Porqpine: A Peer-to-Peer Search Engine

Josep M. Pujol
Technical University of Catalonia
C/Jordi Girona Salgado 1-3, C5-221
08034 Barcelona, Spain
+344017989
jmpujol@lsi.upc.es

Ramon Sangüesa
Technical University of Catalonia
C/Jordi Girona Salgado 1-3, C6
08034 Barcelona, Spain
+3493405640
sanguesa@lsi.upc.es

Juanjo Bermúdez
Technical University of Catalonia
C/Jordi Girona Salgado 1-3, C6
08034 Barcelona, Spain
+344015640
bermudez@lsi.upc.es

ABSTRACT
In this paper, we present a fully distributed and collaborative search engine for web pages: *Porqpine*. This system uses a novel query-based model and collaborative filtering techniques in order to obtain user-customized results. All knowledge about users and profiles is stored in each user node's application. Overall the system is a multi-agent system that runs on the computers of the user community. The nodes interact in a peer-to-peer fashion in order to create a real distributed search engine where information is completely distributed among all the nodes in the network. Moreover, the system preserves the privacy of user queries and results by maintaining the anonymity of the queries' consumers and results' producers. The knowledge required by the system to work is implicitly caught through the monitoring of users actions, not only within the system's interface but also within one of the most popular web browsers. Thus, users are not required to explicitly feed knowledge about their interests into the system since this process is done automatically. In this manner, users obtain the benefits of a personalized search engine just by installing the application on their computer. *Porqpine* does not intend to shun completely conventional centralized search engines but to complement them by issuing more accurate and personalized results.

Categories and Subject Descriptors
I.2.11 **[Distributed Artificial Intelligence]**: Intelligent agents, Multiagent systems. H.3.1 **[Content Analysis and Indexing]**: Indexing methods. H.3.3 **[Information Search and Retrieval]**: Information filtering, Relevance feedback, Retrieval models, Search process, Selection process. H.3.4 **[Systems and Software]**: Distributed systems, Information networks, User profiles and alert services. H.3.5 **[Online Information Services]**: Data sharing. C.2.4 **[Distributed Systems]**: Distributed applications

General Terms
Algorithms, Management, Performance, Design, Experimentation, Security, Human Fctors.

Keywords
Search, Search Engine, Distributed, peer-to-peer, p2p, Collaborative Filtering, Knowledge Sharing, Knowledge Management.

1. INTRODUCTION
Almost any web search engine presently in use (*Google*, *Altavista*, etc.) ignores completely the intentions, interests and preferences of their users. A substantial amount of information about users could be obtained from users when they perform a query or when they receive the results from a query. Some ways for taking advantage of user information have been [1,10,16,20,27,30,31] attempted but not much in the area of web searching engines. In part this is due to the highly centralized nature, at least at a conceptual level, of the indexing tasks of search engines. Our proposal attempts to create a highly distributed system where each user computer stores a part of the web model used for indexing and retrieving web resources in response to queries. All users share these partial models that globally create a consistent model for the web resource that is equivalent to its centralized counterpart. The rationale for the system is quite simple: information about web pages and other resources is shared in a peer-to-peer way. Although the system is not restricted to sharing models of web pages but of any type of resource accessible through the web, its most interesting application from the point of view of information sharing is the ability to share individual web models in order to build a distributed repository that, in principle, could complement the large centralized repositories built by search engines like *Google* or *Altavista*.

Going from a centralized paradigm towards a distributed one, brings in several advantages that cannot be exploited earlier. Basically, they are related to the fact that information has been collected, selected, stored and shared among users according to their profiles and interests. The active contributions of users present multiple advantages. In effect, the creation of a permanent user profile allows filtering search results depending on the user interests, introducing a certain degree of *personalization* in search. For example, an advanced Java programmer does not expect to obtain, for the same query terms, the same type of documents as a novice Java programmer. Moreover, user profiles allow automatic query expansion [12, 21,31] by inserting information about the user's interests so that more precise and specific results should be expected. If one considers users not only as isolated individuals but also as a community then this social dimension could be exploited in order to access to the expertise of people with similar interests. If two people have similar knowledge and interest profiles, for example two advanced Java programmers, they would probably find interesting the same pages but these pages could be seen as not interesting to other no so similar people: novice programmers could find them too technical. The social dimension of the community allows clustering users according to their interests and expertise and so focus on interesting information by reducing the domain of interest.

The usage of a combination of the individual and social dimensions of users interests has been proposed for centralized and distributed knowledge sharing environments

[1,16,21,26,27,20,31]. They usually pose two very important problems that have not been solved in a satisfactory fashion. On the one hand, storage is a problem given the potentially large number of users; repositories become intractable both for indexing and recovery. On the second hand privacy is a concern, since the queries that are issued to the search engine become a delicate piece of information; knowing that every action is used to build a personal profile, people refrain from using the system and so the overall performance degrades, since it depends on people collaboration. Our system is based on user information that is completely distributed in such a way that these two problems are avoided.

A second advantage is that it uses a model of web pages that is not directly based on page contents. Centralized search engines work by using an analysis of contents and by calculating important words in pages. Our system also uses a model based on the most relevant words but these words are not extracted by the system but introduced by human users with a high proficiency in their expertise domains. Users publish new web resources and assign a set of keywords. This can be also automatically inferred form other user's actions: a bookmarking a page can be used to create a model for that page.

The next section describes in more detail all these aspects. Section 3 discusses the prototype architecture. Section 4 shows the application. Section 5 shows experimental results from the simulations performed for system testing. Last section draws conclusions and discusses further work.

2. SYSTEM DESCRIPTION

Porqpine is a fully distributed system; each user is one or exceptionally more than one node in the overall network. All nodes in the network are identical and their total composition results in a distributed search engine in a similar way as other peer-to-peer systems are used for file sharing [13,14,17]. The difference is in the fact that what is shared here are not files but the knowledge that the different users have about web pages.

2.1 Shared knowledge

In order to obtain this knowledge the system resorts to different information sources that are always related in some way or another to the user, be it in an explicit or implicit way. In the first case information contributed by the user is the basis; in the second one, user actions are analysed to extract some relevant information [7,21].

All the knowledge generated by each user is stored in the corresponding user node, i.e., his or her personal computer. In this way each user is the proprietor of his or her information. A *User Agent* protects it and is responsible for privacy protection. Although the knowledge is stored in a local and distributed fashion, it can be recovered through queries against the system. Different nodes share knowledge in a peer-to-peer fashion in such a way that they build a global knowledge on which local search procedures are combined, which results in a global search engine. Web pages and other resources are modelled by the keywords that were used located them. This mechanism is explained in the following.

2.2 Query-based model

In order to index and store web pages and resources our system uses a methods that is different to the ones used by typical search engines. Usually search engines resort to some variation of the vector space model [28] to model the content of pages or the information associated to other resources or to link structure in order to index these pages. These models are known as *"content-based models"* and are quite comprehensive, since they associate a series of keywords to each page extracted from the page content or from the links that point to the page. The selection of which keywords are relevant, i.e., will be used to model the page is done by resorting to information retrieval techniques like, for example, *TFIDF* (Term Frequency-Inverse Document Frequency, [28]). All these processes are performed by crawlers continuously traversing the web, parsing pages and building their corresponding content models. Our system, by contrast, uses no crawler, although it could be possible that each node used several local agents devoted to these tasks.

Each web page contains many words that end up in the corresponding model, however a more accurate web page model could be obtained by using the terms appearing in queries that recover that page. That is the results obtained by a search engine in a given query q can be modelled by using the words that appear in query q and not the keywords extracted from the page. Although, apparently a great deal of information from each page is lost (keywords) more descriptive accuracy is obtained by using the terms in the query since a lot of noise is removed from the model. Another point that lead us to think that queries' words would be a good descriptor is the fact that

A query-based model is more compact and only contains words that are really significant to people, not just that are ranked high by an indexing system. This is the most important advantage of this approach, since people use to have a greater knowledge about a domain and a better synthesis ability than any classification or non-supervised learning method. For example, when a user writes in a query *"java rmi"* obtains a set of pages, the model of each page will be updated with the *"java"* and *"rmi"*. Any other person issuing a query about *"java rmi examples"* will also obtain the same subset of pages.

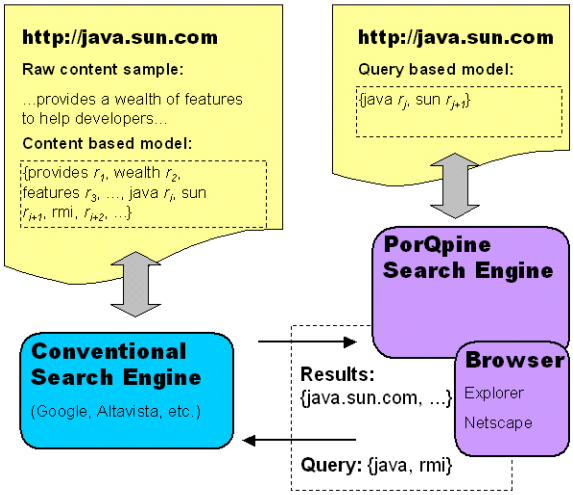

Figure 1. Query-base model diagram

People also tend to use the same subset of words very frequently [22,33]. Thus, these words could be used to model the web page. This system however depends on the existence of a large repository of content-based models so that the query-based model can be used. These repositories could be, for example, the ones

from the big search engines like *Google* and *Altavista*. By using search engines, our system is able to transform content-based models into query-based with all their advantages: smaller models, more specific, more user-oriented. This dependence could be avoided in two different ways. Either the system creates its own repository through a distributed crawler (an option we are currently studying) or an incentive schema is set up to promote that users contribute with web resources to the system.

2.3 Rating and Collaborative Filtering

Once models are stored at each node of the peer-to-peer system, the next step consists in creating a true search engine for recovering the corresponding information. Recovery means not only to return results from different nodes but also to incorporate user profiles and knowledge in order to get a more powerful, personalized rating process.

Centralized search engines always return the same results for the same query independently from users, since they have no personalization method. Some systems [21, 31] go one step further and introduce some filtering of the results obtained by a search engine by using the user's profile.

Centralized search engines do not take advantage from the fact that each user gives a different assessment for each tuple composed by a web page and a query and that each user has a profile based on his interests. In fact, maintaining and updating a centralized repository with this type of personal information is very costly and probably inefficient. That type of information storage adapts well to a decentralized peer-to-peer solution, since personal profiling information belonging to each user is kept in his computer. This is interesting both for economy and privacy reasons.

Each user in the *Porqpine* systems issues either explicitly by a voting mechanism or implicitly through his actions a certain rating for each tuple <*page,query*>. So each node stores the query-based model of each page and the corresponding set of personal assessments for that page in relation to a given query. Each user has a profile that is implicitly built by the system and identifies the user in a given role. With all this components it is possible to apply a collaborative filtering technique. That is, given a user *p* and query *q* *Porqpine* will return pages that have been well rated by people similar to *p* as a result of a query similar to *q*. This pages will be interesting ones for user *p* since its rating will be calculated from opinions of other similar people which (1) have a similar knowledge to *p* 's for the same domain and (2) a similar analysis ability that is better than the one exhibited by automatic methods and (3) have similar interests. Each user with a similar profile to a given one will recommend pages that have been interesting to him, which in turn could be of interest to similar user. This is the core of collaborative filtering systems [6,10,20,26,27,31].

For example, if user *"solso"* issues a query about *"java db jdbc"*, the system will return pages that have been well rated by people with similar interests in a similar query, for example, *"java jdbc"*. Consequently, for the same query, a user with a very technical profile will get more technical pages than user with are more general profile. This is equivalent to asking to people similar to us which pages are more important for a given topic. This approach seems more natural and promising than the present and generalized one of asking an oracle for a given topic (query). However, collaborative filtering is not without its own problems. Typical ones are the *"cold-start effect"* that keeps the system with a low performance until the number of its users reach a critical mass and, even then, the problem due to lack of collaboration on the part of the existing users. In other words, the systems needs a significant amount of data in order to respond and the users have to somehow participate in the rating process for the system to issue trustworthy ratings. Still, the system can work without annoying its users by requesting explicit ratings. In fact it can gather information about the user's interests by observing his actions [7,21].

2.4 Message propagation

So far, we have only spoken about the information or knowledge contained in each node. Nevertheless, this information needs to be shared among all the nodes. To do so, the peer-to-peer approach has been chosen, each single node is not only an information producer or consumer, but also is a router that forwards requests and results. Usually peer-to-peer systems forward requests, in such a way that, once the information or the file is found, a direct communication between the consumer (who queries) and the producer (who serves), is established. Our system does not work this way. The results are also sent back to the consumer through the peer-to-peer network. This is due to the fact that both the request and the result do not have information about who was the real initiator of the request, or even who answered with results. By doing so, anonymity is preserved, as is will be discussed in the next subsection.

In Figure 2 we can see how propagation works. A node *i* generates a query, then node *i* performs a search in the local repository for that query, and at same time the query is forwarded to node *i*'s neighbours, who act in the same manner, search in the local repository and propagate the query by becoming themselves senders of that query. By doing so, when agent *m* receives a query it will not know who the initial sender was. In fact, node *l*, who sent the request to node *m* cannot be sure if node *i* was the real initiator or a mere mediator. Only the node that launched a query knows that it was the initiator. Once results are found in a local repository and sent to the network, the same process happens again: nobody can be sure about who the producer was, because each node could be a mere mediator. When results arrive to the initiator que process finishes, and no node in the network can infer nothing for sure about their neighbours.

With the exception of how results are managed, our system is exactly like a conventional peer-to-peer system. There are several policies for request propagation in the peer-to-peer literature [9,11,13,15,23,29]. Some of them are really simple, such as *Gnutella* [13] with its breadth-first search. Other systems have more elaborated policies, based on the knowledge about other nodes [9,15,23], that knowledge can be used to improve the routing process.

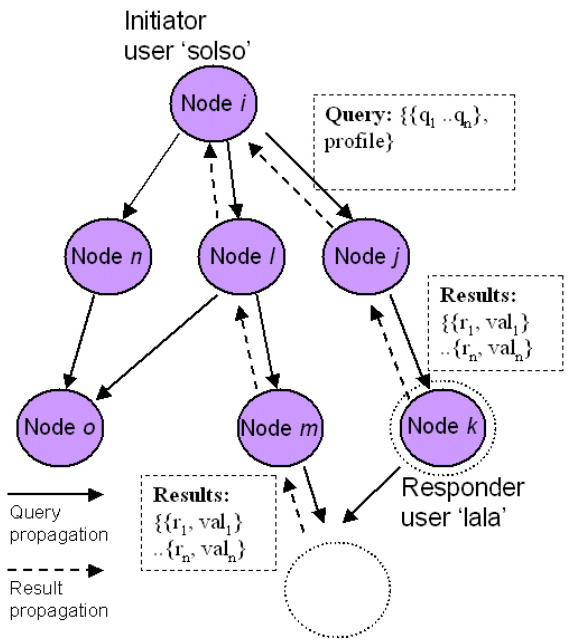

Figure 2. Message propagation

Other techniques use some kind of knowledge about neighbouring nodes in order to route message more efficiently. For example, a possible policy is to change the connectivity between nodes by getting connected to those nodes that in the past responded well to request [9,11,15,23]. Another technique consists in replicating the information on several nodes [11]. Finally, other systems that also use knowledge about neighbours in order to prevent sending the message to nodes that will surely have received the message, this amounts to pruning repeated messages. All these systems work well but cannot be applied to *Porqpine* since our goal is to protect anonymity as much as possible. So, generating knowledge about the contents of a node is forbidden in our approach.

Our solution could be assimilated to a stochastic breadth-first exploration of the neighbours of a node. That is, a message is not sent to all neighbours of a node but only to a certain percentage of them. In this way the number of messages generated by the system is kept low. The risk, of course, is that some nodes may not be ever visited. More details are discussed on the section dealing with experimentation.

Moreover, the system creates a *"neighbour profile"* that represents the knowledge contained in that part of the network that is reachable through each neighbour but that does not reflect the knowledge contained in any individual node. This information is used as a routing table for message propagation.

Choosing and adequate propagation policy is a key factor in the success of searching in a peer-to-peer system. This decision depends on factors such as the information load in the system. That is, if there is little information in the system, what is convenient is to perform a depth-first as, for example, *Gnutella* does. If there is a lot of information in the system, then a policy that uses some type of routing information is preferable. In this way, a depth-first strategy is avoided, as is the generation of a huge amount of responses.

In a distributed search engine, as our system is, the information load depends on the number of models that are stored in each node and also on the features of the performed query that can be quite frequent or rare. So, the exploration method is very dynamic. Stochasticity depends on the results obtained so far. As the number of results increases the search becomes more focused. In fact, the implemented policy is a hybrid one since at the beginning when there is little information in the system, search is quite blind and messages are propagated to a set of neighbours randomly chosen; as the systems builds better profiles, then these profiles influence set of chosen neighbours. In this way, propagation adapts to the volume of results. As the number of results increase the life of a query decreases.

2.5 Privacy and anonimity

The information and the knowledge that the system contains (queries, web page models, ratings, profiles, etc.) is a very critical one from the point of view of privacy. Accessing to this information could lead to the creation of personal profile for illegitimate used by third parties. In fact, in the normal operation of traditional centralized search engines users are subject to the same risks. If each user has static IP, then the search engine can associate that IP to the queries issued by the user. It is to be supposed that centralized search engines are not going to use illegally this information since they risk losing all their reputation as trustworthy systems.

In a peer-to-peer system, each node in the network routs queries and results. So, a malicious node could use the information that flows through it for non-authorized aims [25]. In peer-to-peer systems, direct encryption of data only protects from sniffers spying the communication between nodes in the system. It does not prevent malicious use by a node, since this node can decode the message in order to propagate it. One possibility to prevent this could be to use techniques that use matching of encrypted information pieces [2], however they are not powerful enough to be used in that kind of system.

Another possible technique to ensure privacy is the one used by *Freenet* [11] which is based in the replication and distribution of information across different network nodes in such a way that information contained in a node does not to be owned by it. This system also helps in improving information recovery since it applies an aggressive cache policy. Up to now we have not resort to this technique because we think that is very important to keep all information about a user in his own node.

Our system uses a very simple technique, which is based in the partial vision that each node has of the whole system. A node only knows about its neighbours and it can only send or propagate a request or a result to or from its neighbouring nodes.

In the request message there is no information about the real originator of the message or of the steps performed by the message, i.e., *TTL*. The message contains no information that allows identification of the request or results originator. As it has been already commented in section 2.4, this method is slightly slower since results that usually are directly transferred between the producing and consuming nodes now has to unwind the path between intermediate nodes. This disadvantage prevents nodes from knowing which is the requesting node or the one generating results. Thus, a network of *"blind proxies"* [5] is used to mask the real producers and consumers.

3. NODE ARCHITECTURE

Porqpine is composed by an undetermined number of nodes, each one is an application, more concretely, an application based on a multi-agent system. Taken together all nodes, communicating in a peer-to-peer fashion, become the *Porqpine* system. In Figure 3 there is a sketch of a node's architecture.

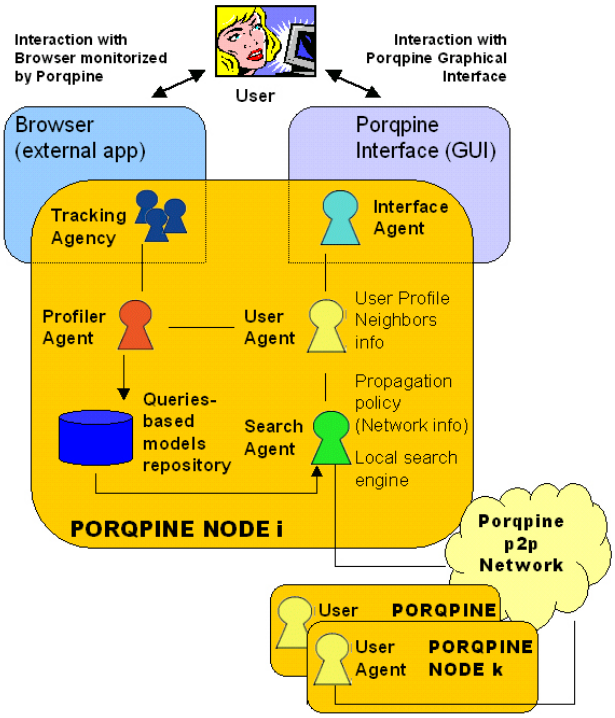

Figure 3. *Porqpine*'s node architecture

The user can interact with the system in two different ways: One is using the *Porqpine's* Graphical User Interface, from now on *GUI*. The other one is through the usual web browser that is monitored by the *Tracking Agency*. It is worth remarking that the system is always running in background, although the user is not using the *GUI*. Each agent, or agency, has a set of tasks to accomplish, which are going to be outlined in the following.

Tracking Agency: monitors the user actions that take place within his web browser. These actions are basically: 1) performing a search on a Conventional Search Engine (*CSE*), such as *Google* or *Altavista*, and 2) navigating through the obtained results and the subsequent logical actions, such as following a link, adding results as a bookmark, printing a page, saving it to disk, and so on. From these actions the system is able to catch information about the user behaviour avoiding the usage of the system's *GUI*. This agency is a set of different agents, since there are several *CSE*s to be monitored.

Profiler Agent: this agent gathers the information harvested by the *Tracking Agent* and the *User Agent* about the user in order to create knowledge about him. This knowledge is the user's profile, which is calculated by integrating the performed queries, as well as the satisfaction issued by the user for a given result and query. This satisfaction is evaluated implicitly [7,21] from the information about the user's action obtained by the *Tracking Agency* and the *User Agent*.

Interface Agent: is responsible for connecting the *GUI* with the node's agency. This agent handles all the actions done by the user within the GUI. This is the only agent that is not running permanently, it is awakened only when the user interacts with the *GUI*.

User Agent: is the agent that encapsulates all the user's information and other ones, such as profile, queries, results, results' ratings, etc. This agent, who is the connector between every two nodes, manages all the privacy rules.

Search Agent: is the agent that searches within the local repository, under the command of the *User Agent*. It is also responsible for handling messages, both requests and responses, keeping propagation policies, even though the information encapsulated by the *User Agent* is also in charge of these aspects. There is a narrow collaboration between these two agents.

Apart from the agents, there is also *Porqpine GUI*, which is the front-end for the human user as well as the knowledge repository, which stores mostly the web pages query-based model. Communication between *Porqpine* nodes is always performed from the *Search Agent* to the *User Agent*. Thus, all the information that flows into a node is managed wisely by the corresponding *User Agent*. After this brief description of the architecture se will outline how the implementation of a *Porqpine's* node follows the described architecture.

4. APPLICATION

Even though the system is not finished, the most important functionalities are already operative. Some details, such as the *GUI* or the connectivity patterns, require an extra effort before the test with real users, which is almost ready to ship. Hopefully a first public version will be available on our website http://www.porqpine.com before the end of the present year.

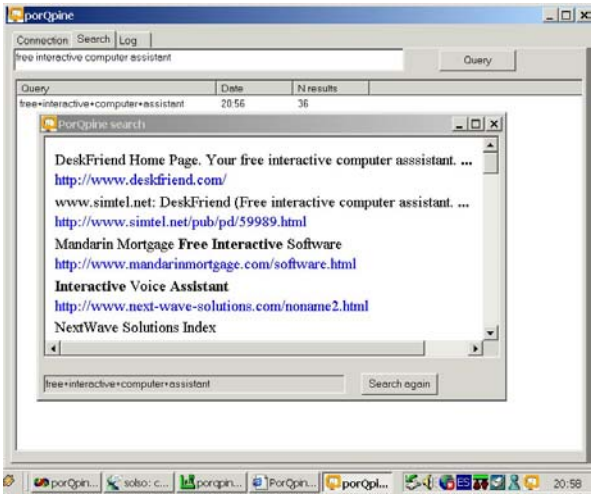

Figure 5. *Porqpine's GUI* snapshot

In Figure 5 a snapshot of *Porqpine's GUI* is shown. The application is always running in background, searching in the local knowledge repository, and routing messages. In Figure 6 some results of the system can be seen. Figure 6-1 shows the results for user *x*'s query about *"free interactive assistant"* using the system's *GUI*. These results are not from the system since there was no match for that query in our system, the results come from a metasearch process against a widely known *SCE*, more concretely, *Google*. Meanwhile, in Figure 6-2 user *y* performs the

query *"free computer buddy"* to *Google* using his habitual web browser, that is *Microsoft Explorer*. In both cases the system is registering implicitly the obtained information. After that, the system is already able to answer to user *z*'s query *"computer assistant"*, the results of such query can be seen in Figure 6-3. It can be seen that user *z* has taken profit of the information introduced by the other two users. However, the other two users were no bothered by asking them to introduce anything. All the process was done automatically by the system, without any user direct intervention. From now on, the system will be able to answer queries about that topic.

Figure 6. *Porqpine's* results example

The application that runs in each node is written in C++. Right now it only runs under *Microsoft Windows* environment, the OS and the web-browser. The executable is smaller than 200Kb, the allocation of resources, such as bandwidth and disk space, are customizable. The information repository, where the query-based models are stored in, is an efficient indexed file system developed by ourselves, in order to obtain efficiency and to avoid that the user have a third-party DB Engine installed.

5. EXPERIMENTS

Let us present and discuss the experiments that will confirm the feasibility of the system and its performance.

The best experiment would be with real users, however, this test is not possible at present time. We are on negotiation with our University Managing Staff in order to deploy the system in the Campus computers. Hopefully, the real deployment will be possible very soon. In the meantime, we have developed a framework to run simulation in order to test the behaviours of our system. The experiments are lead to answer the three more important questions about the systems feasibility and performance. The questions to find out are the following: 1) When is the system able to reply by using its own results without the mediation of the *SCSE*? 2) What quality the results issued by our system have, in terms of user satisfaction, compared with the results obtained from the *SCSE*? And finally, 3) what amount of messages are generated within the peer-to-peer network to operate, in other words, which is the network traffic generated by queries and results?

5.1 Experimental Framework

To carry out the simulations the following framework was deployed. This framework contains information of web pages, queries and users created following models that we will explain after. The framework also contains two cores of a search engine; the distributed one is our system. The other one is centralized, which acts in a similar way than a real *CSE*, from now on it will be called *SCSE*, simulated *CSE*. Each search engine core has only access to its allowed information. Thus, our system cannot access to the content-based models, and the *SCSE* cannot access neither to the query-based models nor the users' models.

The words in the framework are represented by integers. The corpus, that is the set of all the possible words, is a circular vector of size *M*. Proximity between words is equivalent to saying that come from the same domain, that is, their meaning refers to the same or similar topic. Hence, if we take a correlative set of words they will be about the same topic. They are words that usually co-occur in that domain. We are aware that is not always true because words can have different meanings. However, the loss of quality that one can incur by not having semantic consideration would bring a framework even less realistic, since there would be no way to create pages or queries about a given topic. If neighbourhood is not taken into account, pages can be created by using words randomly among all possible words in the language. But in reality, pages do have a topic. The number of words relevant to this topic is always smaller than the total number of words of the language. Each word w_k has a probability of being chosen that follows a Zipf distribution [28], that is, a power-law distribution $p(w_k) \propto k^{-2}$. The occurrence probability does not depend on the position of the word in the corpus. Hence, the occurrence probability of words contained in a set of correlative words, which represent a given domain, also follows a Zipf distribution. This distribution is found to hold for real data. Intuitively it expresses the fact that a lot of words appear rarely and some words appear very frequently.

To simulate the pages, a set of *P* web pages' models was created. Each page contained between 10 to 100 words; the number of words in a page follows a uniform distribution. Conversely, words that are chosen to appear in a page follow a power-law distribution over a correlative subset of *2x1.96xM/100* words. This corresponds to the interval *[-1.96xM/100..1.96xM/100]* if we consider that the initial word, which is the centre of the correlative subset, is represented by *0*. Therefore, by choosing words in this way we ensure that words occurring on a page are topic related, since all the subset contains words that usually co-occur. Somehow, a model with semantic coherence is obtained by this process. The more we reduce the corpus size, the more specific the topic composed by the correlative subset of words is. In order to simulate queries we used the same approach, but the correlative subset was smaller since queries are usually more specific than web pages contents. Size was *M/1000*. The number of words in a query lies in the interval *[1..5]*.

Each page also have an objective and subjective rating, respectively *obj_rat* and *subj_rat*. An objective rating is a real number in the interval *[0..1]*, and is randomly assigned. This value represents the quality of a page from the objective point of

view, and it is usually calculated by the *CSE* by analysing the links' contents and topology [18,24]. This value would be used by the *SCSE* to rank the results, as a real *CSE* does. Conversely, the subjective rating is not present in the *CSE*, since they usually do not take into account user preferences. The subjective rating is a vector of *G* reals in the interval *[-1..1]* randomly assigned. Each element of this vector is the page score for a undetermined subjective attribute, such as the presence of videos in the page or the lack of advertisements. What the attributes stand for is completely irrelevant; they are only used to evaluate the user satisfaction in the framework.

To simulate the users in the framework a set of *U* users has been created. Each user has a knowledge profile, which is updated with the queries that he issues. This profile, based on a vector space model, represents the knowledge of the user as a representation of the queries he created. In a similar fashion as it is done with pages, the user do not use all the words in the corpus. An individual user uses only *2x1.96xM/100* words from the whole corpus. Users, which are nodes in the peer-to-peer network, have a set of neighbours. For each of its neighbour, a user maintains a neighbour profile updated from the results received from it. This profile, which acts as a routing table, is useful for the message propagation policy, as mentioned in section 2.4. In the framework we have used a graph based on the Klemm-Eguíluz model [19] to interconnect the nodes. This model generates a graph, whose topology follows some of the most characteristic properties of complex networks. For example, the in-degree distribution follows a power-law and the average path length is small and the clustering coefficient is high. These topologies, typical in *complex networks*, usually arise in unsupervised dynamic systems like the Web or a peer-to-peer system [3,4,32]. This is the reason why we have chosen this model to generate a network with realistic topology. A simulated user, also has subjective preferences, *subj_pref*, a vector of *G* reals in the interval *[-1..1]* that is initialised randomly. Thus, the subjective satisfaction, *subj_satisf,* between a user and a query is calculated as follows:

$$subj\_satisf(u_i, p_j) = (\vec{subj\_rat}_{p_j} \bullet \vec{subj\_pref}_{u_i} + 1)/2$$

This subjective satisfaction value, in the *[0..1]* interval, is a score of how much the user likes a page. The vectors *subj_rat* and *subj_pref*, are never used by the *Porqpine* system, since they are impossible to build implicitly. It is quite difficult to know if a page is interesting for a user, knowing why he likes it would require explicit knowledge about the page and the user that are not available. These vectors are only used in the framework to calculate the *subj_satif* value. This value is used either in the system or in the framework in order to rank the results. In the real system *subj_satif* is calculated by the *Profile Agent* with information about the user's actions, harvested by the *User Agent* and the *Tracking Agency*. The satisfaction of a user u_i, for a given page p_j, is a combination of the subjective satisfaction and the objective quality of that page.

$$satisfaction(u_i, p_j) = \alpha(obj\_rat_{p_j}) + (1-\alpha)(sub\_satisf(u_i, p_j)) \quad \textbf{Eq. 2}$$

This is the measure we are going to use to test which system retrieves better results. α is set up at 0.5. So, we give the same importance to the single user's opinion and to the objective rating that is calculated by CSE's as a global opinion.

The *SCSE* returns a maximum of 20 results for each query. The ranking of results in the *SCSE* is done by means of Equation 3.

$$match(q_i, p_j) = obj\_rat_{p_j} + |q_i \cap p_j| \quad \textbf{Eq. 3}$$

The match between a query and a page depends on two factors: the objective rating of the page, and how many words they have in common.

Each user of our framework returns a maximum of 20 results, however the user who started the query could receive more than 20 results coming form different users. Ranking is trickier than the ranking done by *SCSEs* since our system takes more factors into account. The similarity between two queries is calculated with Equation 4. Only those queries whose similarity is higher than a given threshold are taken into account, in the framework the threshold used is 2/3.

$$similarity(q_i, q_j) = \frac{|q_i \cap q_j|}{2|q_i|} + \frac{|q_i \cap q_j|}{2|q_j|} \quad \textbf{Eq. 4}$$

Once two very similar queries have been found, user u_j ranks each one of the results, which are pages stored in his local repository, as follows.

$$match(u_i, u_j, q_k, q_l, p_m) = \frac{1}{2} similarity(q_k, q_l) + \frac{1}{2} similarity(u_i, u_j) satisfaction(u_j, p_m) \quad \textbf{Eq. 5}$$

Each user u_j knows partially the profile of the requester, since it comes with the query request. Not all the profile is added for security reasons. Then the user who has a similar query stored in his local repository will infer the user u_i satisfaction for each result u_j has. This inferred satisfaction is calculated as a product of the satisfaction of the owner of the result with that result and the similarity between both users. In other words, if p_m, was liked by u_j, and the profiles of u_i and u_j are similar, it is likely that the same page p_m will satisfy user u_i. This is how collaborative filtering works. The similarity between two users is calculated from the number of common words they have in their profiles as can be seen in Equation 6. The user's profile is a vector space model without *TFIDF* in the framework, and with it in the real system.

$$similarity(u_i, u_j) = \frac{|u_i \cap u_j|}{\min(|u_i|, |u_j|)} \quad \textbf{Eq. 6}$$

5.2 Results

Once the framework has been presented, let us introduce some of the experiments that were carried out.

The corpus size *M* was set to $50x10^3$. The number of users *U* was set to $10x10^3$, and the number of pages *P* was set to $100x10^3$. The number of queries presented to the system was $120x10^3$. The profiles had a length of 100. The graph was set up with parameters *m=10* and *a=4* [19]. The simulation was carried out as follows: at each step a query was generated by a randomly chosen user, and submitted to the system. The first one hundred queries of every ten thousand were used to build the statistics.

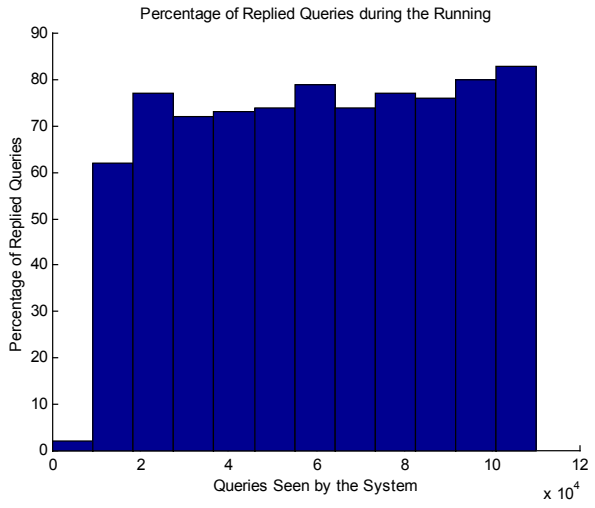

Figure 7. Percentage of Replied Queries

In Figure 7 we can observe how the system learnt to reply new queries by itself. After seeing $20x10^3$ queries, it was able to reply between 70% and the 85% of the new queries. However, once the system reached that stage it seemed to stabilize. That fact is perfectly coherent with the idea presented in [22,33]. There are a lot of queries that are very frequently repeated, and even more queries quite related among themselves. However, there are also some very infrequent queries that cannot be matched with any other previous ones. Then the real system will act as a metasearcher by searching in a centralized search engine, which have a wide content-based model repository.

The learning capabilities of the system, that is, the capacity of replying to new queries from those handled before, has been shown. Therefore, it might be concluded that a system based on distributed repositories of query-based models is able to reply a high percentage of queries. Nevertheless, retrieving information is not enough. The quality of the retrieved pages, i.e. the results is also important. We used the satisfaction between the tuple <*user, result*>, that is Equation 2, to evaluate the quality of the obtained results.

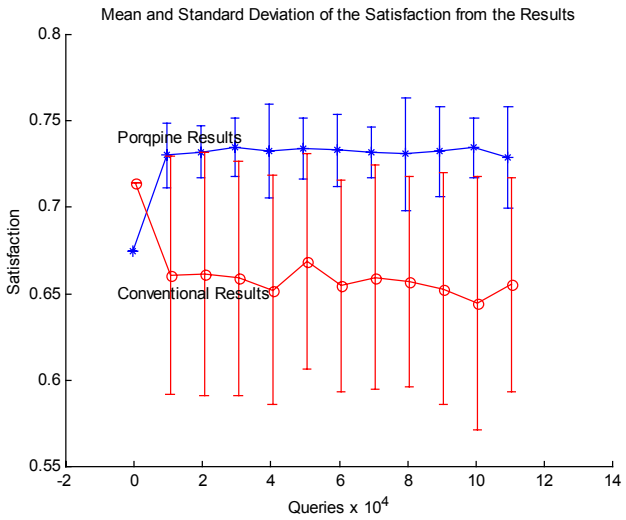

Figure 8. Satisfaction with Obtained Results

Figure 8 shows how the results issued by our systems scored better than those issued by a *SCSE* once our system had managed more than $10x10^3$ queries. The error bars, that represent the standard deviation of the satisfaction, do not overlap significantly. So, the differences in the satisfaction values are quite relevant. Hence, we can conclude that the satisfaction with the results issued by our system is higher. This is due to the collaborative filtering approach, with which the system is able to yield personalized results, against the general results harvested by the *SCSE*. Such personalized results are more suited to the requester preferences. Depending on user preferences, the system will come up with a different set of results. Clearly, this fact also happened in the results issued by our system and the *SCSE*. The common results from both approaches amounted in mean to 47.62% of the whole number of results, with a standard deviation of 33.83.

So far, thanks to the simulations on the proposed framework, we have shown two very important points: 1) the system is able to reply by itself the 70-85% of queries, after handling $20x10^3$ queries, and 2) the results issued by our system are more satisfying than those issued by a simulated conventional search engine.

To conclude the experiments, we would like to make some comments on the traffic of messages generated by the searching process. As we mentioned in section 2.4, there are several propagation policies in the literature. However, the scarce knowledge that a single node has about the whole network prevents us from using most of the techniques proposed by the peer-to-peer community to reduce message traffic. On the other hand, this limitation becomes an advantage when the anonymity of the producers-consumers of queries and results is required. We decided to trade efficiency for anonymity.

The size of the system's graph, i.e. the number of edges, is close to $200x10^3$. So, a naïve approach such as a breed-first search, as *Gnutella* [13] suggested, is absolutely out of question. Since the number of requests for a simple query would be close to the size of the graph, this would lead quickly to the collapse of the system. The stochastic approach, introduced in section 2.4, combined with the routing information given by the neighbour profiles performs very well for reducing the amount of requests. Using this propagation policy in simulations up to $300x10^3$ queries resulted in the following outcomes: the average number of requests was 37384, with a σ of 3908.2. The average number of responses was 116.06, with a σ of 200.32. And, the average percentage of retrieving, which is the ratio between results retrieved versus results available, was 0.9322 with a σ of 0.1127. Thus, comparing our policy with a breadth-first search it is clear that we have reduced the 80% of the messages by losing a 6.78% of the available results.

6. CONCLUSIONS AND FURTHER RESEARCH

Throughout this paper we have described the *Porqpine* system, which is a multi-agent system from which a collaborative and truly distributed search engine. The main features of the presented system are: 1) The introduction and usage of a novel model for web pages based on query terms instead on the content or link structure of a page. 2) The integration of the users' subjective ratings, registered implicitly by the system, in order to filter and rank the results in a collaborative fashion. 3) The avoidance of a central repository, which might be replaced by thousands of

personal repositories spreaded among the users' computers. All these advantages are only possible because the information is distributed among all the nodes of the network, which communicate with each other in a peer-to-peer fashion. The introduction of the query-based model, or the subjective rating is possible because personal information and knowledge is gathered, maintained and managed by each single user.

Our system is very concerned about the privacy issues, which are very important when personal information is managed. (Such as queries, web pages visited, web pages, etc). Privacy in our system lies in the anonymity of its members, either by using nicknames or, more important for us, by the impossibility of being aware who the real consumer-producer of queries and results is.

As it has been shown in section 5, experimental results look promising. The results of the simulations encourage us to keep working on that system, since its feasibility seems to be shown. It is worth remarking that the system uses information coming from conventional centralized search engines with a content-based repository, so the system is not aimed to replace *SCE*s, but to complement them. It does so by adding a user-personalization layer and using collaborative filtering techniques, which bring our system closer to being a recommender of web pages than a *'simple'* search engine. *Porqpine* is not able to reply to all the queries with its own information, but it does in a high percentage of cases. Nevertheless, when results are available within the system, they are of a higher quality, that is, the satisfaction the users for the obtained results for a given query is higher.

On the other hand, the system is not free from drawbacks. The most important one is the fact that it relies on the help of *SCE*s. Implementing a distributed crawling component in the system, as mentioned in section 2.2, might be a way to avoid this dependence. Another drawback is the response time in comparison to *SCE's* response time. This slowness is because of distribution of the information repositories and the peer-to-peer search. However, thanks to the propagation policies, mentioned in section 2.4, this waiting time can be reduced considerably. Anyway, never will the system reply so fast as a *SCEs* will.

As future research there are several open. The tasks which we are going to focus on are: 1) Improving knowledge acquisition from the monitoring of users' actions, in order to evaluate the satisfaction of users implicitly [7,21]. 2) Studying different network topologies, such as power-law, random or regular, to be applied as a connectivity pattern between nodes. The ultimate goal is to find a topology that maximizes a function of several factors: the average path-length, the clustering coefficient and the resilience against random failures or attacks [3,4,7,32]. This topic has not been introduced throughout the paper, but is has a vital importance for improving the propagation of information throughout the network, its tolerance to random failures and its refractiveness to directed attacks.

7. REFERENCES

[1] Ackerman, M.S., and McDonald, D.W. "Answer Garden 2: Merging organizational memory with collaborative help". In Computer Supported Cooperative Work, pages 97–105, 1996.

[2] Adar, E. and Huberman, B.H. "A market for Secrets". HP Laboratories, Palo Alto. Available on http://www.hpl.hp.com/shl/papers/mfs/.

[3] Adamic, L.A., The small world web. In S. Abiteboul and A.-M. Vercoustre, editors, Proc. 3rd European Conf. Research and Advanced Technology forDigital Libraries, ECDL, number 1696. Springer-Verlag, 1999.

[4] Albert, R and Barabási, A.-L. "Statistical Mechanics of Complex Networks". Reviews of Modern Physics, vol 74, pp 47-97, January 2002.

[5] Anonymizer. Avaialable on http://anonymizer.com

[6] Breese, J.S., Heckermann, D. and Kadie, C. *Empirical Analysis of Predictive Algorithms for Collaborative Filtering.*Proceedings of the 14th conference on Uncertainly in Artificial Intelligence, Madison, Winsconsin (July 1998). Morgan Kaufmann Publisher.

[7] Cooley, R. "Web Usage Mining: Discovery and Application of Interesting Patterns from Web Data". Ph.D. Thesis. University of Minnesota. May 2000.

[8] Cowan, R. and Jonard, N. "Network Structure and the Diffusion of Knowledge". MERIT Research Memorandum N. 99-028, 1999

[9] Druschel, P. and Rowstron, A. "Pastry: Scalable, distributed object location and routing for large scale peer-to-peer systems". Proceeding of the 18th IFIP/ACM International Conference on Distributed Systems Platforms (2001).

[10] Firefly. Available on http://www.firefly.net/

[11] Freenet. Available on http://freenet.sourceforge.net/

[12] Glance, N.S. "Community Search Assistant". Xerox Reserch Centre Europe. (2000).

[13] Gnutella. Available on http://gnutella.wego.com/

[14] IMesh. Available on http://www.imesh.com/

[15] Joseph, S. Neurogrid: Semantically Routing Queries in Peer-to-Peer Networks. Available on http://www.neurogrid.net/NeuroGridSimulations.pdf

[16] Kautz, H., Selman, B., and Shah, M. The Hidden Web. AI Magazine, (18), 1997.

[17] Kazza. Available on http://www.kazza.com/

[18] Kleinberg, J. "Authoritative sources in a hyperlinked environment". Technical Report RJ 10076, IBM, May 1997.

[19] Klemm, K., and Eguíluz, V.M. "Highly clustered scale-free networks". ArXiv:cond-mat/0107606, 2001.

[20] Konstan, J.A., Miller, B.N., Mailtz, D. ,Herlocker, J.L., Gordon, L.R. and Rield, J. "*GroupLens: Applying Collaborative Filtering to Usenet News"*. Communications of the ACM, vol 40. N.3. (pages 77-87). (March 1997)

[21] Lieberman, H. "Letizia: An Agent That Assists Web Browsing". Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence IJCAI-95, (pages. 924-929), 1995.

[22] Markatos, E.P. "On Caching Search Engine Query Results". Computer Comunications, vol 24., N.2, (pages. 137-143) 2001.

[23] Neurogrid. Available on www.neurogrid.net/

[24] Page, L., Brin, S., R. Motwani, T. Winograd, "The PageRank citation ranking: Bringing order to the Web". Stanford Digital Library working paper 1997-0072, 1997.

[25] Reiter, M.K. and Rubin, A.D. "Crowds: anonymity for Web transactions". ACM Transactions on Information and System Security, 1(1), pp 66-92, 1998.

[26] Resnick, P., Varian, H.R. "Recommender Systems". Communications of the ACM, vol. 40, nº 3 (Marzo 1997), págs. 56-58.

[27] Sangüesa, R. and Pujol, J.M. "Net Expert: A multiagent system for expertise location". Internation Joint Conference on Artificial Intelligence (IJCAI) Workshop on Organizational Memories and Knowledge Management, pp. 85-93 ,Seattle, Aug. 2001.

[28] Salton, G. and McGill, M.J. "Introduction to Modern Information Retrieval". McGraw-Hill, 1983.

[29] Stoica, I., Morris, R. Karger, D. Kaashoek, M.F. and Balakrishnan, H. "Chor: A scalable peer-to-peer lookup service for internet applications". Proceedings of the ACM SIGCOMM'01 Conference (2001).

[30] Terveen, L., Hill, W., Amento, B., McDonald, D., Creeter, J. "Phoaks: a System for Sharing Recommendations". Communications of the ACM, vol. 40, nº 3, pp 59-62, March 1997.

[31] Vázquez, A., Barrio, I., Vázquez-Salceda, J., Pujol, J.M. and Sangüesa, R. "An agent-based Collaboratory". Proceedings of ACAI2001 & EASS2001 Student Sessions, Prague, July 2001.

[32] Watts, D.J., and Strogatz, S.H., Collective dynamics of 'small-world' networks. Nature, (393), 1998.

[33] Xie, Y. and O'Hallaron, D. "Localitity in Search Engine Queries and Its Implication for Caching". Infocom (2002). http://www-2.cs.cmu.edu/~ylxie/papers/infocom02.ps