitoring

Chord announces performance guarantees in agreement with its main core principles: the distribution of nodes in a ring topology, the use of a finger table, a consistent hash for nodes and keys, and the regular execution of a stabilization process. These concepts are claimed as providing an embedded way of ensuring a good service operation of the framework.

Nevertheless there is no way to monitor a Chord ring and to know the *health state* of such a P2P community. In order to allow a potential manager to evaluate the current Chord performance, we have defined several metrics. These are: the ring dynamics, the discovery performance, the node's equity and the ring consistency. By enabling a manager to monitor a Chord ring and to react consequently, we aim at providing a P2P framework that can effectively ensure QoS specifications on varying condition in the network.

To express our performance metrics, we introduce the following variables:

**$N_{MAX}$:** The upper bound for identifying nodes and keys. We assume the Chord hash method will generate identifier contained in the $[0, N_{MAX}[$ range. Moreover, all Chord operations are done modulo $N_{MAX}$;
**$n_i$:** a node with the $i$ identifier.
**$N$:** the set of nodes $n_i$ that are currently present in a ring, with $N = \{n_i\}$.
**$N_T$:** the total number of nodes in a Chord ring, with $N_T = Card(N)$.
**$k_i$:** a key with the $i$ identifier;
**$K_i$:** the number of keys the node $n_i$ is responsible for, with $K_i = Card(\{k_j\})$ with $id(pred(n_i)) \leq j \leq i$.
**$K_T$:** the total number of keys currently stored in the ring, with $K_T = \sum_{i \in N} K_i$.
**$\overline{K}$:** the average number of keys per node, with: $\overline{K} = \frac{K_T}{N_T}$

### 4.1 Measuring the Ring Dynamics

In order to make Chord performance measurements meaningful, we have to present them in their execution context. Indeed, due to dynamic phenomena, performance of Chord may vary strongly. Thus, we have defined metrics for measuring the ring stability.

The two elements that are submitted to dynamics are nodes and keys. Nodes can join and leave a ring in an unpredictable way while keys can dynamically be inserted or removed, or migrate in order to ensure their persistency. Thus for these two elements we have featured (1) their mean time of stability and (2) their change frequency.

Concerning the nodes, we assume that the arrival and departure times are known. For a particular node, all its joining and leaving operations will be stored in a log. Thus we can easily determine its mean time of presence in the ring.

Moreover, in a global way, we propose to evaluate the current arrival and departure frequency with the relations 1 and 2. Whenever a node joins or leaves the ring, the current time is first collected to update the $JoinFrequency$ and $LeaveFrequency$ indicators and then is stored in the corresponding last time variable.

$$JoinFrequency = \frac{1}{CurrentTime - LastArrivalTime} \tag{1}$$

$$LeaveFrequency = \frac{1}{CurrentTime - LastDepartureTime} \tag{2}$$

Finally, the global presence mean time is calculated by considering a temporal range $T$. The size of this range is determined according to the context[1]. For all the nodes, all the collected presence times that are contained in this range will be averaged in order to provide a global presence mean time of nodes. Equation 3 calculates it.

$$PresenceMeanTime = \frac{\sum_{i \in N} \sum_{t \in T_i} t}{\sum_{i \in N} Card(T_i)} \text{ with} \tag{3}$$

$T$**:** the considered range of time. $T = [T_{Begin}, T_{End}]$.
$T_i$**:** the set of presence time records for $n_i$ in range $T$.
$T_i = \{t \mid t = (T_{Departure} - T_{Arrival}); T_{Departure}, T_{Arrival} \in T\}$

Now concerning the keys, we have collected the same type of values, which allows us to determine the global $InsertionFrequency$, $MigrationFrequency$ and $RemovalFrequency$ of keys.

To conclude, the metrics defined in this section allow a manager to be aware of the dynamics of a Chord community. This evaluation is crucial because it allows all the following performance measurements to be analyzed in a particular context of dynamics, which gives them all the more sense.

### 4.2 Measuring the Discovery Performance

One of the core concepts of the Chord framework is the use of finger tables. It provides a good and stable performance on the average number of hops required to route discovery messages. Chord claims that this average is about $O(\log_2 N_T)$.

[1] it may be since the last hour, day, month, ...

We have chosen to monitor this crucial value for several reasons. First, it enable the manager to confirm that Chord respects the announced performance. Then, it is a meaningful indicator of the good state of the ring. Indeed, a significant increase of this value will indicate a problem of stability or consistency that must be due to important join or leave operations of nodes. Thus, for a manager, applying a threshold that can launch an alarm may be useful.

The way we propose to concretely evaluate this value is described in Equation 4. For each node $n_i$, we define the following metrics:

$NInitiatedRequests_i$**:** The total number of discovery requests that the considered node has initiated;

$NForwardedRequests_i$**:** The total number of requests that the considered node has received and forwarded to another node;

$NReceivedRequests_i$**:** The total number of requests destined to the considered node.

Then, we sum each of these metrics and deduce the real average number of hops per request. This value may be compared to the theoretical $\frac{1}{2}\log_2 N_T$.

$$AveragePathLength = \frac{\sum_{i \in N} NReceivedRequests_i + \sum_{i \in N} NForwardedRequests_i}{\sum_{i \in N} NInitiatedRequests_i} \tag{4}$$

By this way, we are able to monitor the performance of the discovery service that is the main function provided by the Chord framework and the only visible to a user.

### 4.3 Measuring the Nodes Equity

Contrary to the client/server model, the P2P one is a model where resources are distributed. This feature improves the load-balancing and enables the equity between all the participants. Moreover, it avoids any bottleneck that would concentrate the traffic and stand for central points of failure. In the case of Chord, ensuring a well balanced distribution of the keys among the different participating nodes is essential to guaranty a good performance of the system. In the opposite case, if some nodes host the major part of the keys, the Chord performance would collapse due to the too strong solicitation of these nodes; finally, the ring would tend to operate in a client/server way with nodes that are not dedicated to this task.

From this point, we have established a metric that can evaluate the distribution of the keys among the nodes. Chord claims that each node will be responsible for no more that $(1+\epsilon)\overline{K}$ keys and we propose to monitor this value. As shown in Equation 5, it consists in evaluating the variance of the key repartition in a ring. For each node $n_i$, we consider difference between the current hosted number of keys and the ideal average value and average this value on the whole ring.

$$NodesEquity_{\%} = 1 - \frac{1}{K_T} \sum_{i \in N} \lfloor \ | K_i - \overline{K} | \ \rfloor \tag{5}$$

The node's equity measurement is a useful indicator of the keys distribution in a Chord ring. This knowledge, added to the average number of keys per node and the effective number of keys a node hosts, indicates precisely any ring unbalance and we may imagine that, in this case, a manager could assign new nodes identifier and so redistribute the keys in order to improve the ring balance.

### 4.4 Measuring the Ring Consistency

The performance guarantees announced by Chord concerning the lookup service are respected if the ring is consistent. In case of an inconsistency, performance will collapse and the number of hops involved in request routing may increase from $O(\log_2 N_T)$ to $O(N_T)$. To monitor the ring consistency, we have used the constraints defined in [1]. These are:

- **Constraint 1** *Consistence of the immediate successor relations. This constraint is completed if, given the current presence of nodes in the ring, each node is effectively the predecessor of its successor.*
- **Constraint 2** *Consistence of the successors lists. Each node maintains a list of the k first successor in the ring. The constraint is completed if, given the current presence of nodes in the ring, (1) the lists maintained by each node effectively contain the immediate successors and (2) these successors are available.*
- **Constraint 3** *Consistence of the finger tables. This constraint is completed if, given the current presence of nodes in the ring, for each node $n_i$, a finger (1) matches the following relation successor$(n_i + 2^{k-1})$, where $1 \leq k \leq m$ and (2) is available.*

We consider a Chord ring is consistent if each of the above constraints are completed. In order to enable a manager to evaluate the consistency of a ring, each node must make information about its successor, its successor list and its finger table, available.

Concerning the management data, we have defined several elements to indicate the ring consistency. Locally, we have defined four boolean values that can inform the node of the consistency of its information. These boolean values will be evaluated by a manager and pushed in the MIB of the concerned node. Consider a node $n_i$. The first value, named $IsConsistent_i$, deals with the global consistency of the node; it is true if the constraints 1, 2 and 3 are completed. The next three, named $SuccessorIsConsistent_i$, $SuccessorListIsConsistent_i$ and $FingerTableIsConsistent_i$ inform about the respect of constraints 1, 2 and 3 respectively.

Now, considering the global consistency of a ring, we have defined the *IsGloballyConsistent* boolean value that indicates if the ring is consistent, with:

$$IsGloballyConsistent = \bigwedge_{i \in N} IsConsistent_i \tag{6}$$

Then, as shown in Relation 7, to have a more precise estimation of the ring consistency, we have defined a percentage value that indicates the consistency level. The $valueOf$ function returns 1 when the $IsConsistent_i$ value is set to true and 0 otherwise.

$$GloballyConsistencyLevel_{\%} = \frac{1}{N_T} \sum_{i \in N} valueOf(IsConsistent_i) \tag{7}$$

Lastly, in order to help a manager to locate a consistency problem, we have defined, for each node, a counter that references the number of times the considered node is badly referenced by others ones.

To conclude, the different local and global consistency indicators enable a manager to be aware of the good state of a ring. From this point, in case of inconsistency one may envisage several management actions like stopping the current forwarding of requests, forcing the execution of the stabilization process on some nodes that present obsolete knowledge and finally let the stopped request go on.

## 5 A Chord Information Model

Given our objective of Chord global model and the preceding performance measurement definitions, we propose a management model for Chord that is performance-oriented. It lies on a generic model for P2P networks and services that we have designed. The following sections present our Chord model and the way we have applied the preceding theoretical metrics to it.

### 5.1 Chord Node, Chord Ring and Virtual Topology

In the Chord framework, nodes are organized in an ordered ring. In this way, we consider that a Chord ring represents a community of peers. The model we propose is represented on Figure 2. In order to model a Chord ring, we have designed the ChordRing class that inherits from the Community class. The properties of the ChordRing are divided into two sets. the first one provides general information about the ring and the second one contains performance data, according to the metrics defined above.

Chord nodes are modeled through the ChordPeer class. This class inherits from the Peer one and defines several properties. The first set of properties contains general information about the node and the second one informs about the load of the node. It deals with the total number of keys hosted by a node, and the size of its routing informations expressed for the successor list and the finger table.

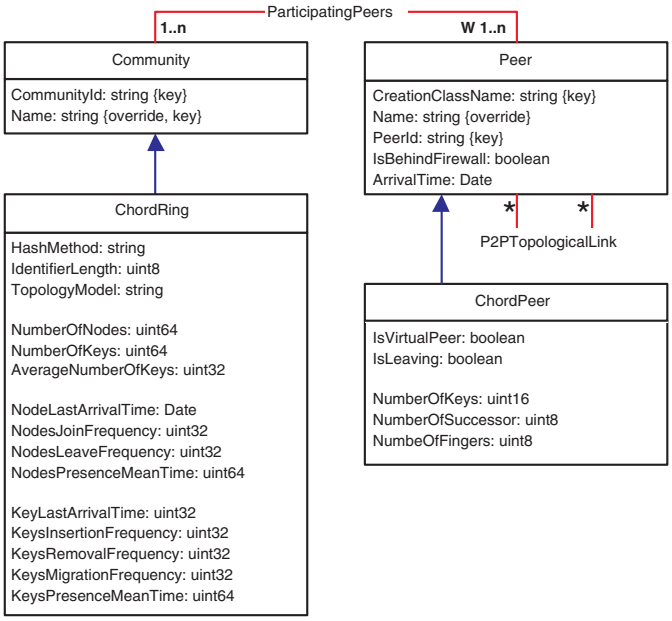


**Fig. 2.** The Chord Peer and community model

Concerning the topology, each Chord node knows its predecessor and its successor. Thus, by extending this mutual local knowledge to all the nodes of a ring, Chord establishes a virtual ring topology. In our model, we use the P2PTopologicalLink association class to represents these links. The Description attribute allows us to distinguish links toward a successor or toward a predecessor. We use this knowledge among others to estimate the ring consistency.

### 5.2 Keys and Resources

In the Chord framework, a node is responsible for a set of keys that represent resources. Nevertheless, the nature of the resources can be of several types. For example in CFS [2] it can be a file or a pointer toward a remote file. On the other hand in DDNS, a key will represent a DNS entry. This is why we have chosen to represent keys with the ChordKey abstract class that inherits from the PeerResource one. This way, we let an application build over Chord inheriting from it in order to represent any concrete resources. Figure 3 presents this specialization. The LastMigrationTime property is a local indicator of the key movement that is used to determine the global dynamics of a ring.

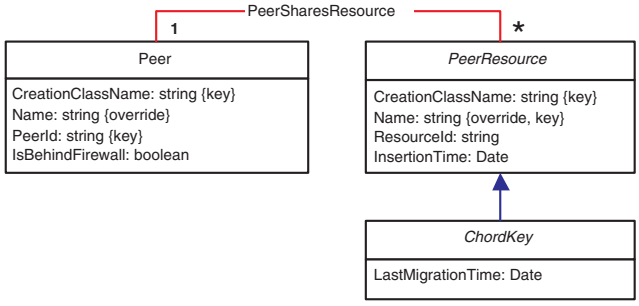


**Fig. 3.** The Chord resources model

### 5.3 Chord Services

The Chord framework is composed of two services. The first one is the lookup service. It represents the main functionality of Chord. The second one is the stabilization service. As described in [1] this process is executed in the background and checks the consistency of the ring.

**Chord Global Services** In the P2P model we have previously designed [5], we have defined a P2P service as being the aggregation of local instances. In this section, we address the Chord services in a global way.

The Chord service model is represented on Figure 4. The lookup and stabilization services are captured through the ChordLookupService and ChordStabilizationService classes. These classes inherits from the P2PService class of our P2P model.

First, the ChordLookupService class aims at providing information about the lookup service behavior. Thus, we have defined properties that match the metrics defined for the lookup service performance measurement. These are the total number of requests, the number of successful requests (that is another ring efficiency criterion) and the average path length involved in requests routing. Finally, the ring balance level provides a global estimation of the distribution of keys among nodes. These values are not directly accessible but are the result of the aggregation of nodes'local information.

Now, concerning the ChordStabilizationService we have just two properties to represent its global features. The IsConsistent boolean property stands for the global stability of the ring and the ConsistencyLevel deals with the average percentage of nodes that are in a consistent state.

**Chord Local Services** As explained above, P2P services are the result of local instances. This is why we have designed the LocalChordLookupService and LocalChordStabilizationService classes.

First, the LocalChordLookupService class contains several properties that are involved in the calculation properties of the ChordLookupService class properties. The NumberOfInitiatedRequest represents the number of lookup requests the local node has started. Then, the NumberOfForwardedRequests informs about the number of requests the local node has relayed. Finally, the NumberOfReceivedRequests informs the number of requests the local node has received as a destination. Finally, the NumberOfSuccessfulRequests represents the total number of requests the node has honored.

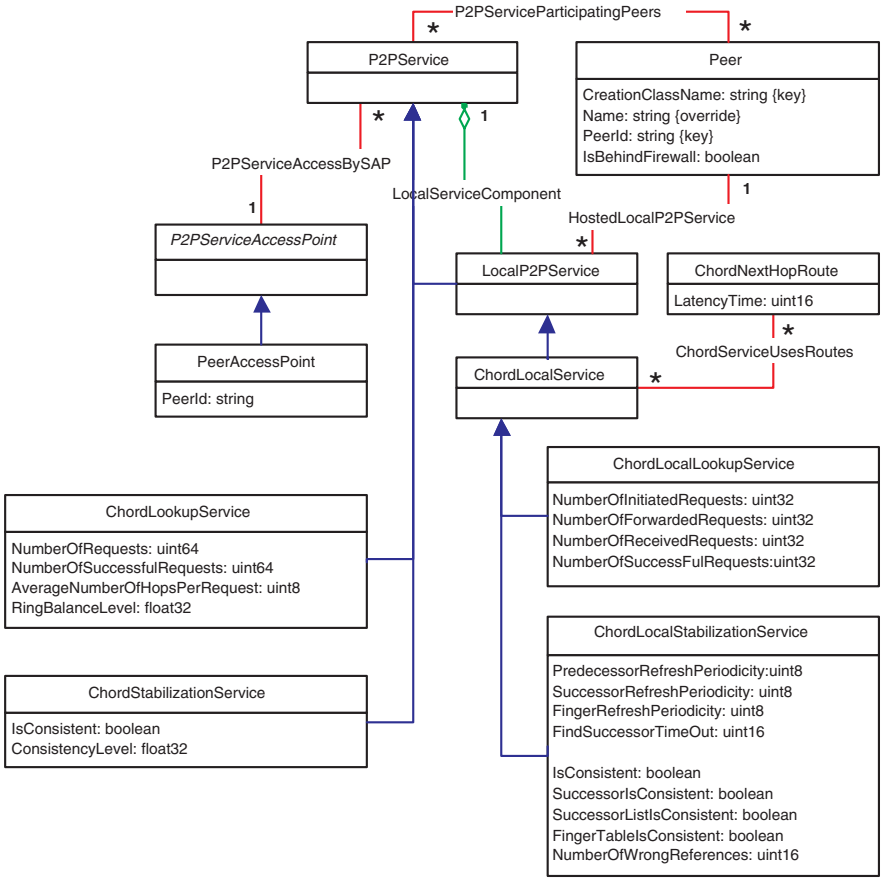


**Fig. 4.** The Chord service model

Then, the LocalChordStabilizationService class deals with the execution of the stabilization process on a particular node. Its attributes are split into two sets. The first set represents information collected by the considered node and pro-

vided to the manager. That is to say, the frequency of the predecessor, successor and finger refreshment. As for the second set of properties, it is computed and provisioned by the manager in the nodes. The NumberOfSuccessor property provides the size of the successor list. Then, the FindSuccessorTimeOut informs on the maximum time a node waits for a response from a node that can be a successor. This timeout is used in the stabilization process during the update of the successor list. Finally, the NumberOfWrongReferences deals with the number of times a node is badly referenced by other ones.

In addition to these services, we have modeled the routing tables of nodes. Nevertheless, this modeling aspect doesn't address the performance issue of the Chord framework and it is described in [5].

## 6 Conclusions and Future Works

Chord is a framework dedicated to the resources discovery in a P2P environment; given a key, Chord associates a node that is responsible for hosting or locating it. Its core principles lie on the use of a consistent and well balanced hash function for nodes and keys, a distribution of nodes on a ring topology, the use of finger tables and the regular execution of a stabilization process. Chord offers several performance guarantees in terms of average path length, load balance and information consistency. Nonetheless, in case of a real Chord deployment, additional behaviors influence the performance of the framework and a management infrastructure is necessary.

To effectively monitor the performance of a Chord community, we presented an abstract model that is performance-oriented. This work is based on a previous one directed towards the design of a generic information model for P2P networks and services. Globally, our model enables a manager to have an abstract view of a Chord community and to feature its operation performance. We have defined several metrics that evaluate the ring dynamics, consistency and balance as well as the lookup performance. Our goal is to provide a manager with tools that will help him to be aware of a Chord ring state and to react consequently. Our opinion is that these feedback data are essential for the deployment of the Chord framework in manageable P2P applications that can respect QoS agreements and deal with phenomena that cannot be captured by analytic models. Concerning the metrics we have defined, we consider them as basic indicators of a Chord community. Moreover, we assume that, in order to provide more advanced estimators, they can be combined and even enter the definition of policy rules. This way, given a Chord community, a manager could be able to act on participating elements in order to ensure service levels.

Our future works will first consist in deploying our model in a Chord implementation to (1) establish the validity of our theoretical model and (2) estimate the cost of our management infrastructure. To do that, we will first have to define a management architecture and a dedicated protocol for P2P networks, that is one of our current work. Indeed, we are thinking about the notion of manager in a P2P context, and the way peers collaborate to perform management

tasks. Then, in conjunction to this first direction, we are extending this work toward the design of a generic performance information model for DHTs. This model would aim at featuring the performance of existing DHT like D2B, CAN or Pastry among others.

## References


1. 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
2. Dabek, F., Kaashoek, M., Karger, D., Morris, R., Stoica, I.: Wide-area cooperative storage with CFS. In: Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP '01), Chateau Lake Louise, Banff, Canada (2001)
3. Ajmani, S., Clarke, D.E., Moh, C., Richman, S.: ConChord: Cooperative SDSI certificate storage and name resolution (2002) Presented at the International Workshop on Peer-to-Peer Systems.
4. Cox, R., Muthitacharoen, A., Morris, R.: Serving dns using chord. In: Proceedings of the 1st International Workshop on Peer-to-Peer Systems (IPTPS), Cambridge, MA (2002)
5. Doyen, G., Festor, O., Nataf, E.: A cim extension for peer-to-peer network and service management. (In: *to appear in* Proceedings of the 11th International Conference on Telecommunication (ICT'2004))
6. Oram, A., ed.: Peer-to-peer: Harnessing the Power of Disruptive Technologies. O'Reilly & Associates, Inc. (2001)
7. Anderson, D.: SETI@Home. Number 5. In: (in [6]) 67–76
8. Foster, I., Kesselman, C.: Globus: A metacomputing infrastructure toolkit. The International Journal of Supercomputer Applications and High Performance Computing **11** (1997) 115–128
9. 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).
10. Babaoglu, O., Meling, H., Montresor, A.: Anthill: A framework for the development of agent-based peer-to-peer systems. In: The 22th International Conference on Distributed Computing Systems (ICDCS '02), IEEE Computer Society (2002)
11. Mischke, J., Stiller, B.: Peer-to-peer overlay network management through agile. In Goldszmidt, G., Schönwälder, J., eds.: Integrated Network Management VIII, Kluwer Academic Publisher (2003) 337–350
12. Bumpus, W., Sweitzer, J.W., Thompson, P., R., W.A., Williams, R.C.: Common Information Model. Wiley (2000)
13. Distributed Management Task Force, Inc.: Common information model v2.7. (www.dmtf.org/standards/standard_cim.php)