P3N: Profiling the Potential of a Peer-based Data Management System

Mihai Lupu
Singapore-MIT Alliance
National University of Singapore
mihailupu@nus.edu.sg

Y. C. Tay
School of Computing
National University of Singapore
dcstayyc@nus.edu.sg

1. INTRODUCTION

A large number of peer-to-peer (P2P) networks have been introduced in the literature since their popular advent in the late 1990s. In particular, structured P2P overlays have gained much attention since 2001. They are noted mainly for their theoretical properties such as balancing of communication, storage and processing load, as well as elegance of design.

The problem that resulted from so many proposals is that, when one needs to implement a new peer-based data management system (PDMS), there are many good network models to chose from and no tools to compare them. Most works that compare P2P networks require the existence of a full implementation and, consequently, are hard to use. They do little to help in the initial phases of the design process.

Our work takes an orthogonal approach to the problem of comparing P2P architectures for PDMSs. We look at the underlying structure and, in particular, at its potential of tolerating node failures. We do this because in a PDMS it is important to have query answer completeness. Otherwise, the system just stores data without an effective way of fully taking advantage of it. We propose a system entitled *Profiler for Peer-to-Peer Networks* (P^3N) which is aimed to answer questions present in the initial phase of the design process of a PDMS. For instance, given our expected node failure rate, can we afford maintaining only a fixed number of neighbors for each node, like in Viceroy [13], or must we use a variable number of neighbors like in Chord [19]? And if the number of neighbors is variable, how many should we maintain in order to keep the query success rate above our quality threshold?

These are basic questions, that are currently hard to answer without a full implementation of the P2P network. Our previous work [12] provides an analytical basis. However, for arbitrary graphs, it is often hard to prove analytical results. Instead, P^3N provides the user with a lower and upper bound on the query success rate of a P2P network for different node failure rates. In the process, it also analyses the bandwidth load and identifies potential bottlenecks in the network. P^3N has been designed around the notion of Cayley graphs, but, as shown before [12], many P2P overlays are either Cayley graphs themselves or are mappable to Cayley graphs.

PVLDB '08, August 23-28, 2008, Auckland, New Zealand

1.1 Related Work

Assessing the quality of P2P networks is a problem that has received considerable attention, mostly in comparing some new proposal with an existing one. Virtually all newly introduced P2P networks have such a characterization, showing improvement against previous solutions.

Still, it remains difficult for a system designer to choose among the various structures. One solution is, of course, to take all the networks, or as many as possible, and run them with the same data to see which one performs best according to some specific metric of interest. A recent such approach is that of [10]. The problem with such a test is that one has to implement, or obtain, the different networks and run them with the same input. Similarly, [3] presents P2PTester, a system to assess the performance of existing P2P networks. It functions as an additional layer on top of some existing architecture. Both systems are interesting for an organization that already has deployed a system, but are of little use to system designers, as they need to make a decision *before* implementation.

Another solution to the problem of comparing different P2P architectures is to provide a general model that unites several of them. For instance, a unified API for structured P2P overlays is proposed in [4]. A P2P application is divided into tiers, where tier 0 is limited to the basic key-based routing API; tier 1 implements abstractions such as DHT or multicast; and eventual higher tiers implement applications on top of the two basic levels. [4] argues that the API can be easily implemented on four types of structured overlay networks but does not provide further results in that direction. Instead, in [18], Shudo et al. propose an implementation of the multi-tier structure of [4] in OverlayWeaver. They further separate the routing process from the routing algorithm, attempting to reduce the amount of code needed to be written by a potential user. Still, OverlayWeaver is complex for two reasons: the underlying framework is thread based and thus limited by the operating system, leading to an upper bound on the number of nodes that can be simulated. Secondly, implementing a new architecture still requires a significant amount of programming work. In this second sense, a more compact system is [11]. However, the compactness is achieved by using a non-standard language, a variant of Datalog, which must be learned before being able to use the system.

Another recent example of a general model is [2]. Even without the benefit of an implementation, [4] and [2] are a motivating exercise, arguing for the case of a unifying framework. The reason for which they lack an implementation is, we suspect, that they are too general and require a fair amount of programming to launch a new system. Consequently, we must restrict the analysis domain if we want to aim for a system that is both useful and useable.

1.2 Impact analysis of overlay structure

The works of Ratajczak and Hellerstein [16], Qu et al. [15] or

Gummadi et al. [5], successfully argue for an analysis of the underlying *static* structure of the P2P overlays before looking at the algorithms implemented on top of it to deal with load balancing and churn. First, [5] presents an empirical analysis based on the simulations of various types of networks and concludes that routing geometry is fundamental and that flexibility is a key parameter. They also suggest that the ring (here by ring they actually mean the chordal ring graph, i.e., a ring plus a set of chords connecting different sections of the ring) is a powerful candidate for the *universal* network, on top of which most routing and balancing algorithms may be implemented. Their analysis still lacks a cohesive analytical framework; even in the case of the ring, the results do not take into account that the arrangement of chords of the ring has a significant effect on the graph's properties.

The empirical results of [5] call for an analytical study to identify the graph theoretic properties of the networks. Such a study was initiated in [16] and independently extended in [15]. Both works identify Cayley graphs as the common underlying structure of the most popular P2P overlays. Additionally, they suggest the possibility of defining new networks based on so-far unexplored Cayley graphs, such as the *star graph* or the *pancake graph*. A useful continuation is the implementation of these proposals, and, maybe more importantly, the identification of significant measures to differentiate existing P2P overlays.

Our initial analysis [12] lead us to further extend the set of P2P architectures that are mappable to Cayley graphs despite their apparent differences. For instance, we have taken a close look at BATON [8], a recently proposed tree-based peer-based data management system capable of answering both equality and range queries. We have observed that, provided the addition of one node (regardless of the size of the tree), there exists a mapping between the tree and the chordal ring representations. In particular, if directed, this chordal ring representation forms the well-known Chord network [19]. The existence of such a mapping indicates that the study of the subclass of structured P2P networks based on Cayley graphs provides useful insights for a much larger array of networks than what may be initially expected.

To analyze the static underlying structure of P2P networks we propose to demonstrate P^{3}N (Profiler for Peer-to-Peer Networks) - a system that allows the user to easily design and test new structured P2P models based on Cayley graphs. P^{3}N is implemented using the Java programming language. P^{3}N simulates a P2P infrastructure on a local machine or, if available, on a cluster of machines, and collects information about the query success rates as well as about the bandwidth load distribution in the network. P^{3}N allows the user to :

- easily modify existing network models: imagine you are interested in the effects of adding a new edge to a Chord [19] overlay, or increasing the dimensionality of CAN [17].
- measure query success rates as a function of the node failure rate - which in data management terms translates as an estimate of query answer completeness.
- assess the bandwidth load distribution in the network, both in the presence and in the absence of node failures.

We will demonstrate how to implement a new P2P structure and assess its properties in terms of query success rate, query response time and bandwidth load balancing - probably the main factors in judging the performance of any peer based data management system.

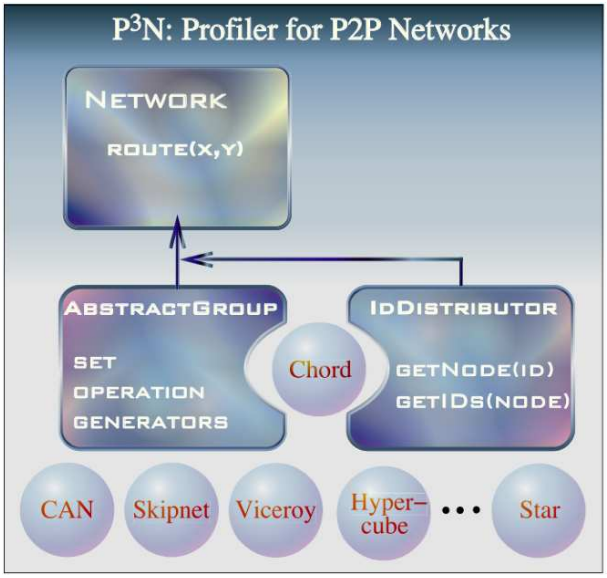

Figure 1: Architecture of Cayley-based implementation of common P2P networks simulator

2. P^{3}N

Before we describe the system, we cannot overlook the definition of a Cayley graph, since that is the main concept underlying P^{3}N.

DEFINITION 1. *A* Cayley graph $Cay(G, S)$ *given by a group* G *and a subset* $S \subset G$ *is the graph whose vertices are the elements of the group and there exists an arc between any two vertices* u, v *if and only if there exists an element* $s \in S$ *such that* $v = u * s$. *Where useful, we will label this arc with the element* s.

Using Cayley graphs, the main goals of the Profiler for Peer-to-Peer Networks (P^{3}N) can be summarized as follows:

- *Quick results.* Our main goal is to provide a tool, using which, a system designer would get an approximate idea of the performance, function of three variables: the number of neighbors of each node, the maximum path length and the query failure rate.
- *Flexibility.* While partially constrained by the Cayley graph theoretical model, we want to give the user the possibility to test innovative structures. Such a system must be modular and each module, representing a new structure, must be easily integrated into the system.
- *Reduced requirements.* To some extent, this is part of our main goal of obtaining quick results. The system must be easily deployable, should not require privileged execution rights on other machines. Also, defining new structures should be done using a popular programming language in order to smoothen and reduce the learning curve.

In assessing the performance of a network, P^{3}N considers two quality measures: the percentage of failed queries and the load distribution on the links and nodes.

2.1 P^{3}N's Architecture

Figure 1 illustrates the architecture of P^{3}N . All components of the simulation framework were implemented using the Java 1.5 programming language, taking extensive advantage of its generic types. The four main components of the framework, as shown in Figure 1, are:

Abstract group: Defines a generic Cayley graph, from which we derive the plug-in classes that implement the specifications of different overlays.

Plug-in modules: The modules, depicted as spheres, are classes derived from `AbstractGroup` specifying the actual group and generator set in use, as defined for instance in Table 1.

ID Distributor: When testing sparse networks, the `IdDistributor` component assigns subsets of the ID space to different nodes.

Network: The network simulator launches test queries which execute according to the group and generator set defined in the plug-in module currently used. It captures success rates as well as edge and generator usages.

As different networks have different identifier types (an integer for Chord, a (level, id)-pair for Butterfly, an array of integers for CAN, etc.), each network specification is parametrized with its identifier type. The requirements for these types are restricted to the existence of a way to iterate through the set of identifiers and the possibility to pick one at random. Even more, the iteration through the list of group elements does not have to follow existing edges (i.e. it does not have to be a hamiltonian path in the graph).

To test a new structured overlay, the user must only derive a new class either from `AbstractGroup` or, if the new overlay is similar to one that is already implemented, from an existing descendent of the `AbstractGroup`. Similarly, a new identifier type may be necessary, or an existing one can be used. Apart from this, the user does not need to modify the network simulator, but only instruct the framework to instantiate the correct class when given some command line parameter.

The simulations can either use a default routing protocol, based on Dijkstra's algorithm to identify the potential of the network, or specify their own routing protocol. While the former is a sort of upper bound on its performance, the latter provides a lower bound, given that no non-standard fault tolerance measures are used, such as caching or replication. Both of these bounds give the designer a significantly better perspective over the capability of the structured overlay to withstand failures in terms of query success rates. For a PDMS implemented on top of such a structure overlay, the results translate into an upper and lower bound on the completeness of the answer to a query.

The default algorithm is implemented in the `AbstractGroup` class, such that any subsequent network architecture derived from this class will have it by default. A specific routing method can then be defined simply by overriding one method in the `AbstractGroup` class.

3. DEMONSTRATION SCENARIO

The demonstration scenario comprises three activities, showcasing the flexibility of the simulator in testing predefined architectures as well as accepting new input.

- First, we will show how to test a predefined network. We provide the set of P2P architectures described in Section 3.1 and the user only has to choose one and launch the process. After a predefined number of random queries, the system will return the query success rate as a function of the percentage of failed nodes in the system.
- Second, we will demonstrate how a given structure can be slightly changed in order to observe the effects of having smaller or larger routing tables or the effects of maintaining some neighbors rather than others. For instance, in Chord, is it always best to maintain neighbors distanced by powers of 2? Why not powers of 3? Why not a cluster of neighbors grouped around powers of 3? And so on. All these scenarios can be easily tested using P^3N.
- Finally, we will show how we can implement a totally new architecture. For this, we must consider an algebraic group of interest and define its elements and operation using the Java programming language. In most cases of interest, such a group is implemented in at most a few hundred lines. In our implementations, the more complicated networks, like the Star, Pancake or Viceroy have around 210 lines, plus an additional 70 lines for the two classes implementing the node identifiers[1].

3.1 P2P architectures tested

Table 1 shows the main eight networks tested in P^3N . However, we have so far implemented and tested over 30 types of networks. Most of them are modifications of the eight main types, as we attempted to answer questions such as: Does using directed or undirected links make a significant difference? Is maintaining more successors in Chord worth the trouble of keeping large routing tables? For a fixed number of neighbors, does their distribution in the network significantly affect the resilience of the network?

3.2 Experiments on a Grid

Though much faster than any other system of its kind, P^3N still needs a non negligible amount of time for each test. Since queries are issued randomly, there is also the problem of statistical significance of the results. This is normally assessed by repeating the tests using the same parameters. P^3N allows us to do that on a local machine, but, if accessible, it can also use a computing grid to run the processes in parallel. In our experiments [12] we use a cluster present at the School of Computing, National University of Singapore. The cluster is operated by a Sun Grid Engine 5.3p6 and consists of 49 nodes using 2 Opteron 2.2GHz CPUs and 2GB RAM, and 42 nodes using 2 P4 2.8GHz CPUs and 1GB RAM.

3.3 User Interface

For ease of use, we have designed a graphical user interface for our simulator. Figure 2 shows sample screenshots of P^3N's graphical user interface. In the simplest execution scenario, where a user simply loads a predefined architecture, the GUI allows the user to indicate the desired number of nodes, their failure rate. The user must also select how nodes should assign themselves portions of the dataspace (random or based on the hamiltonian cycle) and what is the query distribution over the dataspace.

For the second execution scenario, where the underlying graph is tweaked to explore new ways of interconnecting nodes, the user can select/deselect/add new generators to the Cayley graph. For networks where the number of neighbors is fixed (e.g. Viceroy, Cube-Connected Cycles) the user can change the connection patterns by modifying the group operators. Finally, for each network, the routing algorithm can also be changed. These last two types of modifications, operators and routing method, are done via the integrated Java editor in P^3N's GUI. Behind the scenes, the corresponding Java class will be put together, compiled and executed.

4. CONCLUSIONS

We started this project with the aim of identifying a set of measures to follow when choosing or designing a structured overlay for a PDMS. Our strategy was not only to look at graph-theoretic measures, but also to learn from the experience of the past half decade by matching the proposed measures with the existing overlays to discover which of them are indeed significant. In the process of analyzing all these overlays, we have created P^3N: a framework

[1]the Star and Pancake networks share the same type of node identifiers

Table 1: P2P networks and their respective Cayley graph parameters, where $\oplus$ is component wise addition, $\circ$ is permutation composition, $\sigma^l(x)$ is a circular permutation of length l and negation in $\pm$ represents the inverse with respect to the group's operation.

Cayley Group and generating set			
Group set (Cardinality)	**Group operation**	**Generators**	**Overlay**
$\mathbf{Z}_2^n$ (2^n)	$\oplus \bmod 2$	$\{1_i, i = 1, ..., n\}$	**Hypercube**
$\mathbf{Z}_{2^n}$ (2^n)	$+ \bmod 2^n$	$\{2^i, i = 0, ..., n-1\}$	**Chord**
$\mathbf{Z}_{2^n}$ (2^n)	$+ \bmod 2^n$	$\{\pm 2^i, i = 0, ..., n-1\}$	**SkipNet**
$\mathbf{Z}_2^{d \cdot c}$ $(2^{d \cdot c})$	$\oplus \bmod 2^c$	$\{\pm 1_i, i = 1...d\}$	**CAN**
$\mathbf{Z}_n \times \mathbf{Z}_2^n$ $(n2^n)$	$(l,x) * (l',x') = \big(l + l'(mod\ n), x \oplus \sigma^l(x')\big)$	$\{(1, 00...0), (0, 10...0)\}$	**Cycloid**
$\mathbf{Z}_n \times \mathbf{Z}_2^n$ $(n2^n)$	$(l,x) * (l',x') = \big(l + l'(mod\ n), x \oplus \sigma^l(x')\big)$	$\{(1, 00...0), (1, 10...0)\}$	**Butterfly**
$\mathbf{Z}_n \times \mathbf{Z}_{2^n}$ $(n2^n)$	$(l,x) * (l',x') = (l + l'(mod\ n), x + x' \cdot 2^l \pmod{2^n})$	$\{(1,0), (1,1)\}$	**Viceroy**
$\mathbf{S_n}$ $(n!)$	$\circ$	$\{(1,i), 1 < i \leq n\}$	**Star**

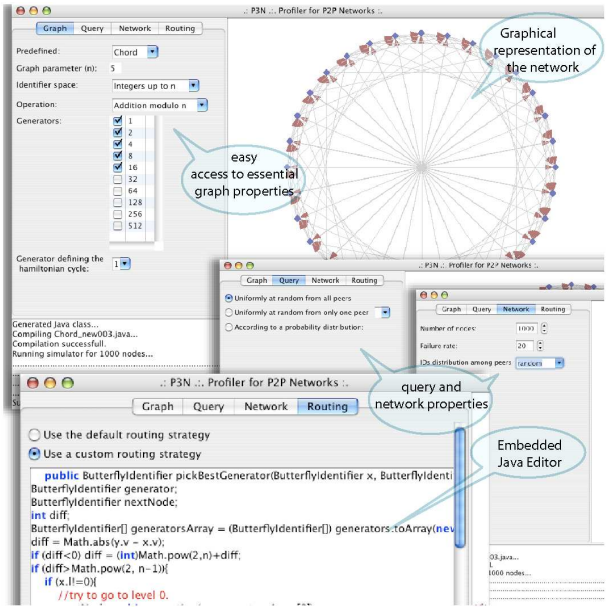

Figure 2: Graphical User Interface sample for P^{3}N

by which a designer can implement and simulate easily different network structures. Within this framework, we can go from idea to simulation results within a day - a significant aid for a system designer.

The demonstration of the system will not only show its potential, but also its limitations. Apart from offering a comparative study of some of the most important P2P network architectures, the demonstration also provides a reference point for discussions on the potential of Cayley graphs as underlying general architectures towards new P2P architectures and systems.

5. REFERENCES

[1] K. Aberer. P-Grid: A Self-Organizing access structure for P2P Information Systems. In *Procs. of ICCIS*, 2001.

[2] K. Aberer, L. O. Alima, A. Ghodsi, S. Girdzijauskas, S. Haridi, and M. Hauswirth. The Essence of P2P: A Reference Architecture for Overlay Networks. In *Proc. of P2P*, 2005.

[3] B. Butnaru, F. Dragan, G. Gardarin, I. Manolescu, B. Nguyen, R. Pop, N. Preda, and L. Yeh. P2PTester: a tool for measuring P2P platform performance. In *Proc. of ICDE*, 2007.

[4] F. Dabek, B. Zhao, P. Druschel, J. Kubiatowicz, and I. Stoica. Towards a Common API for Structured Peer-to-Peer Overlays. In *Proc. of IPTPS*, 2003.

[5] K. Gummadi, R. Gummadi, S. Ratnasamy, S. Shenker, and I. Stoica. The Impact of DHT Routing Geometry on Resilience and Proximity. In *Proc. of SIGCOMM*, 2003.

[6] N. J. A. Harvey, M. B. Jones, S. Saroiu, M. Theimer, and A. Wolman. SkipNet: A Scalable Overlay Network with Practical Locality Properties. In *Proc. of USITS*, 2003.

[7] R. Huebsch, B. Chun, J. M. Hellerstein, B. T. Loo, P. Maniatis, T. Roscoe, S. Shenker, I. Stoica, and A. R. Yumerefendi. The Architecture of PIER: an Internet-Scale Query Processor. In *Proc. of CIDR*, 2005.

[8] H. V. Jagadish, B. C. Ooi, and Q. H. Vu. BATON: A Balanced Tree Structure for P2P Networks. In *Proc. of VLDB*, 2005.

[9] H. V. Jagadish, B.C. Ooi, K.L. Tan, Q.H. Vu, and R. Zhang. Speeding up Search in Peer-to-Peer Networks with a Multi-way Tree Structure. In *Proc. of SIGMOD*, 2006.

[10] J. Li, J. Stribling, R. Morris, M.F. Kaashoek, and T.M. Gil. A Performance vs. Cost Framework for Evaluating DHT Design Tradeoff Under Churn. In *Proc. of INFOCOM*, 2005.

[11] B. T. Loo, T. Condie, M. Garofalakis, D. E. Gay, J. M. Hellerstein, P. Maniatis, R. Ramakrishnan, T. Roscoe, and I. Stoica. Declarative networking: Language, Execution and Optimization. In *Proc. of SIGMOD*, 2006.

[12] M. Lupu, B. C. Ooi, and Y. C. Tay. Paths to Stardom: Calibrating the Potential of a Peer-based Data Management System. In *Proc. of SIGMOD*, 2008.

[13] D. Malkhi, M. Naor, and D. Ratajczak. Viceroy: A Scalable and Dynamic Emulation of the Butterfly. In *Proc. of SPDC*, 2002.

[14] E. Pitoura, S. Abiteboul, D. Pfoser, G. Samaras, and M. Vazirgiannis. DBGlobe: A Service-Oriented P2P System for Global Computing. *SIGMOD Rec.*, 32(3), 2003.

[15] C. Qu, W. Nejdl, and M. Kriesell. Cayley DHTs - A Group-Theoretic Framework for Analyzing DHTs Based on Cayley Graphs. In *Proc. of ISPA*, 2004.

[16] D. Ratajczak and J. M. Hellerstein. Deconstructing DHTs. Technical Report IRB-TR-04-042, Intel Research Berkeley, Nov. 2003.

[17] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A Scalable Content Addressable Network. In *Proc. of SIGCOMM*, 2001.

[18] K. Shudo, Y. Tanaka, and S. Sekiguchi. Overlay Weaver: An overlay construction toolkit. *Computer Communications*, 31(2), 2008.

[19] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A Scalable P2P Lookup Service for Internet Applications. In *Proc. of SIGCOMM*, 2001.