17. Peer-to-Peer Search and Scalability

Burkhard Stiller (University of Zürich and ETH Zürich)
Jan Mischke (McKinsey & Company, Inc., Zürich)

17.1 Peer-to-Peer Search and Lookup in Overlay Networks

In large-scale Peer-to-Peer networks without any central server instances, lookup and search , *i.e.*, locating or finding objects by their unique name or a keyword description, respectively, require the collaboration of many nodes. Peers initiate and forward or route queries for objects along overlay links. Other peers who have information on how to locate the searched for objects send their answers via the same or different overlay links back to the initially requesting peer. Finally, the object itself is normally transferred to the requestor by directly using the underlying network protocol, thus, in most cases the Internet Protocol (IP).

The peer initiating a *search request* or *query* is called the *requestor*, a peer holding the requested object the *object owner*. A peer that merely has information on the location of an object, *i.e.*, a link to the object owner, is termed the *indexing node* or *indexer*. The process of receiving and forwarding search requests through the overlay network until it reaches the indexer or owner of the desired object is called *query routing*. Query routing strongly ties the process of searching in the distributed system, defined through a search protocol, to the structure of the underlying overlay network. A controlled communication for search usually involves the explicit design and maintenance of the overlay network.

Figure 17.1 illustrates the relation of Peer-to-Peer applications, search mechanisms, and overlay networks in a systems view. The overlay network builds on top of the network infrastructure that provides end-to-end connectivity between peers. It is usually designed such as to support the Peer-to-Peer search middleware which hides the distributed nature of object lookup and search from the Peer-to-Peer application. While search is a fundamental part of most Peer-to-Peer systems, further middleware functionality is, depending on the application, also required, like distributed accounting, reputation building, or group management. This other functionality can either build on the same overlay network constructed for search, or it can create and use its own, possibly better-suited overlay structure. Examples of such additional, beyond lookup functionality with specific overlay needs include application-level multicast [107] or fault resilient routing like RON (Resilient Overlay Network, [26]). This implies that several overlay structures may coexist in

R. Steinmetz and K. Wehrle (Eds.): P2P Systems and Applications, LNCS 3485, pp. 269-288, 2005.

the same Peer-to-Peer system, each serving a different purpose. Examples of current search overlay networks are given in the following section.

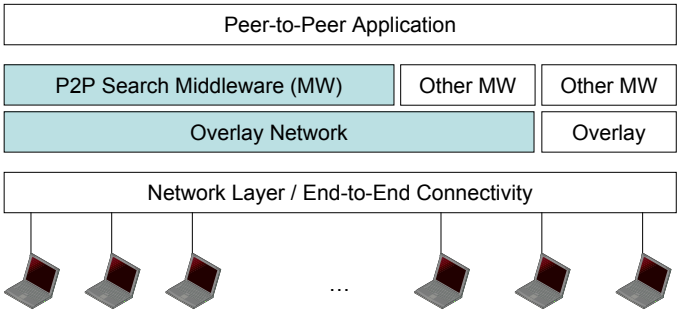


Fig. 17.1: Peer-to-Peer Networks – Systems View and Layering

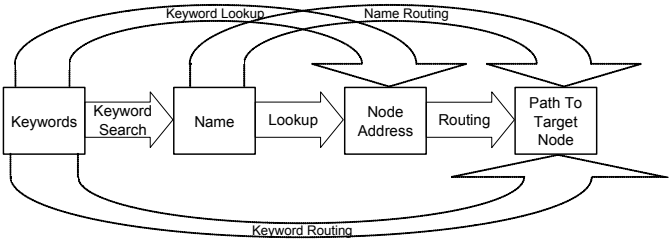
17.1.1 Problem Statement and Chapter Overview

The move toward large and completely decentralized Peer-to-Peer systems, handling millions of active nodes, imposes huge challenges on distributed search and overlay request routing. Many efforts have been undertaken to design and built middleware that overcomes the hurdles of decentralization, to construct suitable overlay networks, and to design Peer-to-Peer lookup and routing systems. However, a complete and clear structure including the delineation of these particular important and advanced designs is yet missing as well as a formalized comparative evaluation, except on an interesting example-driven basis [536], [52], and [276]. Furthermore, there is no clear statement as to which groups of design will show a viable approach in very large networks. Thus, this chapter and additionally [419] provide answers on how to (1) lay out the problem space for single-identifier lookup in Peer-to-Peer networks, (2) define a formal mathematical framework for scalability, and (3) to develop a highly advanced scheme – termed SHARK – for achieving good scalability in these cases.

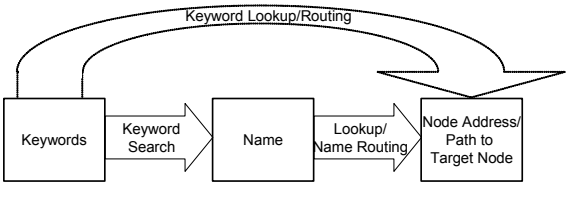
17.1.2 Search and Lookup – Functional Options

In much of the literature today, "search" refers to a wide range of operations on values stored in the network. This may encompass uni- or multidimensional search, full-text search, or aggregate operations. In contrast, "lookup"

refers to finding the node hosting data for a particular identifier. For this work, the canonical search process has been broken into different phases, where necessary steps and possible short-cuts have been identified. Figure 17.2(a) shows the process from *keywords* over *names* and *addresses* to the *path to target node* hosting the desired resources. Of course, for retrieving a document additional techniques may be applied, such as multiple keyword search or approximate keyword search, however, the full set of information retrieval techniques are limited in this section to Peer-to-Peer search and lookup.



(a) Search in Distributed Systems



(b) Search in Peer-to-Peer Systems

Fig. 17.2: Search in (a) Distributed and (b) Peer-to-Peer Systems

Usually, a user wants to specify what he is looking for in terms of *keywords*. In the simplest case, keywords are just one or more terms appearing in the desired content or describing the desired resource. More sophisticated approaches apply content/resource meta information based on attribute-value pairs, *e.g.*, the Resource Description Framework (RDF, [462]). *Keyword search* describes the functionality of mapping the resource meta information onto one, or, in the case of multiple matching resources, several unique *names* or identifiers in the network. Examples of such names are the Uniform Resource Locator, URL, or file names in a Unix file system. *Lookup* maps unique

names onto *addresses* in the network. Addresses specify the network location of the node hosting the resource with a given name, *e.g.*, the IP address of the host. Finally, *routing* is the process of finding a *path* and moving queries to the *target node*.

Three short-cut mechanisms can help optimize search. *Name routing* combines the (distributed) lookup of the target node address with path identification and query forwarding to that node. *Keyword lookup* returns one or more addresses of nodes hosting resources with given keyword descriptions. Napster is the most prominent example. Finally, *keyword routing* directly routes towards a node hosting specified resources. Keyword routing is sometimes also called *semantic routing* or *content routing*. For the Peer-to-Peer case, the process can be simplified as shown in Figure 17.2(b). Since Peer-to-Peer systems build on overlay networks, routing becomes a trivial task: knowing the target node address, the requestor simply creates a new virtual link to that address. Only few circumstances (like the anonymity requirement in Freenet [122]) lead to a more difficult overlay routing approach, which is an issue separate from search.

17.1.3 Design Space

Based on those initial discussions, the focus for this chapter is drawn on scalable keyword lookup for a single keyword. With the search process defined and disaggregated, it becomes obvious that searching requires a series of mappings, from the keyword space to the name space to the address space to the space of paths to nodes. The fundamental structural options in a distributed environment are the same for each mapping, and a complete classification and in our definition optimum design space is provided in Figure 17.3. The criteria of mutually exclusive and collectively exhaustive branches at each level have directly been built into the classification

A mapping can only be defined through a *computation* or a *table*. A (predefined) computation is difficult to achieve but some attempts have been made, usually involving hashing. More widely adopted are *tables* with (updatable) entries for the desired search items, *e.g.*, a node address for each valid name. Mapping then comes down to finding the desired table entry and looking up the associated value. In a distributed environment, a table can either reside on a *central* entity like a search engine server, or be fully *replicated* on each node, or be *distributed* among the nodes.

Distributed tables are most interesting and challenging in that they require for each mapping to collaboratively find and contact the node that offers the desired information or table entry. Two important aspects distinguish distributed table approaches: the *structure of the table, i.e.* the distribution of table entries to nodes, and the physical or overlay *topology* of the network. The distribution of table entries can happen at random or in a well-designed

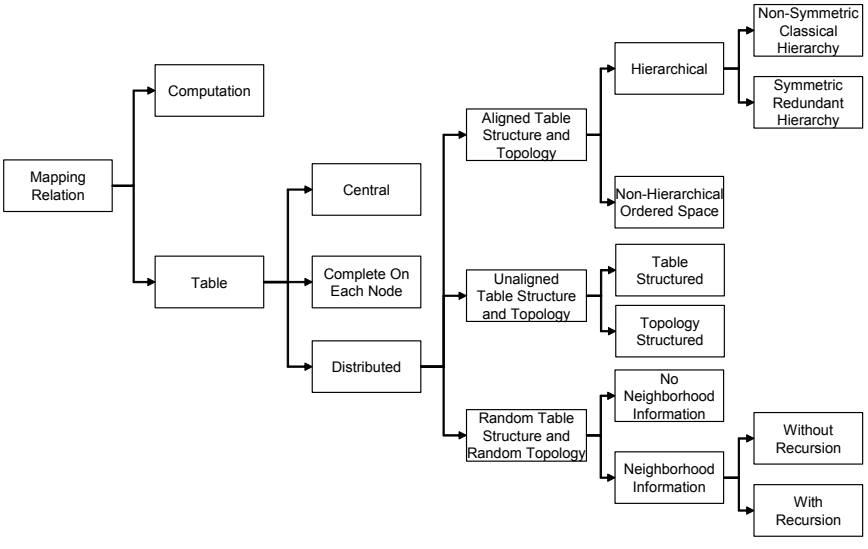


Fig. 17.3: Design Space for Mapping Relations in Distributed Systems

process leading to a clear target table structure; the same applies for the distribution of links and, hence, the topology. Whether the table structure and topology are designed and aligned, or both random or at least one of them designed but not aligned with the other has a substantial implication on search.

In a *random table structure and random topology*, it is natural that each node at least carries information about itself, *i.e.* its address, the names of the objects it hosts, and corresponding keyword descriptions. In addition to information on their own tables, nodes may have knowledge on the table entries of their *neighbors* in an aggregated or non-aggregated form. The knowledge on neighboring table entries will in some cases be restricted to the direct neighbors, but can also involve *recursion*: An arbitrary node A not only learns about the table entries of its neighbors B_i, but also through B_i about B_i 's neighbors C_{ij}, C_{ij} 's neighbors D_{ijk}, and so on. This way, nodes eventually know about most or even all keywords, names, or addresses in the direction of each neighbor in a usually aggregated way.

Rather than keeping explicit knowledge on neighboring table entries, nodes can exploit *implicit knowledge* when the table distribution and topology follow a clear and *aligned structure* that every node knows. The most common approach is certainly the *classical hierarchy*. A root node informs about table areas represented by a number of second-level nodes. The second-level nodes, in turn, delegate to third-level nodes for sub-areas within their own area, and so on, until a request finally reaches the leaf node responsible for

the desired entry. Particularly in the quest for scalable Peer-to-Peer search algorithms, "symmetric hierarchies" have been created by adding redundancy. In *symmetric redundant hierarchies*, every node can act as the root or be on any other level of the hierarchy. This can be achieved by replicating root information on table areas on each node as well as second-level information on sub-areas. Symmetric redundant hierarchies show structural similarities to k-ary n-cubes (cf. [535]).

Non-hierarchical structures are also possible and available. In an *ordered space*, the table is split into consecutive areas. Each of the areas is represented on one node. The nodes, in turn, are ordered in the same way, *i.e.* neighboring table areas reside on neighboring nodes. Examples of such spaces are rings or Euclidean spaces, but other forms are possible.

Unaligned table structures and topologies occur when the table is distributed according to a clear structure, but the topology is random, or the topology is designed, but the table structure random, or both table and topology are clearly structured, but in different ways. While the first case is helpful to allow for aggregation of table area information, the second case is advantageous for performance improvements compared to a completely random approach. It appears difficult to gain from the third case.

Designs based on any kind of structured table regardless of the topology are often referred to as *Distributed Hash Tables (DHT)*.

17.1.4 Overlay Topology Requirements

Peer-to-Peer systems of all kinds build on overlay networks. Nodes in Peer-to-Peer networks are required to have physical, link layer and network layer connectivity. For instance, in TCP/IP, every node is connected to the global Internet. It is, however, possible to contact every node via the corresponding network layer protocol in other networks as well, like X.25, or Novell IPX. In fact, it is sufficient to have a transport service that allows every node to send data to every other node. Theoretically, it would, *e.g.*, even be possible to use the Short Message Service (SMS) in GSM (Global System for Mobile Communications) or UMTS (Universal Mobile Telecommunications System) to build an overlay network of mobile nodes, provided appropriate software can be installed on the mobile devices.

The assumption of network layer connectivity allows Peer-to-Peer networks to abstract from the underlying networking infrastructure and its complexities and form an overlay network among themselves, which is discussed elsewhere in this book.

17.1.5 Overlay Topology Parameters

In general, network topologies can be characterized through their degree of symmetry, the network diameter, the bisection width, the average node degree, and the average wire length [313]. The functional and performance requirements determine the desired target characteristics:

- **Symmetry**
  Only symmetric topologies are appropriate for true Peer-to-Peer systems as only in this case all peers are equal from a topology point of view, thus forming the term symmetry with respect to peers' behavior. Consider a non-symmetric topology like the classic tree: It is obvious that the root of the tree has a far more central role than all leaves. At the same time, symmetry assists load-balancing. While, in non-symmetric networks, hot spots with high traffic load (the root in the tree) may exist, the load will balance over available connections in a symmetric network. Examples of symmetric topologies include rings, buses, hyper-cubes, complete meshes, cube-connected circles, or k-ary n-cubes. While symmetry appears to be one of the most basic requirements for a Peer-to-Peer topology, measurements as stated in [535] prove a huge heterogeneity among peer nodes in terms of their uptime, average session duration, bottleneck bandwidth, latency, and the number of services or files offered. Thus, it can make sense to explicitly design asymmetric overlay networks, where some peers adopt a server-like role.
- **Network Diameter (D)**
  The diameter of a network is defined by the number of hops required to connect from one peer to the most remote peer. It strongly influences latency and aggregate bandwidth for communication. In this context, hops in the overlay network have to be counted; actual hops in the underlying networking infrastructure will indirectly be considered through the wire length (cf. below). The average rather than the maximum number of hops between any two nodes in the overlay network is called *characteristic path-length* PL_{Ch} [462].
- **Bisection Width (β)**
  The number of connections from one part of the overlay network to the other part defines its bisection width. Assuming proper load balancing (which can be ensured through symmetry, at least partly), the maximum throughput of the network is proportional to the bisection width (and the average bandwidth of a connection). Even more importantly, there is a direct relation between bisection width and fault tolerance: the bisection width determines the number of links that have to break before the system goes down or, at least, operates only as two partial systems.
- **Node Degree (d)**
  The node degree is defined as the number of overlay links that each peer has to maintain. While a node degree higher than one is desirable for improved

fault tolerance of the network from the perspective of a single peer, the node degree can be a significant inhibitor for scalability: The node degree determines the size of the routing table on each peer with the according impact on memory consumption and processing power.

– **Wire Length ($\bar{\tau}$)**
The wire length is the average round trip delay of an overlay link, contributing to the latency in the system. The wire length is closely related to mapping an overlay network properly onto a physical network: A low wire length in a Peer-to-Peer overlay network can be achieved by choosing neighbors that are also neighbors or at least physically and topologically close in the underlying network. Closely related to wire length is the notion of stretch: The *stretch* of a path in an overlay network is the ratio of total physical network hops underlying the overlay path that separates two peers to the minimum number of physical network hops between the two when routing is not confined to the overlay network.

17.2 Scalability in Peer-to-Peer Systems

Scalability determines a key metric of distributed systems to describe in which sense this system is able to cope with many occurrences of an event. Thus, the mathematical model and formal definition of scalability is given for the non-Peer-to-Peer world and refined for Peer-to-Peer specifics in order to derive Peer-to-Peer-relevant metrics for scalability and efficiency in all relevant dimensions of scale.

Strict scalability of a system demands that its efficiency asymptotically remain constant as the system grows to large (or infinite) scale [313]. Production systems, for instance, often exhibit significant economies of scale, where efficiency even increases with scale, whereas communication systems tend to demonstrate disadvantages of scale due to the communication overhead involved. Therefore, most Peer-to-Peer systems are unlikely to be strictly scalable. It is thus sensible to introduce a more pragmatic definition and apply it throughout this thesis. *Scalability* is the asymptotic ratio of a system's efficiency to the efficiency of an idealized reference system. Assuming an efficiency of 1 for the reference, this is equal to the asymptotic efficiency behavior of the system. In contrast to scalability, *extensibility* refers to the possibility to grow a system step by step. That means, growing scale can be accommodated for through incremental addition of resources. This avoids the necessity to build a large system right in the beginning, dimensioned to cope with the ultimately expected scale.

It is important to understand that while Peer-to-Peer systems are usually highly extensible – with resources automatically being added along with joining peers – reasonable scalability properties are not ensured. For instance, the Gnutella network can grow simply by new nodes installing the servent and

contacting an arbitrary peer in the system, but query traffic can eventually lead to a collapse. In order to quantitatively assess the above measures, it is necessary to define suitable metrics, which determine a standard of measurement that can be applied to a corresponding dimension. This metric quantifies a dimension in that it associates each pair of elements of that dimension with a number or parameter reflecting the distance of these members along that dimension. The metric also defines the unit to measure the distance in.

17.2.1 Definition of Peer-to-Peer Scalability

Before providing a mathematical notation of scalability in a Peer-to-Peer context, this section introduces the notions of resources, efficiency, and scale that the scalability definition directly or indirectly builds upon. *Resources* are the replenishable goods used to perform a certain task or produce a certain product. Replenishable resources of a Peer-to-Peer system are processing power, memory/storage, bandwidth, time/latency. For efficiency and scalability considerations, consumption of these resources has to be separated into productive resource consumption and overhead resource consumption. Productive resource consumption of a Peer-to-Peer system is defined by reference to an idealized, overhead-free system that does not incur any distribution overhead:

- *Processing power*
  Productive is the processing power required on a single processor machine to perform a computation, where the result of the computation should be desired by the service user, otherwise it would be overhead. In Peer-to-Peer, a significant proportion of the processing power might be used for overhead like synchronization and routing in the distributed environment. Processing power is measured in terms of the number of operations on a reference processing unit.
- *Memory/storage*
  Productive is the memory needed for intermediate computation results or content or application storage as demonstrated in a non-distributed environment. In Peer-to-Peer, storage of, *e.g.*, routing tables on a peer node may constitute to the system overhead. Memory/storage is measured in KByte.
- *Bandwidth*
  The bandwidth needed to transmit the desired results of a service, *i.e.*, computation results or content, is productive. There may be communication overhead for discovering other peers, for instance. Bandwidth is measured in KBit per second (Kbit/s).
- *Time/latency*
  Particularly from a service user perspective, time is an important good and waiting times or latency can not only be annoying but also lead to

system malfunctions. Significant latency can result from several hops onto different peer nodes. No latency is productive, all is overhead. Latency is measured in milliseconds (ms).

All resource consumption of a Peer-to-Peer system that is not defined as productive above is considered overhead. More specifically, the overhead comprises all protocols and functionality of the distributed architecture that are necessary to operate the system and to give it certain properties. In a mathematical notation, total resource consumption can be represented by a vector

$$\vec{p}_{tot} = \vec{p}_{prod} + \vec{p}_{OH}; \qquad \vec{p} = \begin{bmatrix} \text{ProcessingPower} \\ \text{Memory} \\ \text{Bandwidth} \\ \text{Latency} \end{bmatrix}$$

including straightforward notations for total (tot), productive (prod), and overhead (OH) resources. Note that resource consumption is a function of scale $\vec{\sigma}$, where the elements of the scale vector denote those different dimensions of scale.

17.2.2 Efficiency and Scale

Efficiency and Scale Efficiency is the ratio of productive resource consumption and total resource consumption. It describes the level of optimization of an approach to perform a task or produce a product with respect to its resource consumption. An efficiency of 1 indicates a perfect approach in that it does not incur any overhead resource consumption. Due to the multi-dimensionality of the resource vector, the ratio of productive to total resource consumption has to be represented as a multiplicative equation:

$$\vec{p}_{prod} = \varepsilon \cdot \vec{p}_{tot}; \qquad \varepsilon = Diag \begin{bmatrix} \text{ProcessingPowerEfficiency} \\ \text{MemoryEfficiency} \\ \text{BandwidthEfficiency} \\ \text{LatencyEfficiency} \end{bmatrix},$$

where Diag() creates a diagonal matrix from a vector, *i.e.*, a matrix with all zeroes except for the diagonal elements. Like the resource consumption, the efficiency heavily depends on the scale $\vec{\sigma}$.

Scale is the size and frequency of tasks or the system performing the tasks along possibly multiple dimensions. In light of the variety of possible dimensions scale can refer to, it is necessary to identify a set of requirements that

a proper choice of scale dimensions should fulfill. With an eye to scalability, suitable *scale metrics* have to be mutually exclusive, collectively exhaustive, and relevant:

- *Mutually exclusive:*
  Different dimensions of scale chosen must not overlap. For instance, the number of peers and the number of files per peer are mutually exclusive, whereas the total number of files in the system as a third dimension could be derived from the previous two, resulting in an overlap that is to be avoided.
- *Collectively exhaustive:*
  The set of dimensions chosen has to describe all aspects in terms of size of the system that matter. For instance, the number of peers alone are not sufficient to derive storage requirements for all objects in the system.
- *Relevant:*
  The term "that matter" above already indicates that only those dimensions are to be introduced that are relevant for the specific consideration. For scalability research in particular, in order to be relevant, the selected dimensions have to be (a) *key drivers of efficiency* of the system in consideration: As scalability is defined through the efficiency depending on scale, only scale dimensions that actually have an influence on efficiency are relevant; and (b) *subject to growth*: Only those dimensions that are expected to grow are obviously interesting for scalability. While the number of peers in a system is usually growing, this may, *e.g.*, not be the case in a specific closed-group scenario.

In the sense of the definition of scale above, a task in a Peer-to-Peer system could be, *e.g.*, a file download, a computation, or any combination of these or other simple tasks. The number and the average size of tasks obviously constitute the most high-level scale dimensions for a Peer-to-Peer system. A multiplicative disaggregation automatically ensures that the set of dimensions is mutually exclusive and collectively exhaustive. This yields:

- The size of a task as its productive resource consumption in terms of processing power, memory/storage, and bandwidth.
- The number of tasks that can be disaggregated into the multiplicative components: (a) number of potential peers; (b) average percentage of peer nodes active, *i.e.*, online and responding to Peer-to-Peer messages; (c) average number of objects per peer, *e.g.*, number of services or content files or information items; (d) request frequency, *i.e.*, number of tasks per online peer node, object, and unit of time.

The most important dimensions are the number of peer nodes and the number of objects, as they are set for the fastest growth. While these scale dimensions probably cover most aspects of efficiency, some of the overhead functionality might have additional scale dimensions as key drivers for resource consumption. Even though a detailed evaluation is only possible for a

concrete system, one can hypothetically expect a couple of scale dimensions in addition to the ones above. The scale vector can now be defined as

$$\vec{\sigma} = \begin{bmatrix} \#\_\text{peerrs} \\ \%\_\text{online} \\ \text{objects\_per\_peer} \\ \text{task\_frequency} \\ \text{processing\_size\_of\_task} \\ \text{memory\_size\_of\_task} \\ \text{bandwidth\_size\_of\_task} \\ \dots \end{bmatrix},$$

where these dots symbolize additional scale dimensions that may be required for specific systems.

In summary, mechanisms for managing the (resources of the) Peer-to-Peer network include the Peer-to-Peer overlay network management, driven by the rate of joins to and departures from overlay Peer-to-Peer network, and a QoS control. Mechanisms for offering and retrieving services in a distributed environment address (a) a service description and classification driven by the variety and complexity of services, (b) lookup and search driven by users' total queries per successful query/task, (c) pricing, indexing, and advertising, (d) negotiations and the percentage of negotiated tasks, and (e) contracting driven by the percentage contracts per task. Mechanisms for fulfilling a service cover accounting and charging and invoicing, a.o. driven by the number of clearing or payment authorities. The set of organizational and self-learning mechanisms for peers includes the building and maintenance of peer groups, affected by the diversity of interests of peers, and the reputation of peers, effect by the frequency of reputation updates. Finally, security mechanisms, such as the identification, authentication, authorization, encryption, and decryption, are driven by the percentage of tasks requiring the respective security mechanism.

17.2.3 Scalability Metric and Notation

Building on these notations for scale and efficiency, scalability can now be captured mathematically. Strict scalability, the asymptotically constant efficiency requirement, can be translated into

$$\varepsilon = \varepsilon\left(\vec{\sigma}'\right) \rightarrow const; \vec{} \sigma \quad = \infty$$

where ' denotes the matrix/vector transposition. For the non-strict scalability definition, the scalability matrix Σ is defined by

$$\varepsilon_{\text{System}}\left(\vec{\sigma}'\right) = \sum\left(\vec{\sigma}'\right) \cdot \varepsilon_{\text{Reference}}\left(\vec{\sigma}'\right)$$

where the scalability matrix is required to be diagonal. The structure of the matrix is:

$$\sum = \begin{bmatrix} \Sigma_1(\sigma_1, \sigma_2, ...) & 0 & 0 & 0 \\ 0 & \Sigma_2(\sigma_1, \sigma_2, ...) & 0 & 0 \\ 0 & 0 & \Sigma_3(\sigma_1, \sigma_2, ...) & 0 \\ 0 & 0 & 0 & \Sigma_4(\sigma_1, \sigma_2, ...) \end{bmatrix}$$

All elements of the diagonal refer to the corresponding resource in the efficiency matrix, *i.e.*, Σ_1 is the processing-power scalability, Σ_2 is the memory scalability, Σ_3 is the bandwidth scalability, Σ_4 is the latency scalability. Each of these scalability metrics depends on all or a subset of the scale dimensions $\sigma_i, i = 1, 2, ...$

17.3 An Assessment Scheme for Peer-to-Peer Lookup and Search Overlay Network Scalability

The scalability evaluation can be simplified. Rather than by determining the efficiency as a function of scale and then deriving the scalability matrix as defined above, it is possible to investigate scalability by only calculating the overhead for a certain functionality as a function of scale and determining its behavior in a growing system. This allows a straightforward assessment whether or not the overhead grows out of bounds. Furthermore, it does not preclude a more formal determination of percentage efficiency values as described above as a subsequent step.

Usually, a logarithmic increase of overhead resource consumption with scale is regarded as the maximum tolerable. Care is due, though: even a linear increase may be alright if the absolute amount of overhead at maximum expected system size is still low, and, vice versa, even logarithmic or sub-logarithmic increase may be fatal when the starting value is already very close to bearable limits. For illustration, Table 1 gives an idea of expected system sizes and most important resource constraints for different Peer-to-Peer applications.

This section focuses on overhead for lookup and search – for two reasons. First, lookup and search with their complex n-to-n relationship of peers appear most vulnerable to network growth, with the number of nodes that possibly need to get involved growing linearly in n and the number of possible peer relationships even growing with the factorial of n. Second, the assessment scheme may be applied to other schemes and algorithms to allow for an objective comparison.

System	Expected Scale (# Peers)	Resource Constraints
File Sharing Peer-to-Peer Trading	$10,000,000$	Bandwidth: $56 \ldots 2,000$ Kbit/s Latency: < 7 s
Corporate Backup System	$30,000$	Storage: $1 \ldots 10$ GB Processing Power: 5% of 1-2 GHz
Peer-to-Peer Special Interest News	$1,000$	Bandwidth: $56 \ldots 2,000$ Kbit/s Latency: < 7 s
Mobile Collaboration Groupware	50	Bandwidth: $9.6 \ldots 384$ Kbit/s

Table 17.1: Peer-to-Peer System Sizes and Resource Constraints (Illustrative)

17.3.1 Overhead for Lookup and Search

The overhead resource consumption for lookup and search is a function of scale

$$\vec{p}_{OH,\text{search}}\left(\vec{\sigma}'\right) = \vec{p}_{OH}(\text{task\_size}, \text{task\_freq}, \text{queries\_per\_task}, \#\text{\_peers}, \%\text{\_online}, \text{objects\_per\_peer})$$

applying the scale vector defined in Section 17.2.1 and the additional dimension queries_per_task that was identified as being relevant for search task_size stands for bandwidth/memory/processing_size_of_task, respectively.

This expression can be simplified. First of all, task_size is only relevant for productive resource consumption to accomplish tasks, not for overhead. Second, #_peers and %_online will only appear in product form, so it is possible to aggregate. Similarly, it is possible to aggregate the query frequency: query_freq = task_freq · queries_per_task. Finally, search functionality can be separated into query (routing and processing) and overlay network management. The query overhead will be directly proportional to query_freq, while the overlay network management will be independent of it. Abbreviating

$$|o| = \text{objects\_per\_peer}$$

yields

$$\vec{p}_{\text{OH,search}}\left(\vec{\sigma}'\right) = \vec{p}_{\text{query}}\left(n, |o|\right) \cdot \text{query\_freq} + \vec{p}_{\text{OVLmgmt}}\left(n, |o|\right)$$

17.3.2 Dimensions of Lookup and Search Overhead and Quantitative Drivers

Resource consumption comprises the dimensions bandwidth, latency, processing power, and memory/storage. The most crucial scalability dimensions are the ones that refer to the most critical resources. More specifically, the scalability consideration should be focused on the resources that the system runs out of first or that are most expensive. As current utilization of processing power on PCs is usually around 4% [90], and storage is getting cheaper and cheaper, the most critical resources in a Peer-to-Peer system are most likely bandwidth and latency. Bandwidth is particularly scarce as many 'last hop' rather than backbone connections are involved in a Peer-to-Peer network. The following paragraphs will take a closer look at the drivers of bandwidth and latency overhead for queries before giving some intuition on memory and processing power overhead for queries as well as on overlay network management overhead in general.

The *request path-length* PLR is the number of hops that a lookup or search request makes on a Peer-to-Peer overlay network. Depending on the context, it may be a random variable or denote an average.

The *pruning factor* $f_p = PL_R/PL_{CH}$ denotes the average percentage of the characteristic path-length that a request needs to travel before being pruned off. The pruning factor can be calculated from the pruning probability at each hop $p_{p,i}$ (*i.e.*, the probability that the requested object is found at that hop) through

$$fp = \frac{1}{PL_{CH}} \sum_{i=1}^{PL_{CH}} i \cdot \prod_{k=0}^{i-1} (1 - p_p, k); i, k \in \aleph$$

The pruning probability $p_{p,0}$ at node 0, the requesting node, will usually be zero.

The *latency* L for a query is driven by the characteristic path-length PL_{Ch} and the wire length $\bar{\tau}$ as well as the pruning factor f_p:

$$L = PL_R \cdot \bar{\tau} = PL_{Ch} \cdot f_p \cdot \bar{\tau}$$

Routing efficiency ε_{route} is defined as

$$\varepsilon_{\text{route}} = \frac{d - x}{d - 1},$$

where x denotes the number of nodes that a query is forwarded to at any hop. The routing efficiency is defined to be 1 if only one node has to be contacted at each hop and 0 if all d neighbors have to be contacted. In that sense, Gnutella with its flooding approach has a routing efficiency of 0, whereas consistent hashing algorithms like Chord [575] have a routing efficiency of 1.

With the aggregate number of messages sent in the network to resolve a query, m_{agg} and the average size of a query message message_size, the aggregate data transmission B becomes = message_size $\cdot$ m$_{\mathrm{a}}$. Further analysis yields:

$$B = \text{message\_size} \cdot E\left[\sum_{i=1}^{PL_R} \prod_{k=1}^{i} \left[d - \varepsilon_{\text{route,i}}\,(d-1)\right]\right]$$

, where E[.] yields the expected value in case the request path-length PL_R, the node degree d, and/or the routing efficiency are random variables.

As for the latency, the characteristic path-length and the pruning probability influence the bandwidth overhead (and scalability) in a major way, bearing in mind $PL_R = PL_{Ch} \cdot f$. Furthermore, the routing efficiency plays a significant role. It is also obvious that the packet size should be kept as small as possible. The equation further suggests that the node degree be kept low. However, this applies only if the routing efficiency is smaller than 1. And even then, a lower node degree entails a larger characteristic path-length with its negative influence on aggregate bandwidth. Note that a higher node degree also increases in principle the bandwidth available as it augments the number of links from or to a node. However, these links are only virtual links in the overlay network that all have to be mapped onto one and the same physical access line of a node.

Memory overhead for search is mainly driven by the state information to be kept on each node. In particular, this is the size of the routing table, determined by the node degree d, as well as any other state information like object links to objects on remote nodes. Processing power is mostly consumed for query routing. Hence, the routing table size and thus the node degree should be kept low to keep processing overhead in bounds. The frequency of messages to be routed will automatically be optimized when attempting to reduce the number of aggregate messages m_{agg}.

The overlay network management overhead is too system-specific to be properly addressed in this general section. It comprises all tasks to create and maintain overlay network links and routing tables. In random networks like Gnutella, *e.g.*, it is limited to ping and pong messages only, whereas it becomes more complicated in structured networks like Chord. Typical tasks then include the insertion of new nodes and new object links into the overlay. As for queries, the path-length and the aggregate number of messages for these insertion events have to be evaluated.

17.3.3 The Assessment Scheme

17.2 develops the scheme to assess the scalability of a Peer-to-Peer lookup or a search mechanism. It starts from the most generic or aggregate components or contributors to overhead resource consumption on the left hand side and

Component	Resource	Disaggregation 1	Disaggregation 2	Disaggreg. 3	Unit	Dependence
p_{query}					n/a	f(n)
	Latency				s	f(n)
			$\bar{\tau}$		s or link layer hops	f(1)
	Latency / Bandwidth		PL_R		hops	f(n)
				PL_{Ch}	hops	f(n)
				f_p	%	f(1)
	Bandwidth				Byte	f(n)
		m_{agg}	(PL_R)		(see PL_R)	(see PL_R)
			ε_{route}		%	f(1)
			d		1	f(n)
		message_size			Byte	f(1)
	Processing Power,	Message frequency	(m_{agg})		(see m_{agg})	(see m_{agg})
	Memory/ Storage	State Information	(d)		(see d)	(see d)
query_freq					s^{-1}	$f(n, \|o\|)$
$p_{OVLmgmt}$					n/a	n/a
	Cost of node insertion				n/a	f(n)
	Cost of object insertion				n/a	$f(n, \|o\|)$
	other				n/a	n/a

Table 17.2: Scalability Assessment Scheme for Peer-to-Peer Lookup and Search

analyzes the types of resources affected. It then breaks overhead resource consumption down into more specific or granular drivers or metrics on the right hand side in three levels (Disaggregation 1-3), applying the formulae derived throughout this section. For each metric, it shows the respective unit of measurement as well as its expected primary dependence on scale, which in most cases is a function of the number of nodes ($f(n)$) or the number of objects per node ($f(|o|)$). When evaluating the scalability of a lookup or search system, the alternative levels of granularity yield equivalent information and can be chosen at the evaluator's discretion.

17.4 Scalable Search with SHARK

Having outlined the key concepts on search and lookup in overlay networks and having defined the model for a formal Peer-to-Peer scalability approach,

a particular scheme termed "SHARK" (Symmetric Hierarchy Adaption for Routing of Keywords) is developed. This novel concept and middleware is scalable and offers a service for search in Peer-to-Peer networks [417, 418] and is discussed in the light of the formal model presented. Rather than flooding a network like Gnutella or imposing numerical IDs on objects like distributed hash tables, it is based on directed routing of keywords in a multidimensional redundant meta-data hierarchy. SHARK autonomously arranges nodes and objects in the network and in semantic clusters. In spite of its rich keyword search capabilities, it achieves a high degree of scalability, outperforming random networks by several orders of magnitude. It can easily be adopted for applications as diverse as file-sharing, Peer-to-Peer trading, or distributed expert and knowledge market places.

A Peer-to-Peer network consists of a set of *nodes* N being connected via a set of *links*. Each neighboring node stores a set of *objects* which constitute to the unique objects in the Peer-to-Peer network. An object is described through *meta-data* M, which determine the essential hierarchical structure for the construction and operation of SHARK. *I.e.*, this example addresses music categorization. The meta-data M^{11} yields the top-level music genre that is further divided into subgenres M^{21}. In a second dimension M^{12}, music is classified by decade of release, then by more granular timing M^{22}. M^0 is the search string, *e.g.*, 'John Patton: Let 'em roll'. In general, applications may choose to add dimensionality just for certain categories, *e.g.*, add an 'instrumentation' dimension to 'rock&roll' in addition to subgenres and timing. Search for objects in SHARK is based on query routing. A *query* is defined through a meta-data description M_q of the desired object(s) and thresholds t_{struct} and t_{rand} for the minimum required similarity of object and query description for the structured and the string expression part of the meta-data description, respectively. SHARK returns a set of *query answers* including is a *similarity metric*. The development of reasonable similarity metrics is orthogonal to and, hence, not focus of this work.

SHARK arranges nodes into a multidimensional *symmetric redundant hierarchy*. The overlay topology exactly matches the structure of the query meta-data such as to exploit the alignment for query routing. Figure 17.4 shows as an example a simplified description of two levels and two dimensions.

Each node is assigned to a *group-of-interest* (GoI) according to the objects it stores and to its prior request behavior. Each GoI represents a leaf in the hierarchy. Let $P^{(1)} = \left(p^{11}, p^{12}\right)$ denote a position on level one of the hierarchy $P^{(2)} = \left(p^{11}, p^{12}, p^{21}, p^{22}\right)$, a position on level 2. The values p^{ij} numerically represent the respective meta-data information m^{ij}. Node A in the figure would then be a member of the GoI on a leaf position $P_A = (7, 2, 2, 3)$. In contrast to a classic hierarchy, an symmetric redundant hierarchy adds redundancy so that all peers have symmetric roles in the overlay; *i.e.*, each peer can assume the role of the root of the network or be on any other level.

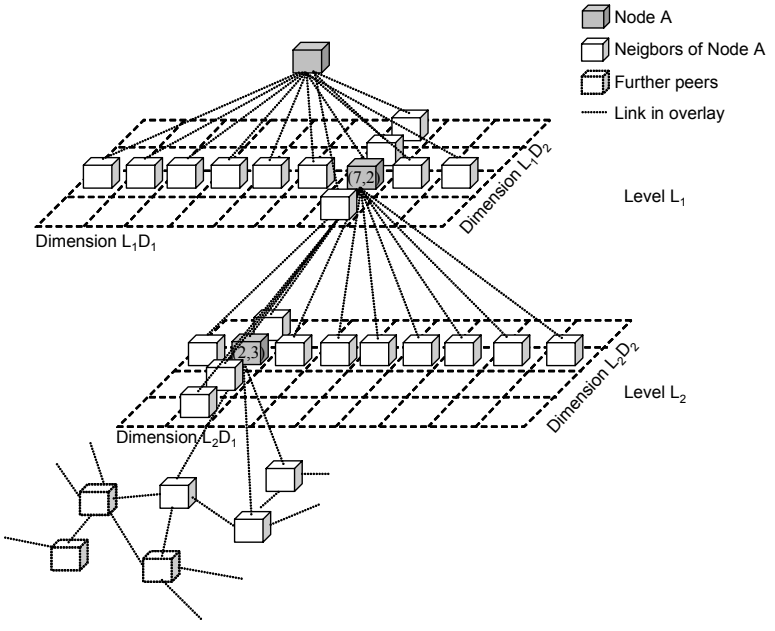


Fig. 17.4: Multi-dimensional Symmetric Redundant Hierarchy in SHARK

This improves fault tolerance and load-balancing as there is no single node acting as a root, and waives the necessity of central infrastructure, hence removing the largest roadblocks for an adoption of hierarchical structures in Peer-to-Peer networks.

SHARK adds redundancy as follows. Each node N_A on a leaf position $P_A^{(2)} = \left(p_A^{11}, p_A^{12}, p_A^{21}, p_A^{22}\right)$ also assumes partial responsibility for the parent positions $P_A^{(1)} = \left(p_A^{11}, p_A^{12}\right)$, the parent's parent and so on up to the root (in the two level case, the parent's parent is identical to the root). Hence, each node is virtually replicated on every level of the hierarchy (cf. dark grey nodes in Figure 17.4). The partiality of the responsibility results from two reasons. First, many different peers share the same parent position, thus inherently distributing the load of that position among themselves. Second, a node only maintains links to a subset of the positions on the respective next lower level in the hierarchy. As indicated in the figure, those positions form the relevant level-i-subset for a node N_A that differ from position $P_A^{(i)}$ in only one dimension. We have chosen this approach to limit state information on the nodes as well as the maintenance burden when nodes join or leave the network, thus increasing scalability of the system at the cost of only one additional hop for query routing per level. Within the leaf GoIs, peers maintain links to further neighbors SN_A^R, as indicated in the figure. The

overlay network at this stage is, however, unstructured or random. It has been shown that such networks exhibit a power-law distribution of links.

17.5 Summary and Conclusions

This overview on Peer-to-Peer lookup and scalability has shown that efficiency and scalability of these mechanisms can be formalized and may have impacts on existing systems. A heuristic approach to scalability evaluation currently prevails. Therefore, this work provides an analytical yet pragmatic assessment scheme that can help to formalize and standardize scalability investigations of Peer-to-Peer systems. As a newly proposed scheme SHARK has been outlined as a scalable approach of a symmetric hierarchy adaptation, which autonomously arranges nodes and objects in the network and in semantic clusters.