Semantic Cooperation and Node Sharing Among P2P Networks

George Exarchakos, James Salter and Nick Antonopoulos

Department of Computing, University of Surrey, Guildford, Surrey, United Kingdom
e-mail: {g.exarchakos, j.salter, n.antonopoulos}@surrey.ac.uk

Abstract

Peer-to-Peer networks are dynamic computational environments that can experience massive and sudden increases of their workloads. Typical P2P networks cannot cope with significant workload increases due to the limited processing power of their member nodes. In this paper we propose the G-ROME system which aims to interconnect multiple independent DHT-based P2P networks for the purposes of transferring capacity from underutilized to overloaded networks. The architecture is built upon the ROME protocol which has been shown to keep Chord rings at near optimal size based on their workload. We describe the G-ROME processes and we show simulations that prove its benefit in increasing workload scenarios.

Keywords

Peer-to-peer networks, capacity sharing, semantic cooperation, reactive systems

1. Introduction and Background

The architecture of Peer-to-Peer (P2P) networks can be described as being the opposite of that of the traditional client/server model. Rather than machines being designated either as servers (where resources are held) or clients (which require access to the resources), in P2P environments nodes can take on both roles. P2P technology has been quickly embraced in a variety of applications, leading to the evolution of some of the largest networks in existence. This has created the need for development of numerous novel schemes for organisation and location of nodes and their associated resources. Early examples include Napster (Saroiu et al, 2002) and Gnutella (Ripeanu, 2001). In contrast with GRID environments where networks tend to be used in multiple applications, a new P2P network is constructed for each application. To participate in an application a node must join the associated P2P network, which will often involve making some of its storage and/or processing available for others to use. Depending on the nature and popularity of the network, at times this can mean that there is an excess or shortage of capacity available.

We have previously proposed ROME (Salter et al, 2005) which aims to reduce the cost of data lookup within Chord (Stoica et al, 2003), a member of the Distributed Hash Table (DHT) class of P2P networks. DHTs provide guaranteed lookups in typically O(log n) messages where n is the number of nodes in the network (Kelaskar et al, 2002). ROME reduces this cost by providing a mechanism for controlling the size (in terms of number of machines) of the P2P network: Each node runs a ROME process which continually monitors the node's workload to determine whether it is within bounds or under/overloaded. Through a number of defined actions, extra nodes can be recruited into the network structure to deal with overload and unnecessary nodes removed to deal with underload, thus optimising the size and therefore lookup cost of the network. Nodes that are not currently members of the structure are held in a node pool on a machine designated the *bootstrap server*. A side-effect of using

ROME is that networks will contain just enough nodes to support the current workload. While ROME can deal with an overloaded P2P network by recruiting more nodes into the structure from the bootstrap server's node pool, overload conditions cannot be solved when no nodes are present in the pool. In this situation, the underlying network will be forced to drop excess workload which it does not have the capacity to cope with.

In this paper we propose an additional layer that will run on top of ROME to take a more global-view of the problem: Supposing there were two P2P networks optimised by ROME, one with many available nodes in its node pool and one with workload exceeding capacity and no pool nodes left. Our layer will create a network of ROME bootstrap servers, allowing overloaded networks to search for and use nodes from networks with spare nodes in their pools. By sharing pool nodes among networks we are able to increase the maximum capacity of a network above that of the total capacity of nodes which have joined that network.

Previously, Garcés-Erice et al (2003) have proposed the linking of multiple DHT structures in the form of a two-tier hierarchy. Nodes are grouped into DHT structures with others that, dependant on the application, may be topologically close. Each DHT structure is then linked via a higher level DHT with representative nodes from each sub-group. By assuming nodes in the higher level structure are less likely to fail than those in sub-groups, Garcés-Erice et al are able to demonstrate their approach can yield lower lookup costs than that of an equivalent single layer DHT. However, they do not use the linking of DHTs to enable sharing of nodes to increase capacity of DHTs which are overloaded. Similar multi-tiered topologies have also been proposed by others including Mislove and Druschel (2004) and Triantafillou (2003), but only for reducing latency or increasing node organisation within a single application.

Castro et al (2002) describe the concept of a universal ring linking multiple structured P2P networks. This is used for bootstrapping, allowing a node to discover an application of interest and the address of a contact node already a member of the associated network. While allowing new nodes to join their selected network, the universal ring scheme does not provide any mechanism for networks to share and release nodes as they require them.

2. Global ROME (G-ROME)

The proposed architecture, Global ROME, is designed to provide an interconnection of multiple independent ROME-enabled P2P networks, thus constructing a two-layered hierarchy of networks. The overlay network of G-ROME is used for node discovery by the ROME bootstrap servers that need extra capacity not available locally to cope with their ring's workload. This architecture is an extension of the ROME model since it establishes an unstructured P2P network on top of the ROME bootstrap servers (from now on bootstrap servers are referred to as servers) but it does not introduce any changes to the ROME functionality. The requested resources in this network are sets of nodes that contain the required amount of capacity.

A very important feature of the architecture design is the keyword-based (semantic) capacity discovery. The discovery mechanism supports the use of keywords for the description of the applications a Chord ring may serve (file sharing, lookup applications or distributed processing for example). This enables the construction of relationships between servers that have common keywords. The number of these common keywords expresses the semantic closeness of two servers. Similarly, nodes contain a keyword list to specify the set of applications that they are not willing to serve (Keyword Exclusion List - KEL). Relationships

are also created between servers and nodes since the intersection of the keywords of the two can only be the empty set. Therefore, servers may link directly to others that have common keywords (reducing the path length and thus the number of messages to discover resources) and can only manage nodes with keywords other than theirs.

The second very important feature of the proposed G-ROME is the way each server initializes and updates its neighbour list. A new server may enter the network using any other existing server (registrar) by copying the latter's neighbour list. The new server updates the neighbour list as it receives queries and answers based on several criteria. The keywords of each server help him decide whether the originator of an incoming query is semantically close enough to place the latter in its neighbour list. If the common keywords of the server and the incoming query are more than the size of the intersection of the keywords of the same server and those of another one in its neighbour list, then the query originator is semantically closer to the server than the one in the neighbour list. Therefore, the query originator is placed in the neighbour list either by appending it or replacing another one. This draws closer servers more likely to be able to provide compatible nodes. Consequently, each query also carries the keywords of the originator. The KEL enables the node's owner fully control the way its own resources will be used by the system. This contextualization helps the responder to filter the nodes that are able to participate in the requestor's ring.

The following figure provides a general overview of the proposed architecture:

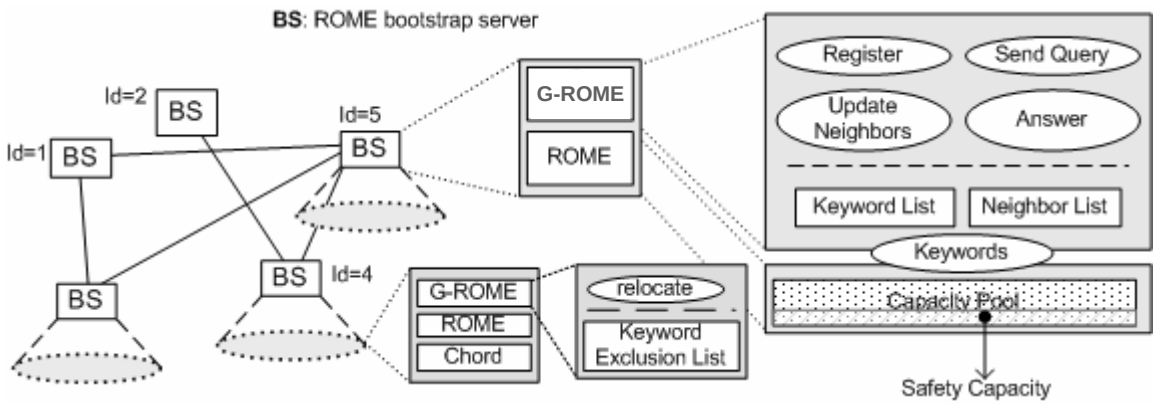

Figure 1: G-ROME's components and processes

Each ROME server connects to a number of other servers. The server's functionality can be divided into three layers: ROME operates over the Chord ring as a monitoring system; similarly G-ROME monitors ROME to invoke the suitable node discovery mechanisms when necessary. Each G-ROME server supports the following operations:

1. **Register**: has to be able to register a new server and get registered with the help of an old server. That is, exchange a list of neighbours with another server.
2. **Update Neighbours**: during the lifetime of a server within the network its neighbour list may be updated based on a heuristic function when a new query or answer is received. For instance, a query/answer originator may be chosen to enter the neighbour list if it is more *semantically relevant* than others in the list. That is, the originator has more common keywords with the receiver than another in the list.
3. **Send Query**: based on the needs of the ring if ROME cannot help with the local Node Pool, the server sends a query for the additional capacity requested to all its neighbours. The keywords of the server are attached to the query so that the remote server is able to find nodes that could serve the traffic of the requestor.

4. **Answer Query**: the remote server that receives a query either answers back if it can provide the requested capacity by supplying nodes that do not include the originator server's keywords in their KEL. Otherwise, it propagates the query to all its neighbours.

Each server keeps a certain amount of free capacity in its pool for future use by the local ring. The server does not give away this capacity to any requestor. The purpose for this technique is the reduction of the messages transmitted in the network since it prevents servers from sending queries whenever they experience slight fluctuations in their ring's workload.Both servers and nodes have to be G-ROME enabled to be able to participate in this system. That is, the servers need the G-ROME processes as described above and the nodes need a small G-ROME process (**relocate**) to resolve which is the server that monitors it. This node's G-ROME process is invoked when the node changes server. The role of '**relocate**' is to kill the old ROME process that was configured to refer to the previous server and restart it with the new server's address.

2.1 Architecture Overview

The G-ROME enabled server has three storage components: the Node Pool (pool of nodes that provide a certain amount of capacity) and the semantic-enabled neighbour and keyword lists. The pool updates the safety capacity it reserves depending on the workload of the ring and the rate this workload changes. Figure 2 illustrates the main functionality of the server which refers to the query processing (sending/propagating/answering queries).

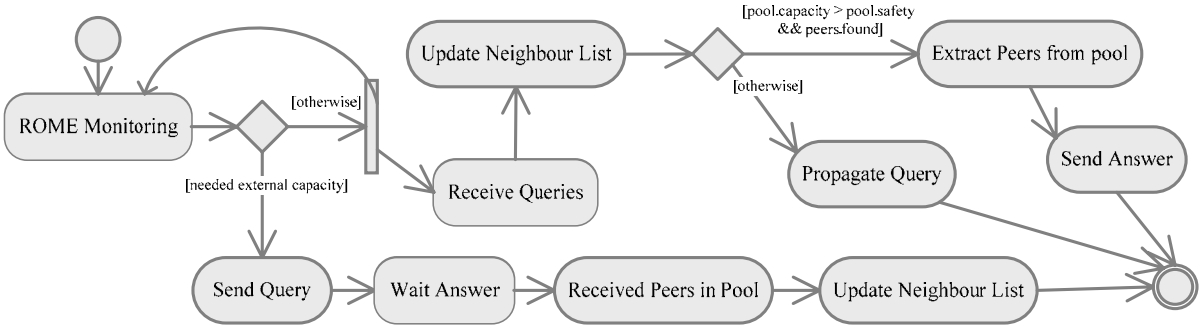

Figure 2: G-Rome Server Processes

After ROME has monitored the local ring the server is able to decide whether there is a need to acquire external capacity resources. If there is, then the server builds the appropriate query that sends it to its neighbours. When a set of nodes that satisfy its needs is found, it places references to them into its Node Pool and updates its Neighbour List with the originator of the response, if semantically closer than existing entries.

On the other hand, if no external capacity is needed the server waits for queries. For every incoming query it updates its neighbour list, if the query originator is semantically closer than an existing entry of the list, and either propagates the query to its neighbours or answers by extracting at least the requested capacity from the Node Pool. In the latter case it notifies directly the originator of the query with the IP address of the node that provides the requested capacity.

2.2 Registration Process

There are two different registration processes for a G-ROME enabled server. The first is regarding the way nodes register with a server and the second is about the way a server enters

the G-ROME network. In the first registration process the Keyword Exclusion List of the new node is checked against the Keyword List of the server to ensure that their intersection is null.

By registering within G-ROME, a server initializes its Neighbour List which can use to send queries to other servers. The process involves the new server sending a registration request to another registered server (registrar). The registrar provides its own Neighbour List to the new server. The new server is now capable of initiating queries into the system. However, its registration is not yet complete because no other server yet points to it and therefore it cannot be queried itself for capacity. To resolve this, the new server asks its new neighbours to add itself into their lists. Thus, immediately after the registration the new server's neighbours can query it the same way it can query them.

The initial neighbours of the new server may have very few keywords in common to its own list. However, the update process which is triggered through outgoing or incoming queries ensures that the neighbour list of the new server gets populated with more relevant neighbours.

2.3 Query Processing

Queries for capacity discovery are generated by any server whenever its Node Pool is empty and its ring's workload surpasses the available capacity. For the update process of the neighbour list and the keyword-based node selection, the query contains the requestor's address and keywords thus specifying the kind of applications these nodes must support.

The query is first forwarded to the neighbours, which in turn propagate the same query to their own neighbours if they cannot serve it. Even if they cannot serve the query, they may enter the address of the requestor in the neighbour list if it shares more common keywords than another server of the neighbour list. The issues of deadlock and infinite loops are handled by appending to the forwarded query the id of the server and a Time-To-Live (TTL) counter which defines the maximum number of times the query can be forwarded from one server to another (controlled flooding). If the recipient of the query has a Node Pool capacity above the required minimum safety capacity, has one or more nodes that can collectively provide the whole required capacity and the keyword intersection between the query keywords and the keywords of the nodes is null, then it sends a message to the originator with the relevant details including the unique ids of the nodes to be transferred. Otherwise, the recipient reduces the TTL counter by one and if that is bigger than zero, it forwards the query to its neighbours. This happens even if the current server can provide only some of the capacity required. This restriction is important for two reasons: a) the provision of partial answers would have required the deployment of capacity reservation schemes which could in turn result in deadlocks, and b) the requesting server can always select the first received answer because it is guaranteed to provide the capacity required and as a result the server does not have to wait for the query to explore the whole TTL-defined network horizon.

Each answer, apart from the nodes' details that are discovered, includes the address and the Keyword List of the responding server as well as the Keyword Exclusion List of each node that is included. The requestor uses the responder's information to determine whether it could replace any of its existing neighbours with this server. This will only happen if the responder is *semantically* more *relevant* than at least one current neighbour.

The requestor needs two more messages to complete the transfer of the nodes found. After

receiving the answer, it sends back a confirmation to the responder that it accepts its answer and then sends a message to every fetched node to invoke its G-ROME '**relocate**' operation.

3 Simulation and Evaluation

To evaluate the proposed architecture, a simulator was built in C++ which has mainly two aims:

1. to prove that, by interconnecting independent ROME networks, it is possible to achieve massive increase of the handled workload in the whole system without increasing the capacity available globally.
2. identify the effect of the semantic contextualization of the queries as well as the semantic-based Neighbour List updating.

The G-ROME simulator executes each experiment in three steps:

1. system-wide keyword production. That is the global set of keywords that exist in the simulated network of servers.
2. server construction and initialization with initial Node Pool, Keyword List, random Neighbour List and no ring. Each server is represented as a composable object by the other objects like: Pool, Keyword List etc.
3. for every simulated TTL the simulator executes a number of iterations linearly increasing the workload in the whole network in every iteration. This workload is randomly distributed on a random subset of servers. Some servers may get a negative workload so in each iteration some rings shrink and new available capacity is produced.

In every TTL and iteration several metrics are recorded: number of queries, number of failed queries, number of messages and each server's ring size. It is assumed that no servers fail during the lifetime of the experiment. Furthermore, the registration process of the servers is done only during their initialization. Their Neighbour Lists are initially filled with random servers and thereafter they update them through the processes described above.

The following experiments show the effect of TTL on discovering the appropriate resources in terms of the number of failed queries (capacity not found) and total number of generated messages. The last of the three experiments illustrates the ability of a single ring to handle much more workload when its server participates in a G-ROME network than it could do alone. The size of the network for these experiments is 1,000 servers each having a maximum initial capacity of 500 units. The size of each neighbour list is set to 3 entries and the system-wide keyword superset is 10,000 keywords of which a maximum of 10 are randomly assigned to each server and 3 to each node. The experiments were run with 1,000 iterations and a fixed global workload increase per iteration of 6000 units. On every iteration a random subset of servers are selected to assign the workload; no more than 10% of it is assigned to each one. TTLs {0, 1, 2, 3, 5, 7} were tested.

The following Figure 3 illustrates the reduction in failed queries (left graph) when the separate ROME networks are interconnected and (right graph) the cost (number of messages) of these queries with the increase of TTL when interconnecting the G-ROME enabled servers. In disconnected ROME networks (TTL=0) the rings cannot handle more workload and any effort to discover more capacity would result in query failure. G-ROME enables them increase their capacity and with certain TTL and workload increase, they can find all the requested capacity every iteration. Linear increase of the system-wide workload causes exponential increase in the number of messages.

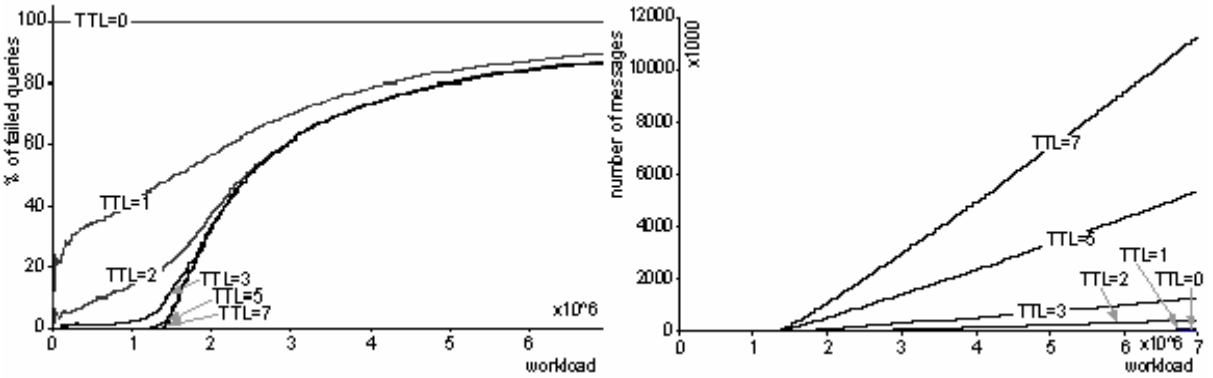

Figure 3: a) Percentage of failed queries over the total number of queries (left) and b) the number of messages generated with the increase of the workload (right)

As far as the figure 3.a, initially the lower the TTL the more queries are failed. This is because the search depth is small and given the semantic-based search of the network the available resources within this depth are limited. Furthermore, the rate of increase in the percentage of failed queries is initially lower since the semantic-based updating of Neighbour Lists creates links to servers that have enough spare nodes to serve a limited workload. As these are exhausted the failed queries increase rapidly. At any TTL, the number of failed queries is less than or equal to that of TTL=1. Increasing the TTL does not always give better results since any TTL bigger than a certain point causes the exploration of the whole network. In this experiment, this point appears to be TTL=5. Finally, the network is flooded by huge number of messages without any benefit over non-connected independent ROME networks (TTL=0).

In the 3.b figure, the base line (TTL=0) represents the state of disconnected ROME servers which do not produce any messages since there are no interconnected servers. The distance between the lines for different TTLs increases exponentially. With the increase of the TTL, the number of servers that are explored increases exponentially too. This distance between the lines stops increasing exponentially after the TTL which forces the exploration of the whole network. Any TTL above it would produce the same number of messages since we record the address of previously visited servers to prevent them being repeatedly sent the same query.

The actual benefits of G-ROME become more evident in the Table1 which illustrates the ring size increase of a single server as the TTL increases. During the experiment the simulator chose randomly to monitor the ring size of server 788. At TTL=10 the ring size is almost 21 times bigger than in TTL=0 (equivalent to a disconnected ROME server). In the first few TTLs it increases since the size of the network that is explored in every TTL increases but after a certain TTL, which causes the exploration of the whole network, this increase starts reducing.

	TTL=0	**TTL=1**	**TTL=2**	**TTL=3**	**TTL=5**	**TTL=7**	**TTL=10**
Number of nodes	226	363	749	1921	2915	4214	4728

Table 1: Ring Size of server 788

G-ROME enables ring sizes to increase while there is available capacity on the interconnected servers. Nevertheless, if the workload increases further than the available capacity, G-ROME stops having any fundamental benefit even if the queries make an exhaustive search of the network. In general, a server would need bigger TTL to discover the required capacity in the case of scarce as opposed to plentiful available capacity in the network. Assuming that the system-wide workload is uniformly distributed over the servers as far as the available capacity is adequate, small TTLs appear to be suitable for finding the requested capacity. As

the global available capacity becomes scarce slightly bigger TTLs are required. On the other hand, if the workload is distributed on a small set of servers (query hotspots) large TTLs appear to be necessary to find the requested capacity in the case of scarce available capacity and relatively small TTLs in the case of plentiful available capacity.

4. Conclusions and Future Work

In this paper we described the architecture of the G-ROME system which aims to reduce the number of failed user queries in DHT-based networks that experience high workloads. G-ROME achieves this by firstly creating a Gnutella-like interconnection of independent P2P networks and secondly by providing mechanisms that enable overloaded networks to use the underlying interconnection in order to acquire spare nodes from underutilized networks. The resulting system is simple and our simulations have shown that it can significantly increase a network's ability to deal with sudden and considerable increases of its workload.

We are currently investigating the applicability of other searching mechanisms that can potentially reduce the number of messages required to find a given capacity. We are also developing a prototype that will enable us to measure more accurately the potential benefits and costs of G-ROME.

5. References

Castro, M., Druschel, P., Kermarrec, A-M. and Rowstron, A. (2002), "One Ring to Rule Them All: Service Discovery and Binding in Structured Peer-to-Peer Overlay Networks", in *Proceedings of the SIGOPS European Workshop*, France, September 2002.

Garcés-Erice, L., Biersack, E.W., Ross, K.W., Felber, P.A. and Urvoy-Keller, G. (2003), "Hierarchical peer-to-peer systems", in *Parallel Processing Letters*, Vol. 13, No. 4, pp. 643-657.

Kelaskar, M., Matossian, V., Mehra, P., Paul, D. and Parashar, M. (2002), "A study of discovery mechanisms for peer-to-peer applications", in *Proceedings of the 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid Workshop on Global and Peer-to-Peer on Large Scale Distributed Systems*, Berlin, Germany, May 2002, pp. 444-445.

Mislove, A. and Druschel, P. (2004), "Providing Administrative Control and Autonomy in Structured Peer-to-Peer Overlays", In *Proceedings of the 3rd International Workshop on Peer-to-Peer Systems (IPTPS'04)*, San Diego, CA, USA, 26th-27th February 2004, pp. 162-172.

Ripeanu, M. (2001), "Peer-to-Peer Architecture Case Study: Gnutella Network", *University of Chicago Technical Report TR-2001-26*, University of Chicago.

Salter, J., Antonopoulos, N. and Peel, R. (2005), "ROME: Optimising Lookup and Load-Balancing in DHT-Based P2P Networks", in *Proceedings of the 2005 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'05)*, Las Vegas, NV, USA, 27th-30th June 2005.

Saroiu, S., Gummadi, P.K. and Gribble, S.D. (2002), "A Measurement Study of Peer-to-Peer File Sharing Systems", in *Proceedings of Multimedia Computing and Networking 2002 (MMCN'02)*, San Jose, CA, USA, 18th-25th January 2002.

Stoica, I., Morris, R., Liben-Nowell, D., Karger, D.R., Kaashoek, M.F., Dabek, F. and Balakrishnan, H. (2003), "Chord: a scalable peer-to-peer lookup protocol for internet applications", *IEEE/ACM Transactions on Networking*, Vol. 11, No. 1, pp. 17-32.

Triantafillou, P. (2003), "PLANES: The Next Step in Peer-to-Peer Network Architectures", In *SIGCOMM Workshop on Future Directions in Network Architectures (FDNA-03)*, Karlsruhe, Germany, 27th August 2003.