Sloppy Management of Structured P2P Services

Paolo Costa[1] Guillaume Pierre[1] Alexander Reinefeld[2]
Thorsten Schütt[2] Maarten van Steen[1]
[1] Vrije Universiteit Amsterdam
[2] Zuse Institute Berlin

1. INTRODUCTION

While most structured peer-to-peer services are conceptually very simple, their implementation is not. Even though similar observations can be made about any type of software, it can be shown that large parts of implementation complexity derives from the way management tasks such as error condition handling are dealt with.

The traditional way to manage distributed systems software is to list all possible error conditions such as churn and partial node or network failures, and come up with repair algorithms that take care of maintaining the desired structure despite adversary conditions. However, implementing repair algorithms is cumbersome, and any error can potentially lead to complex liveness bugs [6], not to speak about unwanted cross-interactions of repair schemes.

We observe that most — if not all — structured peer-to-peer systems deploy strategies to sustain temporary structure inconsistencies that derive from error conditions. In this context, our position is that *explicit repair algorithms can and should be avoided in the implementation of structured peer-to-peer services. Instead, we should use continuous lazy background algorithms to handle non-functional management tasks such as routing table maintenance, while relying on the original structured algorithms for the functional tasks such as routing messages through a DHT.* We call this form of probabilistic overlay maintenance "sloppy management".

We already demonstrated in two simple situations that such a dual approach based on a structured deterministic functional plane and an unstructured probabilistic non-functional can actually work [2, 8]. As an example we briefly discuss in Section 2 how gossip-based protocols can be used to maintain the routing tables of a Chord DHT. However, if we are to take the sloppy management approach seriously, many other non-trivial management tasks should be handled by the probabilistic plane. We discuss them in Section 3, and finally conclude in Section 4.

2. EXAMPLE

Teaching the algorithms from Chord to graduate students can be done in no more than three slides, if one is willing to ignore all management issues. The first slide introduces the circular ID space and the way nodes and data items are hashed into IDs; the second slide presents routing tables and shows which fingers should be stored where; the last slide shows how messages can be routed to a key in $\log N$ hops. As every peer-to-peer expert knows, the real difficulty is to build and maintain the routing tables over time in a decentralized fashion.

In [8] we showed how very simple gossip-based probabilistic algorithms can be used to efficiently maintain Chord routing tables, without using any explicit repair algorithm. The probabilistic routing table management algorithm is decomposed in two separate gossip-based overlays. At the bottom level, nodes periodically gossip with each other to create an ever-changing random network within themselves. It is important to note that gossipping takes place at a fixed periodicity, even if no structure inconsistency appears in the Chord routing tables.

The second management overlay uses similar periodic gossipping, but this time it does not aim at building a random network. Instead, it selects links that are good candidates for being used in the Chord finger table. Each node periodically exchanges its current list of fingers with one peer from either of the two gossipping overlays. Under steady conditions, routing tables are kept unaltered. However, if failures occur, the system can easily replace failed nodes by acquiring new identities from its neighbors. Note that this does not require any particular action from the periodic gossip: all error conditions such as churned nodes and partial network failures are simply ignored.

One can show that this algorithm converges extremely fast and can maintain the Chord structure over time, even under massive churn.

Gossip-based management algorithms have a cost, though: gossipping takes place periodically irrespective of the appearance of structure inconsistencies to repair, which means that some continuous background network traffic is created. This traffic can however be kept low, in the order of less than one message per second and per node on average, which is of the same order (if not lower) to the typical overhead of *alive* messages or periodic stabilization protocols executed by structured systems [1].

3. RESEARCH PLAN

As discussed in the above example, epidemic protocols have proved to efficiently maintain topological properties of several overlay networks [8, 5]. Nevertheless, it is not yet clear to what extent this can be generalized to any possible structure. For instance, while maintaining Chord finger tables is relatively easy, maintaining more complex structures such as [7, 4] may be harder.

The most prominent research challenge, however, is to extend this approach to the management of other non-functional properties, beyond topological constraints. Ideally, one would only need to specify the non-functional invariant that must be maintained (e.g., routing tables should be consistent, system load should be balanced, etc.); the probabilistic layer would autonomously maintain the desired properties, without the need to define explicit algorithms to detect and repair invariant violations.

One interesting issue of large-scale distributed systems that could be addressed by sloppy management is load balancing in the presence of skewed load distributions. Traditional techniques to avoid 'hotspots' resort to periodically run network-wide protocols in order to compute the load of each node, and reorganize the network if some inequality is discovered. This may however incur significant overhead, thus reducing the overall performance of the system. In addition, defining the optimal frequency of this operation is highly critical. A too short period would result in excessive overhead, while a too long period would poorly tolerate unexpected bursts of activity. Similarly, on-demand protocols, running only when one node considers itself overloaded, would save some bandwith but may be inappropriate to ensure timely recovery.

In contrast, sloppy management would continuously work in the background to balance the load. By periodically gossiping with random nodes in the network, a node could detect if other nodes are less overloaded and could properly repartition the load, *before* becoming a hotspot. Furthermore, potential inconsistencies arising in the process (e.g., a node leaves while the takeover is occurring) could be seamlessly solved, relying on the aforementioned fast convergence of gossip-based protocols.

A more difficult challenge is to autonomously deal with other types of system misbehavior due to (partial) failures or system peculiarities. For instance, most peer-to-peer protocols assume, unrealistically, the absence of firewalls and the possibility to establish connections between any pair of nodes. In reality things are more complicated and specific machinery is required to encompass firewalls and network policies. We believe that epidemic protocols have the potential to detect and work around these anomalies in a fully decentralized and autonomous way [3]. For example, in Chord, if a firewall prevents a node to communicate with one of its fingers, the local gossip-based layer will declare that finger dead (from his perspective) and fill in the finger table with another node, thereby maintaining correct routing.

We imagine that epidemic protocols could be used to autonomously work around other types of misbehavior, such as certain types of malicious attacks. For instance, the *Eclipse* attack consists for one or several attackers of attracting most of the overlay links to them before disconnecting, thus creating unrecoverable network partitions because most links are gone. We imagine that, thanks to the very fast convergence properties of epidemic protocols, innocent nodes could autonomously detect and 'repair' the implied skew of in-degree distributions, thereby defeating the attack. Interestingly, in this case, it would not be necessary to identify the origin of the attack nor the vector by which it intruded into the system; simply, good nodes of the overlay would detect that some property is not as it should be, and take autonomous action to bring the system back to a workable state. Thus, for a healthy system it must only be ensured that the majority of the nodes are at a healthy state.

4. CONCLUSION

Most implementation complexity of large-scale distributed overlays is due to algorithms that aim at repairing the overlay in the presence of many adversary events, such as node churn and partial failures. Instead of devising ad-hoc repair algorithms to take care of each possible issue, we propose an approach where **the programmer simply specifies the non-functional invariants to be maintained**, while a simple probabilistic background task is in charge of maintaining the invariant.

We have successfully applied this technique to a few simple examples such as building and maintaining Chord routing tables over time. This encourages us to envisage more ambitious uses of the same technique, for example to balance the load of an overlay under (ever-changing) skewed load distribution, or even to repair the effects of firewalls on the connectivity between nodes.

Pushing the approach to its extreme, we imagine that we could use similar techniques to handle a range of possible attacks. We however remain extremely careful about that last claim: attackers should be expected to deploy the best available *strategy* to defeat the sloppy management system. We therefore consider that any contribution, no matter how modest, of sloppy management towards improving the security of peer-to-peer overlays, would constitute a convincing proof of their power to handle conventional issues such as churn, load balancing, and partial network failures.

Acknowledgements

This work was funded by the XtreemOS E.U. FP6 project.

5. REFERENCES

[1] Castro, M., Costa, M., and Rowstron, A. Debunking some myths about structured and unstructured overlays. In *Proc. NSDI* (2005).

[2] Costa, P., Pierre, G., and van Steen, M. Autonomous resource selection for large-scale grid systems. Submitted for publication, 2008.

[3] Drost, N., Ogston, E., van Nieuwpoort, R. V., and Bal, H. E. ARRG: Real-world gossiping. In *Proc. HPDC* (July 2007).

[4] Gupta, A., et al. Meghdoot: content-based publish/subscribe over P2P networks. In *Proc. Middleware* (2004).

[5] Jelasity, M., and Babaoglu, O. T-man: Gossip-based overlay topology management. In *Proc. Intl. Workshop on Engineering Self-Organizing Applications* (Hakodate, Japan, may 2006).

[6] Killian, C., Anderson, J. W., Jhala, R., and Vahdat, A. Life, death, and the critical transition: Finding liveness bugs in systems code. In *Proc. NSDI* (Apr. 2007), pp. 243–256.

[7] Schutt, T., Schintke, F., and Reinefeld, A. Structured overlay without consistent hashing: Empirical results. In *Proc. CCGrid* (2006).

[8] Voulgaris, S., and van Steen, M. An epidemic protocol for managing routing tables in very large peer-to-peer networks. In *Proc. DSOM* (Oct. 2003).