ROME: Optimising Lookup and Load-Balancing in DHT-Based P2P Networks

James Salter
Department of Computing
University of Surrey
Guildford, Surrey, UK

Nick Antonopoulos
Department of Computing
University of Surrey
Guildford, Surrey, UK

Roger Peel
Department of Computing
University of Surrey
Guildford, Surrey, UK

Abstract – Distributed Hash Tables (DHTs) have been used in Peer-to-Peer networks to provide key lookups in typically O(log n) hops whilst requiring maintenance of only small amounts of routing state. We extend ROME, a layer which runs on top of the Chord DHT to provide control over network size through monitoring of node workload and propose the use of processes to reorganise nodes and add or remove them from a pool of available machines. We show this can reduce further the hop counts in networks where available node capacity exceeds workload, without the need to modify any processes of the underlying Chord protocol.

Keywords: peer-to-peer, distributed hash tables, Chord, hop count

1.0 Introduction

Currently, a large volume of research is taking place into the use of Distributed Hash Tables (DHTs) to provide lookup services in structured Peer-to-Peer (P2P) networks. These provide guaranteed lookup in typically logarithmic hop counts whilst maintaining only small amounts of routing state [1]. One such DHT is Chord [2], which organises a network of *n* nodes into a one-dimensional key space forming a ring. Each node is given a unique identifier and becomes responsible for keys whose numerical hash lie between its identifier and that of its predecessor in the ring. Given a key, Chord looks up the associated value in O(log *n*) hops around the ring.

Since the number of hops to process the lookup is directly proportional to the number of nodes in the ring (*n*), lookup hop counts can be optimised by keeping the number of nodes in the ring as small as possible. We have previously presented ROME [3], a layer running above the standard Chord protocol to control the construction of the network via selection and placement of nodes within the Chord ring. ROME provides a monitoring process to detect overloaded and failed nodes and actively seek either direct replacement or additional nodes to service the workload, with nodes taken from a pool of available nodes held on a bootstrap server. Nodes are registered with this server rather than being immediately added to the ring, allowing the ring size to be kept minimal. Initialisation and node join operations plus actions to deal with node replacement, addition and failure have been defined. In this paper we extend ROME, allowing for reorganisation of workload across nodes already present in the ring and for removal of excess nodes from the ring caused by a reduction in workload.

Many extensions and modifications to the standard Chord protocol have been proposed (e.g. [4,5]), several of which share our aim of reducing the number of hops to process lookups. However, to our knowledge none of these do this by controlling the expansion of the Chord ring itself. In fact, ROME is complementary to these extensions and using them together may provide greater message savings.

A model using a similar process to our swap action to achieve load balancing has been proposed [6]. However, they divide each node into multiple virtual servers, creating coarse granularity of keyspace which can be moved since as only whole virtual servers can be moved to different nodes, rather than individual keys as in ROME. They implement a similar directory scheme to our blackboard approach, but each node periodically reports its status to the directory, increasing message cost over ROME, where nodes only post requests to the blackboard if they are under/overloaded.

Reassignment of keys to neighbouring nodes

(such as in our slide action) has been previously discussed [7]. However, their approach allows nodes to randomly check the status of neighbouring nodes and *always* rebalance the load, probably more often than necessary. Our slide action is only executed when a node is under/overloaded.

2.0 ROME

Modifying a node in the Chord ring has the potential to influence the workload of the node's immediate successor, so both the node and its successor must be locked before actions can be taken, thereby removing the likelihood of chain reactions and over-compensation. If action is required, ROME will send a message to the node's successor. A response will be sent back, indicating whether or not the lock was successful. If the lock was unsuccessful, ROME will wait for a random exponentially increasing time period (as per the standard Exponential Back-Off Algorithm) before restarting the monitoring process. Otherwise, ROME will lock the current node and attempt to resolve the under/overload by performing one of the node actions. Placing the current node in control of the locking procedure and only seeking to lock the successor prevents potential deadlock situations from occurring.

Dependent on whether the node is under or overloaded, actions are attempted in a predefined order: If a node is overloaded, a slide operation is attempted first. If this is unsuccessful, the node's details will be sent to the blackboard, followed by a wait to see if a swap is triggered. If no swap trigger message is received, ROME will attempt to replace the current node with a more capable one from the node pool. If no suitable node is available, ROME will request details of another node to add to the ring from the node pool. If the overload has not been resolved, the process will be restarted following a wait defined by the Exponential Back-Off Algorithm.

If a node is underloaded, a slide and swap operations are attempted first. If these are both unsuccessful, ROME will seek to remove the node (as long as doing so would not cause the node's successor to become overloaded). If none of these actions are successful, ROME will wait for a time period (defined again by the Exponential Back-Off Algorithm) and restart the monitoring process.

2.1 Slide

In some situations, a node may be overloaded while its successor is underloaded or vice versa. In this scenario, the node could be slid around the ring to share workload with its successor by taking on more or giving away some of its key space. For example, the node may reduce its workload to within its upper threshold boundary by sliding away (lowering its node identifier) from its successor, thereby moving a portion of its keyspace to its successor. The slide operation is the first choice when action needs to be taken, as it can be both beneficial to the current node and its successor.

To perform a slide operation, ROME on the node taking action sends a message to the node's successor containing the amount of workload that should be added/removed from the current node to restore it to its target workload. ROME on the successor checks, in the case of the current node wishing to offload some of its workload whether the successor's current workload combined with the node's excess workload lies within the successor's upper threshold, or in the case of the current node requesting to take on more workload whether the successor's current workload less the node's requested workload is at least as great as the successor's lower threshold. ROME on the successor then indicates whether the criteria have been met. If so, ROME on the current node will calculate its new identifier such that it takes on the correct amount of keys to reduce/increase its workload (in the case of it wishing to take on further workload the identifier will have been calculated and sent by the successor). It then sends messages to the Chord protocol running on itself plus its successor and predecessor to inform them of its new identifier. The Chord protocol will then transfer the necessary keys.

The slide action does not require the presence of the bootstrap server. This allows at least a possibility of reaction to under/overload in ROME even if the bootstrap server has failed.

2.2 Swap Nodes

The slide operation performs a degree of local load-balancing within the ring. However, it may also be that there is a node on one side of the ring

that is underloaded and one on the other which is overloaded. This may cause the overloaded node to trigger an add operation while the underloaded node triggers a remove. It would be desirable to swap the two nodes instead.

To enable swap operations, we add a blackboard to the bootstrap server. This acts as a global notice board on which nodes can post requests to swap with others because of under or overload. If a node wishes to perform a swap operation, it sends a message to the bootstrap server containing its current workload and its threshold values. The node then waits for a predefined time period for a reply signifying a match has been found. If it does not receive this, ROME judges the swap request unsuccessful and will try other actions instead. The swap request is also automatically removed from the blackboard at the end of this time period.

A matching process running on the bootstrap server searches for nodes that could be swapped: For two nodes A and B, if *A_Upper_Threshold* > *B_Current_Workload* > *A_Lower_Threshold* and *B_Upper_Threshold* > *A_Current_Workload* > *B_Lower_Threshold* then A and B can be swapped. In this case, a message is sent to ROME running on each of the nodes to be swapped, informing them of the ID of the other node. ROME then sends messages to the Chord protocol layer running on the two nodes plus their successors and predecessors informing them of the new identifier.

2.3 Remove Node

The actions introduced above are mechanisms to cope with nodes becoming overloaded and to better organise workload within the ring. However, it is also possible that workload may decrease to an extent that underload cannot be rectified through re-organisation, making it unnecessary to keep all nodes in the ring.

When ROME wishes to remove its host node, it sends a message to its successor containing its current workload. ROME on the successor checks whether the node's workload combined with the successor's current workload is over the successor's upper threshold. It responds indicating whether the remove operation can take place. If the node can be removed, ROME sends a message to the Chord protocol on its predecessor informing it its successor is now the current node's successor. Similarly, it sends a message to its successor informing it its predecessor is now the current node's predecessor. Finally, ROME sends a message to the bootstrap server, which triggers the node to be re-added to the pool of available nodes.

3.0 Evaluation

Here we present an evaluation of the success of ROME, demonstrating how the newly introduced processes to slide, swap and remove nodes allow ROME to respond to reduction as well as increase of workload. To test this we built a simulator in Perl and conducted an experiment to measure the number of nodes in the underlying Chord ring and the utilisation of the ring capacity (how much of the available capacity provided by nodes in the ring is used to service the workload), as the network-wide workload was varied over time. The results for a network containing 1000 available nodes are plotted in Figure 1. In the experiment, each of the 1000 nodes was randomly assigned a capacity between 200 and 5000 units, with the target workload set at 50% of

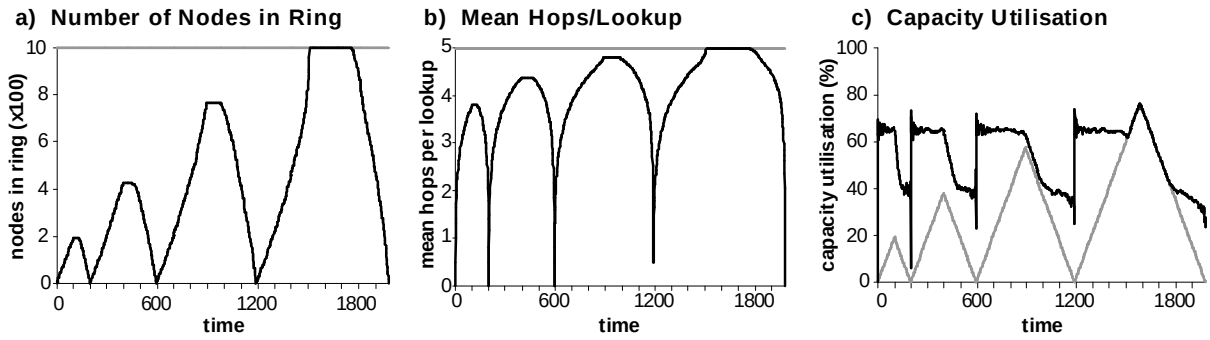

Figure 1: Comparison of ROME (black line) against standard Chord (grey line)

the capacity and upper and lower thresholds at 75% and 25% respectively. Workload was increased and decreased linearly to progressively greater peaks throughout the experiment.

Figure 1a shows how the number of nodes in the ring varies in response to the increase/decrease in workload using ROME in comparison with the constant 1000 node ring if only Chord is used. The slightly curved line observed when the ring is increasing in size is due to the difference in capacities of available nodes: ring nodes are re-arranged or replaced with nodes with increased capacity before nodes are added to the ring, so node addition becomes more common when the highest capacity nodes are already present in the ring. This curve would become more obvious the more variable the nodes in the pool. The flattened peaks are caused by the cautious approach to node removal: although we wish to keep the Chord ring as small as possible, it would be imprudent to remove a node to immediately have to re-add it (and incur the associated message cost of removal and addition) if the workload were to begin increasing again.

The mean number of hops taken per lookup is plotted in Figure 1b. Since the mean hop count is directly proportional to the number of nodes in the ring ($\frac{1}{2}\log_2(n)$ [2]), this follows a similar pattern to the number of nodes shown in Figure 1a. It is obvious that, because of the constant ring size in standard Chord, the mean hop count observed when using ROME is smaller (or convergent to) that of Chord.

Finally, Figure 1c plots the capacity utilisation of the ring employing ROME compared to standard Chord. We define capacity utilisation to be $\frac{network\ wide\ workload}{\sum ring\ node\ capacities}$. Excluding the brief periods where workload is changing between decrease and increase at times 200, 600 and 1200, ROME's capacity utilisation rests between 35% and 65% for the majority of the simulation. Whilst this may at first seem low, this sits comfortably within the zone of normal workload where action will not be taken (recall the thresholds were set at 25% and 75% and the target workload to 50% for this experiment). Crucially, capacity is more effectively utilised compared to Chord, except where the rings contain the same number of nodes where both ROME and Chord have equivalent capacity utilisation.

4.0 Conclusion

In this paper we have proposed extensions to ROME to slide, swap and remove nodes from the underlying Chord ring. Through simulation we have demonstrated construction of rings smaller than standard Chord rings is possible using ROME when the total capacity of machines available to become members of the ring exceeds the workload placed on the ring, with the benefit of achieving lookups in less hops than standard Chord. Where capacity and ring workload are near equal, ROME provides lookup in equivalent hops to standard Chord.

5.0 References

[1] M Kelaskar, V Matossian, P Mehra, D Paul and M Parashar, "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.

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

[3] J Salter and N Antonopoulos, "ROME: Optimising DHT-based peer-to-peer networks", in *Proceedings of the 5th International Network Conference (INC 2005)*, Samos Island, Greece, 5th-7th July 2005.

[4] M F Kaashoek and D R Karger, "Koorde: A simple degree-optimal distributed hash table", in Proceedings of the 2nd International Workshop on Peer-to-Peer Systems (IPTPS03), Berkeley, CA, 20-21 February 2003.

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

[6] B Godfrey, K Lakshminarayanan, S Surana, R Karp and I Stoica, "Load balancing in dynamic structured P2P systems", in *Proceedings of the IEEE INFOCOM 2004*, Hong Kong, 7-11 March 2004.

[7] D R Karger and M Ruhl, "Simple efficient load balancing algorithms for peer-to-peer systems", in *Proceedings of the 3rd International Workshop on Peer-to-Peer Systems (IPTPS04)*, San Diego, CA, 26-27 February 2004.