Chord-on-Demand

Large-Scale Distributed Systems

Sébastien Vaucher
sebastien.vaucher@unine.ch

17 December 2015

Contents

1 Introduction **1**

2 Algorithm **2**

3 Analysis of the results **3**
3.1 Convergence of T-Man 3
3.2 Convergence towards a correct DHT 4
3.3 Bootstrapping the fingers 4
3.4 Multiple successors 5

4 Conclusion **6**

MASTER IN COMPUTER SCIENCE

unine
UNIVERSITÉ DE NEUCHÂTEL

1 Introduction

This report presents the results obtained in the third and last assignment of the Large-Scale Distributed Systems course taught at the University of Neuchâtel. The goal of the assignment is the implement the Chord-on-Demand [1] algorithm, also known as *T-Chord*. Chord-on-Demand uses T-Man [2] to emerge a Chord [3] structure from the random graph provided by a peer-sampling service. The implementation is done in the Lua programming language, using the Splay framework.

Apart from this report, a number of files are supplied:

chord-on-demand.lua
: Implementation of the Chord-on-Demand algorithm (Chord & T-Man).

pss.lua
: Implementation of a peer-sampling service carried over from the first assignment.

chord-on-demand.sh
: Bash script to launch the program on a local machine.

***.txt**
: Raw logs generated by the program running on the cluster of the university.

***.gp**
: Gnuplot scripts generating the graphs found in this report.

parse_tman.pl
: Log parser in Perl that computes the average distance between nodes in T-Man view throughout the execution.

parse_bootstrap.py
: Log parser in Python that computes the number of nodes having the correct Chord successor at each T-Man iteration.

generate_graphs.sh
: Script to generate the plots found in this report from the raw logs. Some plot data is directly generated by this script, while other are delegated to Perl and Python scripts.

All the data presented in this report is the result of executions on the Splay cluster of the university. For this assignment, we were given more liberties regarding what we wanted to do. The workplan that was agreed for this project was as follows:

1. Implement the T-Man algorithm
2. Integrate Chord jump-starting using T-Man
3. Add fingers management (bootstrapped from T-Man)
4. Implement and test failure-resilience:

a) Have a list of k successors as backup for each node

b) When a successor node fails, replace it with its successor

c) Implement maintenance of the Chord ring by regularly testing for nodes aliveness

Tasks 1 through 3 were successfully realized and tested. Task 4 was only partially implemented. Only sub-task 4a was implemented.

2 Algorithm

The goal of Chord-on-Demand is to *bootstrap* a Chord structure from a random graph. The algorithm is described in [1], but a summary is provided in this section. Adaptations to the algorithms described in the paper needed to be made; they are listed in this section.

The T-Chord algorithm proceeds by iterations. At each iteration, each node will exchange its current view, a random subset from its peer-sampling service, along with information about itself. When a node contacts another, it sends these nodes metadata. The passive node will also exchange the same type of information. The exchange is therefore bi-directional.

Whenever a node receives information about peers from another node, it will combine them with already known information residing in its local view. The resulting aggregation will be sanitized, to remove doubloons and copies of the local node. A sorting function is applied to the sanitized list. This function defines how to organize the emerged structure. In our case the nodes are sorted in ascending order of distance (using the identifier from Chord) to the local node. After n iterations, the converged list of nodes contains the nodes that are closer to the local node, i.e. immediate predecessors and successors in the DHT.

To this procedure, we added a step that guarantees that the immediate successor of the local node is always kept, otherwise there is a probability that the algorithm would never manage to bootstrap a valid DHT structure.

When a fixed number of iterations have been performed, we start bootstrapping the Chord structure from the T-Man view. What happens is that we sort the T-Man view by Chord ID in ascending order. The node that is immediately before the current node becomes its predecessor in the Chord ring. The node that immediately follows the current node becomes its successor. To increase the reliability of Chord, we add all the nodes that follow the current node, and are located in the following half of the ring space, as backup successors.

The algorithm as described above, while creating a fully-functional Chord DHT, performs miserably. The Chord algorithm mandates the addition of a finger-table that

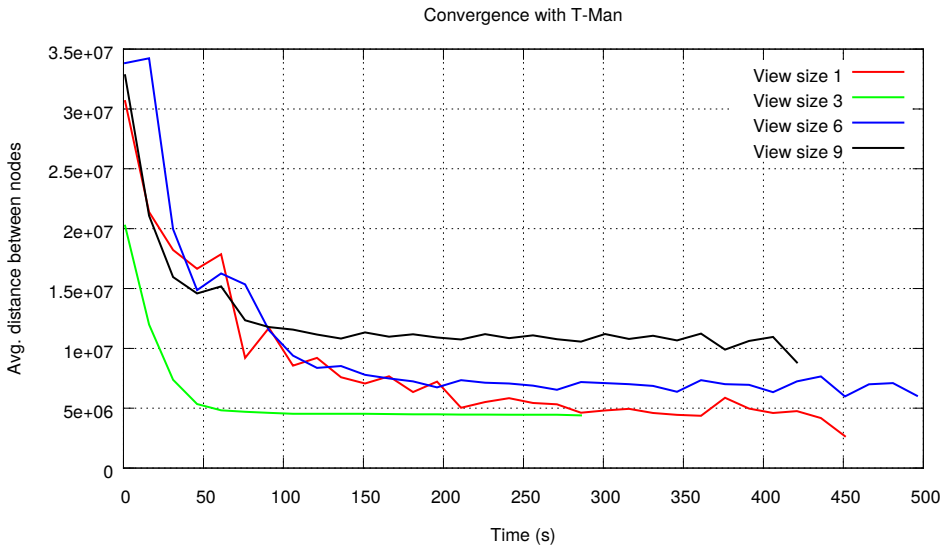

Figure 1: Evolution of the average distance between nodes across T-Man iterations

provides shortcuts across the ring, leading to smaller hop-counts for individual queries. These fingers can be discovered using the T-Man algorithm. Our implementation can be schematized as m individual instances of T-Man, each with a view size of 1. m is the size of the finger-table, which is linked to the number of bits in the identifier of Chord. Of course, we did not ran $28 + 1$ instances of T-Man in parallel. What we did was integrate the finger selection logic in the existing T-Chord pipeline. This way, there is no network overhead.

3 Analysis of the results

3.1 Convergence of T-Man

We tested our implementation against a group of 64 nodes. In Figure 1, we show how the average distance between nodes evolves. We see that the T-Man algorithm correctly manages to rapidly feed each node with a view containing nodes that are close. With lower view sizes, the average distance between nodes is smaller. This does not indicate better performance, but is a side-effect of the reduced neighbors per node. With less neighbors, a node only chooses the closest nodes and discards distant nodes. This leads to a smaller average distance between nodes.

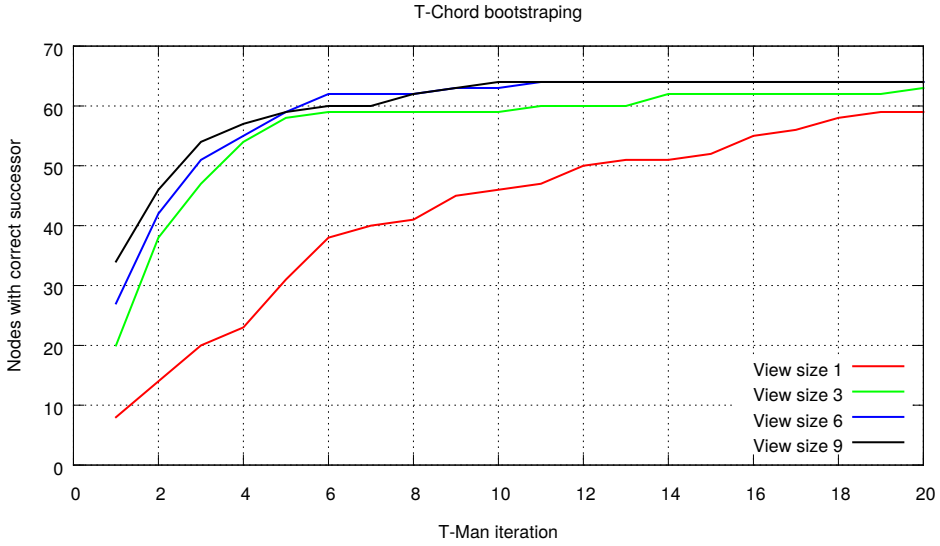

Figure 2: Bootstrapping performance across T-Man iterations

3.2 Convergence towards a correct DHT

Our specific usage of T-Man is T-Chord. To generate Figure 2, we tried to bootstrap a Chord DHT after each T-Man iteration. In the Y axis, we display the number of nodes in our system that managed to bootstrap Chord using the correct successor. A result of 64 indicates that the perfect ring was discovered.

The size of T-Man's view has a clear influence on the time it takes to convergence towards the DHT structure. With reasonable values (view size of 6 and up), we can see that the perfect result can be reached in 11 iterations. More importantly, we see that, for all view sizes, the slope of the curve is never negative. This proves that our implementation is working as intended.

The number of cycles we measure is quite high in comparison with results from [1]. In this paper, the authors needed < 8 cycles for a network of 2^{10} nodes. In our result, we see that the algorithm convergences rapidly to more than 60 correct successors, but then stagnates and takes some more cycles just for the last incorrect nodes.

3.3 Bootstrapping the fingers

To analyze whether our finger bootstrapping mechanism works as expected, we let the T-Chord algorithm run for 20 iterations. After these iterations, we performed random queries on the DHT and collected the number of hops needed to reach the responsible node.

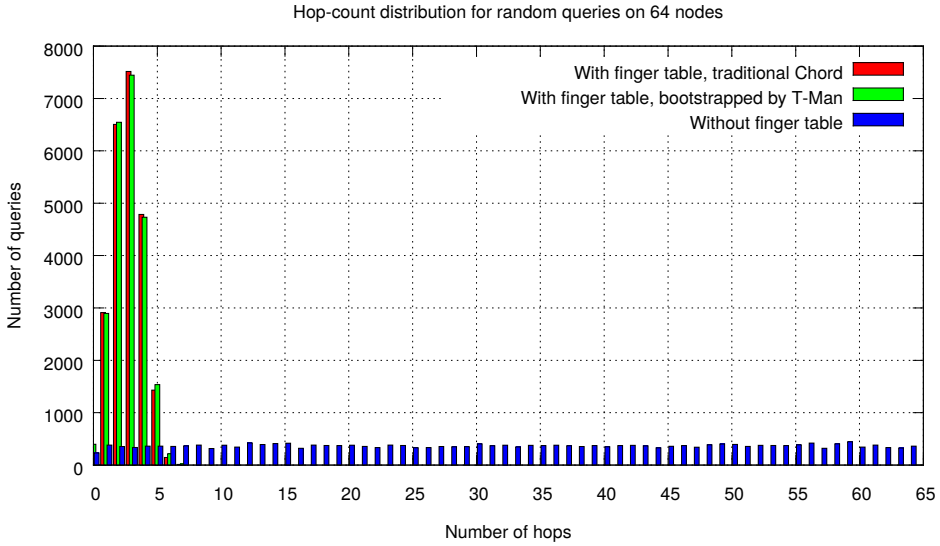

Figure 3: Distribution of hop-counts with and without fingers

In Figure 3, there are 3 different results plotted. The blue bars show the case with no finger table. As expected, the distribution is flat, denoting very poor performance. The results using red bars are brought over from the previous assignment on classical Chord. Finally, the result that uses fingers bootstrapped from T-Man is shown in green.

We can see that the fingers discovered from T-Man provide sensibly the same result than fingers discovered using the mechanisms from traditional Chord. This highlights that our implementation is totally functional. Combining the discovery of the finger-table within the existing T-Chord pipeline is in fact a valid approach.

3.4 Multiple successors

Our implementation of T-Chord bootstraps at least one node as successor. In cases where it is possible, our implementation of the algorithm will bootstrap successive successors as backups.

In a fault-tolerant implementation of Chord, these backup successors would take place of a failed successor to permit the continuation of operations under churn. Each node should periodically check if its successor is alive. When a failure is detected, the node will replace it with the failed node's own successor. This resiliency mechanism was not implemented for this assignment, mainly due to a lack of time (it was in the workplan as tasks 4b and 4c).

4 Conclusion

The goal of the T-Chord algorithm is to create a DHT from a group of nodes. We have successfully implemented the necessary components that allow to transition from a random graph to an organized DHT meeting Chord's specifications. Our implementation is able to converge to a correct DHT in a handful of cycles.

We made some modifications to the algorithm proposed in [1]. Instead of choosing finger nodes from the final view, we integrated the finger-bootstrapping procedure in the existing T-Man pipeline, in the select_view function[1]. This method does not incur additional costs in terms of network usage, and proved to discover fingers that performed as great as those discovered using the complete finger discovery mechanism used in the non-fault-resilient variant of the Chord protocol.

We were able to reuse pieces of code written for the two previous assignments with only minor modifications to them. The peer-sampling service and the implementation of Chord were brought over in this fashion. This assignment can therefore be considered as a successful mashup of different distributed systems techniques.

The foundations to render the Chord protocol fault-resilient are in place. Each node is bootstrapped with multiple successors, and the implementation of Chord is able to use the fix_fingers and stabilize mechanisms. Some more work would be needed to integrate fallback procedures into the regular query methods. An additional step would be to periodically check for the consistency of the ring, and transition to a backup successor if appropriate.

References

[1] A. Montresor, M. Jelasity and O. Babaoglu, "Chord on demand", in *Peer-to-Peer Computing, 2005. P2P 2005. Fifth IEEE International Conference on*, Aug. 2005, pp. 87–94. DOI: 10.1109/P2P.2005.4.

[2] M. Jelasity and O. Babaoglu, "T-Man: gossip-based overlay topology management", in *Engineering Self-Organising Systems*, ser. Lecture Notes in Computer Science, S. Brueckner *et al.*, Eds., vol. 3910, Springer Berlin Heidelberg, 2006, pp. 1–15, ISBN: 978-3-540-33342-5. DOI: 10.1007/11734697_1.

[3] I. Stoica *et al.*, "Chord: a scalable peer-to-peer lookup service for internet applications", *SIGCOMM Comput. Commun. Rev.*, vol. 31, no. 4, pp. 149–160, Aug. 2001, ISSN: 0146-4833. DOI: 10.1145/964723.383071.

[1] Known as merge $(message, view)$ in [1].