Proceedings of the Sixth International Conference on Machine Learning and Cybernetics, Hong Kong, 19-22 August 2007

INTELLIGENT OBJECT LOCALITY NAMING MODEL IN AN OBJECT-BASED DISTRIBUTED SYSTEM FOR ENGINEERING APPLICATIONS

CHUN-CHE FUNG[1], JIA-BIN LI[1] , ARNOLD DEPICKERE[2]

[1]School of Information Technology, Murdoch University, Murdoch, 6150, Western Australia
[2]Division of Arts, Murdoch University, Murdoch, 6150, Western Australia
E-MAIL: l.fung@murdoch.edu.au, j.li@murdoch.edu.au, a.depickere@murdoch.edu.au

Abstract:

Complex engineering applications demand powerful computing resources. An object based distributed system is suitable for such applications due to the inherent parallelism nature in the object based computing model. As the model is based on message passing, which in turns relies on location transparent object names, an efficient name translation system is therefore required to map the object names to the corresponding physical addresses dynamically. In this paper, we propose a heuristic distributed search algorithm, the hierarchy random walker (HRW), to take advantage of the locality of object communications thereby increasing the overall system performance. Analysis of the simulation results from a number of benchmark programs have demonstrated that the benefits of the proposed algorithm are in terms of flexibility and efficient to locate the most used objects. The approach will be applied to intelligent techniques for engineering problems.

Keywords:

Object based distributed systems; Distributed searching algorithm; Naming model; Peer-to-peer systems

1. Introduction

Complexity of modern engineering problems demands new approaches to handle large amount of data and to determine solutions in a short duration of time. Examples of such problems are real-time control, optimization, on-line scheduling and forecasting. Other issues are unstructured and multi-media information such as video and sound. Intelligent techniques such as evolutionary computing, neuron-fuzzy systems, artificial neural networks, data mining and machine learning have all contributed in diverse ways to the knowledge and understanding in engineering and computer science disciplines. However, one of the key issues is how to supply the intensive computing resources required to solve such complex problems. Apart from increasing the throughput of the computing system by improving the hardware, another approach is to deploy a distributed or parallel computing platform to adopt a form of "division of labors". An object-oriented distributed system appears to suit such application naturally. Objects are self-contained and could be distributed in a network environment through migration. An object-based computing model is based on message passing between objects and the approach is inherently parallel. The approach is therefore suitable for distributed applications.

Message passing relies on object names. A name is required to deliver messages to a specific object that is referred by that name. The name may or may not be related to the physical location of the object. In the former case, the name is location dependent. On the other hand, the name is location independent if it does not reveal the physical location. While the location dependent approach provides efficiency, the latter one provides support for transparency, scalability, mobility and reliability. Thus, current distributed systems largely implemented a location independent naming scheme. Such naming scheme, however, implies the requirement of a dynamic mechanism to map a name of an object to its corresponding physical address. A persistent name server is commonly used to provide name translation for distributed systems due to its simplicity. However, such server has issues such as poor scalability, a possible single point of failure, and bottleneck in the system performance. Consequently, a peer model has been widely studied to provide scalable and decentralized name translation for large-scale distributed systems. In a peer model, every node takes the responsibility of answering messages that query the node's local content. The node also has to delegate a message to its neighbors if the node cannot answer the query message. An overlay layer is therefore required to allow peer nodes to establish virtual links between other nodes for query delegation. In general, there are two common approaches to construct a naming system that provides dynamic name translation. They are unstructured

1-4244-0973-X/07/$25.00 ©2007 IEEE

model and structured model based on the design of the overlay layer.

Structured models [1-4] facilitate load balancing and efficient query routing. In such models, both objects and nodes are systematically organized such that objects are uniformly distributed to each node in the network.

In the structured models, the object placement policy is tightly controlled. Hence, this facilitates the process of routing the query in locating the target object. This leads to the advantage such that the search step is bound to *log(N)* with high probability (N is the size of the network) as shown in [3]. However, such tightly controlled, uniform object placement destroys the flexibility in object placement and related objects are not necessarily located close-by. This is known as the locality issue as discussed in [5] In addition, the structured model also complicates the updating procedure due to nodes arrival or departure, and object migrations.

On the other hand, unstructured models [6-8] imply neither an object placement policy nor an overlay structure. With unstructured model, nodes are free to choose their neighbors. The features of an unstructured model include support for locality due to flexibility of object placement, resilient to topology change and support for object migration. However, routing efficiency and scalability are the main weaknesses of unstructured models.

1.1. Object based distributed systems

An object based distributed system consists of a group of inter-related computing objects. Some of them are interlocked in the sense that the computation result of an object depends on the computation results from another object or objects. These objects exchange messages to each other as part of the program execution. These objects are active and thus their computation is sensitive to network latency and routing locality. This is in contrast to passive objects such as files in a peer-to-peer file sharing system. These objects have no computation implication and the network latency has less impact on the system performance. In an object based distributed system, the population of active objects varies due to frequent object creation and removal. This only requires a simple naming model that provides efficient update on any change to the object topology. Furthermore, load shedding is essential in such system as it is the process to redistribute the load so as to achieve load balancing. This, however, requires support for object migration in the system. Consequently, an unstructured model is a natural candidate for design of a dynamic name translation system for object based distributed systems. This is particular useful in the case of the implementation of intelligent engineering algorithms.

To overcome the issue of inefficient query routing, we propose a name translation model that exploits the locality of objects in order to improve the system performance. Object communications exhibit a power-law distribution as illustrated in Figure 1. We have developed a profiler to examine the run-time characteristics of object-oriented programs. Based on the benchmarking results of four open-source programs as described in Section 4, it was observed that nearly 58% of the overall messages were sent to only 5% of the objects among the population. That is, only a small group of objects receive most of the messages. This is an issue related to the locality of the objects. Our model, termed hierarchy random walker (HRW), takes advantage of the locality of objects to provide efficient search. The complete model is described in Section 3. Simulation-based analyses of four benchmarking programs are given in Section 4. The performance measurements used in Section 4 include:

- *Routing state* – the size of routing table and scalability of the overlay system.
- *Routing performance* – the search path in overlay routing.
- *Name cache* – the effect of the size of name cache on the routing performance.

Section 5 concludes the paper with a discussion on further research plans.

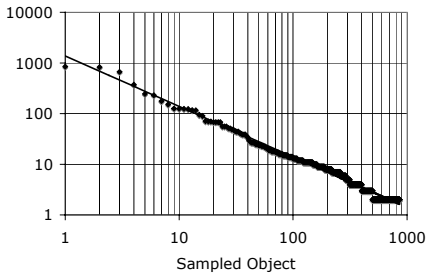

Figure 1. Plot of accumulated received no. of invocation vs no. of objects

2. Related Work

The recent study of location lookup for large-scale, dynamic networks focuses on a distributed model, particularly in the area of peer-to-peer (P2P) file sharing systems [1-4, 6-8]. In a P2P system, each node has the similar functionality and shares the responsibility of object binding and location lookup with other nodes. Because both control and data are distributed among the nodes, name

services can be performed locally, leading to better scalability and efficiency. To find a node that is responsible for the particular symbolic name or key, the requesting node queries other nodes by sending messages. The two models - structured and unstructured, for building distributed name translation system are discussed below.

2.1. Structured models

A number of well known structured models are CAN (content addressable network) [1], Chord [2], Pastry [3] and Tapestry [4]. In these systems, namespace is structurally organized and tightly controlled. The namespace is uniformly assigned to each node in the network and hence the model ensures that the network load is balanced among the nodes. A different data placement policy may lead to a different routing scheme and thereby affects the performance of the location lookup process. For instance, CAN structures namespace is based on a d-dimensional Cartesian coordination. Each node has a constant number of neighbors O(d) and the expected querying path lengths are $O(dN^{1/d})$, where N is the number of nodes in the network. Chord uses a one-dimensional circular structure with the expected number of neighbors to be O(logN), and path lengths of O(logN).

Despite the popularity of structured model, the model has several issues regarding to the tightly controlled data placement. First, objects are not randomly located but are placed at specific locations in a structured model. Thus, it makes object migration inconvenient as object migration is essential when load has to be shed in distributed computing systems. Secondly, structured models assume objects have uniform demands. In other words, the objects are assumed to send or receive number of queries evenly. However, most practical systems [9, 10] follow a power-law distribution, that is, only a few objects have most of the incoming requests. Kalogeraki et al in [11] point out that non-uniform traffic can cause query hotspots and routing hotspots in structured models. Query hotspots are caused due to the result of some popular objects that are frequently being requested by other objects. This problem can be resolved by caching the popular objects. Routing hotspots are caused by unbalanced requests sent by some node. However, this issue is harder to handle due to the use of fixed routing used in most structured models. Routing hotspots will cause link congestions. Finally, the flat namespace used in structured models destroys object locality [5], that is, related objects may not be located closely in structured models.

2.2. Unstructured models

Unstructured model is another popular approach to provide distributed location lookup. Unstructured models imply neither a centralized control nor any structured data organization is deployed. The model relies on a self-organizing protocol to update any change of the system's state such as node arrival or departure. Without the complete knowledge of the network structure, *flooding* is a typical approach to locate an object in the network. In this approach, a node queries all its neighbors within a certain radius [6] in order to locate the target object. This approach is extremely costly and leads to poor scalability and long response time. Furthermore, flooding causes duplicated messages being received by the nodes in the neighborhood. These duplicated messages do not improve the probability of finding the requested object but they can cause network congestions and block other useful messages.

Alternative protocols such as probabilistic flooding [7, 8], and random walkers [12] have been proposed to improve scalability and routing efficiency. In probabilistic flooding, a subset of the neighbors is selected to forward each query message. However, this model still gives poor performance due to the duplicated messages. Highly compressed data structures like Scalable Query Routing (SQR) [13] are proposed to reduce the number of query messages being propagated to the neighbors.

Random walkers models have been proposed in [8, 12] where only one neighbor is chosen at each cycle of query propagation. This significantly cuts down the total number of queries that is required. The weakness of the random walk model is that the expected search path is extremely long due to the randomness of the algorithm. A better approach is to use N random paths in the search [12]. However, the choice of N has significant impact on the efficiency because use of excessive random walkers leads to the same problem in flooding. This leads to the proposal of the improved hierarchy random walkers model as described in the next section.

3. Hierarchy Random Walkers

Issues concerning search in power-law networks have been studied in [14, 15]. Based on the characteristics of object communication, our model is optimized for objects which received the most messages or requests. These objects are termed "hot objects". A message is first sent to nodes with the hot objects, that is, those nodes have the highest degree of priority. If the requested objects are not in these nodes, the message is then passed to nodes with a lower degree of connection. The process will repeat till it reaches

the target object. When an object is created, the node where the object resides is marked as "cold". As the object's popularity grows, the node will become "hot" as its number of connections increases and such object will be easier to find. Objects will be removed when their computation is finished. The nodes whose objects are removed are marked as "dead" (i.e. connectivity is reset to zero) and thus a query message will avoid visiting these "dead" nodes.

Each node contains a routing table that includes information of its logical neighbors and its degree of connectivity. Table 1 shows an example of a routing table for Node 2.4 as illustrated in Figure 3. The routing table lists and orders the nodes and their respective degree of connection. For example, with Node 2.4, neighbors 0.1 and 1.2 both have 6 other neighbors (degrees). As they have the highest degree of connection, they are allocated in the first row. The first column of the table gives the size of degree of connections and the second column gives the physical or logical addresses of the corresponding neighbors. In this illustration, Node 2.4 has only one neighbor, Node 1.1, which also has a degree of 5. The last row contains two neighbors with a degree of 4 which are Node 4.5 and Node 5.3. It is worth noting that there is no upper bound on the size of the routing table like structured models. In fact, the routing tables are self-organized like Gnutella [6] and the size varies according to applications. Routing starts from the highest degrees. If there are multiple nodes available at each entry, a node is randomly chosen so that the load can be evenly distributed among the neighbors with the same degree of connections. This aids to overcome the problem of hotspots caused by preliminarily routing in structured models.

Table 1 An example of a routing table maintained for Node 2.4.

Degree of connection	Node Address of connected neighbors
6	0.1, 1.2
5	1.1
4	4.5, 5.3

To further improve the efficiency to locate the objects, each node learns the content of its neighbor's neighbors by listening to the communication passing through and stores the information in a secondary lookup table.

3.1. Locating algorithm

The pseudo code to locate objects is presented in Figure 2. Let n be the node being enquired and n' be the successor node in the search path. Given an object key x, the enquired node first looks for the key in its primary directory, where the information is more reliable. If the key is not presented in the primary directory, the querying node searches the same key in the secondary directory. This directory stores the information gathered from communications passing through from or to its neighbors. We call such information "rumors" or "gossips" [16]. If both steps fail to find the key locally, the query message is then forwarded to one of the highest degree neighbors which has not been visited before. This eliminates deadlock in query routing.

```
1.  // look for an object with key x
2.  n.lookup(x)
3.      if x ∈ primary_directory
4.          n' = primary_direcotry.lookup(x)
5.          return n'.getAddress()
6.      elseif x ∈ secondary_directory
7.          n' = secondary_directory.lookup(x)
8.          return n'.getAddress()
9.      else
10.         // randomly select one of the highest degree
11.         // neighbors which has not yet been visited
12.         for i = 1 to m
13.             n' =n" ∈ {routing_table[i].neighbors}
14.             if n' ∉ {visited_neighbors}
15.                 return n'.getAddress()
```

Figure 2. Pseudocode of the hierarchy random walkers locating algorithm

The routing table organizes nodes in a hierarchical structure is based on the size of degrees. An example is illustrated in Figure 3. In contrast to structured models, the topology is evolving as the connection state of a node changes. Like structured models, query routing utilizes knowledge of the network structure to improve searching efficiency. Figure 3 shows an example of a search path for finding the grey-colored object at Node 8.2. As a demonstration, the object at Node 8.2 is requested by an object in Node 4.7. Node 4.7 is connected to two nodes of equal degree of connection. They are nodes, 4.5 and 0.9. One of these is randomly selected, assuming Node 4.5 is passed with the query in this case. Node 4.5 subsequently forwards the query to the highest degree node which is Node 0.1 as shown in this case. As node 0.1 s not a neighbour of Node 8.2, Node 0.1 forwards the query to another highest degree node and it is Node 1.2. The target is now found.

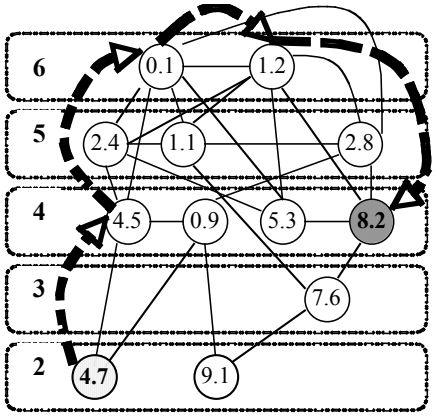

Figure 3. Illustration of Hierarchical organization of nodes based on the degree of connection of neighbors.

4. Simulation Analysis

We have conducted a series of simulation tests to study the performance and scaling property of hierarchy random walker (HRW). The network traffics were produced from traces that were extracted from an execution of four object-oriented benchmark programs. Java [17] was chosen as the object-oriented language in this study due to its popularity and modernity. Originally developed with an objective to provide platform-independent applications, it is now more noted for its extensive use in networked applications. Four separate Java applications were chosen as benchmarks for study.

- *AutoFocus* [18] is a Java-based desktop search engine. It features in Cluster Maps to present the search results.
- *DynamicJava* [19] is a Java source interpreter aimed at easing the creation of Java programs.
- *ImageJ* [20] is a Java-implementation image processing and analysis program with the use of multithreading.
- *Rhino* [21] is an open-source implementation of JavaScript written entirely in Java.

4.1. Computational model

The simulation system implemented two types of essential entities, computational nodes and active objects. A computation node is an abstraction of an execution unit, consisting of a data processing unit and a communication unit. An active object is responsible for reproducing the history of execution for a particular software object. For the sake of simplicity, each node ran exactly one object. Furthermore, a trace file contains only object communications profiled at runtime. We have assumed local computation follows a normalized distribution.

The design of active objects is based on a distributed object computational model termed Actor model [22]. An actor performs a combination of the following actions in response to incoming messages.

i) to create new actors;
ii) to send messages to other actors; and
iii) to update its state.

The last point is related to local computation while a communication model influences the other two points (i) and (ii). Message passing is asynchronous in such model. In addition, a physical network structure was also included in simulation in order to study the effects of communication delay on the search algorithms. A 2-dimentsional grid was implemented because it is one of the most common structures used in cluster computing systems.

4.2. Routing state

Routing state refers to the size of routing table contains at each peer node. In general, the larger size of routing table each node has, the faster the target object can be located. Thus, there is a trade off between routing efficiency and routing state. Structured models provide asymptotically optimal solutions to construct a routing table at a peer node. The size of the routing table is bounded to an upper limit, e.g. *log(N)* and thus it is generally proportional to the size of the overlay network. On the other hand, unstructured models do not imply any tight control on the network topology and thus they provide versatility for constructing neighbor links and routing tables. In HRW, the size of the routing table is depending on the size of computing objects, instead of the size of peer nodes. As illustrated in Figure 5, the average size of routing table per node decreases, as shown by the dashed line, as the size of the overlay network increasing from 900 to 6300, given that the object population remained at 860 by simulating the same benchmark program with varied network size. Table 2 also summaries the total size of routing table of all nodes, the average size per node, and the maximum table size.

4.3. Routing performance

The routing performance is generally measured by the search path that query messages have to travel to locate the target object. In this section, we presented the average length of search path with respect to the network size in Figure 4, and, the histogram of the length of the search path in Figure 5 for the proposed HRW. Figure 4 suggests that the average length of search path is independent to the size of the overlay network. It is because HRW allows nodes to learn new neighbors and establishes new virtual links with respect to

the topology of object communications as the execution progresses. This is contradicting to the preliminary overlay management in most structured models in which the nodes have the fairly constant or fixed number of neighbors. Consequently, the average search path of HRW is dependent to the object topology but it is independent from the size of the overlay network.

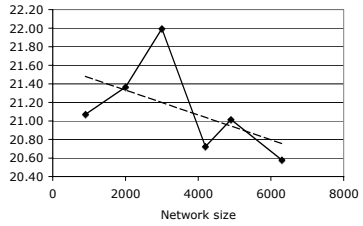

Figure 4. Plot of the routing state as a function of network size.

Table 2. Summary of the total size of routing tables of all nodes, the average size of routing table for each node, and the maximum size of routing table for each node.

Network size	900	2000	3000	4200	4900	6300
Total	17489	17797	17926	17512	17022	16772
Average	216	219	212	215	214	223
Max	21.07	21.36	22.00	20.72	21.01	20.58

Furthermore, HRW incorporates object locality by beginning searches from popular objects. Consequently, most query messages are able to deliver to the target node using a short path as illustrated in Figure 6. However, the graph has a long tail, i.e. few query messages were sent through a very long path. It implies that HRW is not efficient to locate cold objects while it can efficiently locate hot objects. This leads to the same scalability issue as found in other random walker models. The average search path to located hot objects and cold objects is listed in Table 3. In contrast, structured models, e.g. Chord, exhibits a normalized distribution in search path [2].

4.4. Name cache

In HRW, a secondary name cache is implemented to store the information of the content of neighbor's neighbor by listening to the query traffics flew through a node. In this section, we studied the impact of the size of name cache on the performance of query routing. The size of a name cache refers to the maximum number of entries can be stored in such cache. Furthermore, we implemented two cache replacement policies, least frequently used (LFU) and least recently used (LRU) for study.

Figure 7 illustrates the simulation time of the system with different cache sizes as a function of the network size. It is obvious that the use of a name cache can significantly improve routing efficiency as the simulation time is lower for the system with a larger cache. The simulation time measures the overall execution time of a benchmark program running on the simulator. In general, the execution time is a function of the message passing time in object based distributed systems. In other words, faster message passing leads to shorter execution in such system.

Figure 8 illustrates the performance of two cache replacement policies implemented in the system. It is interesting that the performance of two policies were fairly close. This suggests that there is a strong correlation between the lifespan of an object and its access frequency. That is, objects with a longer lifespan are likely to receive more messages than objects with a shorter lifespan. As a result, other form of caching policy needs to be explored, such as caching based on the similarity of two messages.

Table 3. Comparison of the average search path for hot objects and cold objects.

	Average Search Path
Hot objects	4.342750533
Cold objects	9.084308674

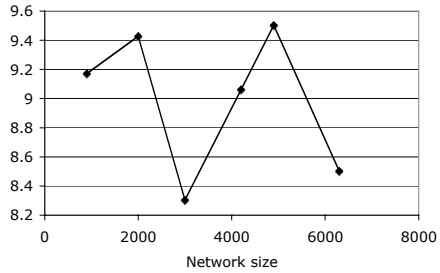

Figure 5. Plot of avg search path in HRW as a function of the network size.

5. Conclusions

This paper proposed a naming model, termed hierarchy random walker (HRW), for object based distributed systems to be used for the implementation of intelligent techniques for engineering applications. The model is based on a random walker model that provides "blindness" search. However, the model also incorporates object locality by guiding query routing to start from nodes with popular objects. Thus, HRW can provide efficient search on hot objects as demonstrated by simulation. However, HRW shares the same scalability issue as other random walker models, whereby it requires a longer path to locate rather less popular objects. On the other hand, the size of routing table in HRW is independent to the size of overlay network as opposite to structured model. This is due to the flexibility provided by HRW to allow nodes to learn new neighbors and establish new virtual links. Furthermore, the use of name caches is vital to the performance of the system. However,

the performance difference between two policies of cache replacement is not obvious due to the strong correlation between the lifespan of an object and its access rate. This approach is most suitable for the implementation of intelligent engineering solutions in a distributed environment to solve practical problems in the real-world.

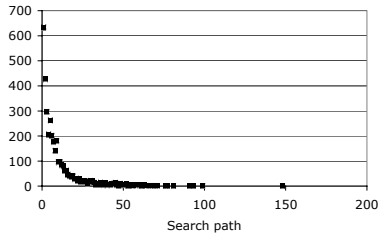

Figure 6. Plot of the histogram of the length of search path in HRW.

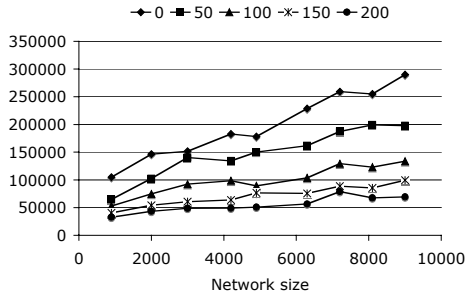

Figure 7. Plot of the simulated execution time of five sizes of name caches, as a function of the network size.

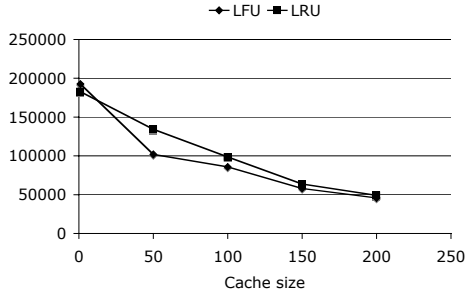

Figure 8. Comparison of the performance of two cache replacement policies as a function of the cache size.

References

[1] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, "A Scalable Content-Addressable Network," presented at ACM SIGCOMM, 2001.

[2] I. Stoica, R. M. Karger, F. Kaashoek, and H. Balakrishnan, "Chord: A scalable Peer-to-Peer Lookup Service for Internet Applications," presented at ACM SIGCOMM' '01, 2001.

[3] S. Rowstron and P.Druschel, "Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems," presented at IFIP/ACM International Conference on Distributed Systems Platforms, 2001.

[4] B. Y. Zhao, J. Kubiatowicz, and A. Joseph, "Tapestry: An Infrastructure for Fault-tolerant Wide-area Location and Routing," University of California, Berkeley, Technical Report UCB/CSD-01-1141 2001.

[5] P. Keleher, B. Bhattacharjee, and B. Silaghi, "Are Virtualized Overlay Networks Too Much of a Good Thing," in Peer-to-Peer Systems: First InternationalWorkshop, IPTPS, vol. 2429, Lecture Notes in Computer Science: Springer, 2002, pp. 225-231.

[6] Gnutella website, [url] http://www.gnutella.com/.

[7] V. Kalogeraki, D. Gunopulos, and D. Zeinalipour-Yazti, "A Local Search Mechanism for Peer-to-Peer Networks," presented at ACM CIKM '02, 2002.

[8] V. V. Dimakopoulos and E. Pitoura, "Performance analysis of distributed search in open agent systems," presented at International of Parallel and Distributed Processing Symposium, 2003.

[9] W. Aiello, F. Chung, and L. Lu, "A random graph model for massive graphs " presented at Proceedings of the thirty-second annual ACM symposium on Theory of computing, 2000.

[10] M. Faloutsos, P. Faloutsos, and Christos Faloutsos, "On Power-Law Relationships of the Internet Topology," presented at ACM SIGCOMM '99, 1999.

[11] S. Ratnasamy, S. Shenker, and I. Stoica, "Routing Algorithms for DHTs: Some Open Questions," in Routing Algorithms for DHTs: Some Open Questions, vol. 2429, Lecture Notes in Computer Science, 2002, pp. 45-52.

[12] Q. Lv, P. Cao, and E. Cohen, "Search and Replication in Unstructured Peer-to-Peer Networks," presented at ACM ICS '02, 2002.

[13] A. kumar, J. Xu, and E. W. Zegura, "Efficient and Scalable Query Routing for Unstructured Peer-to-Peer Networks," presented at INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies, 2005.

[14] L. A. Adamic, R. M. Lukose, A. R. Puniyani, and B. A. Huberman, "Search in power-law networks," Physical Review E., vol. 64, 2001.

[15] N. Sarshar, P. O. Boykin, and V. P. Roychowdhury, "Scalable Percolation Search in Power Law Networks," Cornell University 2006.

[16] Freenet website, [url] http://freenetproject.org/.

[17] J. Gosling, B. Joy, and G. Steele, Java Language Specification: Sun Microsystems, Inc, 1996.

[18] AutoFocus website, [url] http://www.aduna-software.com/home/overview.view.

[19] DynamicJava website, [url] http://koala.ilog.fr/djava/.

[20] ImageJ – Image Processing and Analysis in Java, National Institutes of Health, 2006, [url] http://rsb.info.nih.gov/ij/ 25 May.

[21] Rhino website, [url] http://www.mozilla.org/rhino/.

[22] C. Hewitt, P. Bishop, I. Greif, B. Smith, T. Matson, and R. Steiger, "Actor induction and meta-evaluation," presented at Proceedings of the 1st annual ACM SIGACT-SIGPLAN symposium on Principles of programming languages, 1973.