P2PTester: a tool for measuring P2P platform performance

Bogdan Butnaru[1] Florin Drăgan[1] Georges Gardarin[1] Ioana Manolescu[2]
Benjamin Nguyen[1] Radu Pop[2,3] Nicoleta Preda[2] Laurent Yeh[1]
[1] PRiSM Laboratory, Université de Versailles Saint-Quentin en Yvelines, France [2] INRIA Futurs, Gemo group, France
[3] Mandriva, France

1 Introduction

Recent years have seen significant development of peer-to-peer (P2P) file and data sharing systems. Distributed data structures specifically adapted to this setting have been proposed, some of which build on previous research in the field of distributed shared memory. Distributed data structures underlying peer networks are typically called *overlay networks* [3, 6, 9, 14, 16]. The purpose of overlay networks is to provide efficient support for for the crucial *locate* operation: locate a data item satisfying certain criteria, in a potentially large set of peers.

Distributed Hash Tables (DHTs) [4] are a particular class of overlay networks, relying on the abstraction of a single, large hash tables where some data item can be inserted by any peer, associated with some search key.

On top of overlay networks, numerous P2P content management platforms have been developed [1, 8]. The basic functionalities of a P2P content management platform are not fundamentally different from the goals of traditional distributed databases [12]: give to the user *the illusion of a centralized system* while executing complex data management operations in a distributed setting. Thus, what really makes the difference between such systems (and presumably, what will determine their success or failure) is their *performance*. Performance measures are typically based on benchmarks and/or systematic testing suites.

Benchmarks for P2P data management have only started to appear recently [11]. This benchmark is oriented towards information retrieval tasks, not towards database-style queries. Producing a good benchmark of database-oriented processing in P2P is a difficult task, due to the diverse data models (relational [8], XML [1] or RDF [17]) and query languages supported by various systems. Moreover, for the time being, no single data management application has emerged as the most representative in a P2P setting.

A separate technical challenge in assessing P2P system performance is building a performance measurement tool. One reason is the inherent complexity of testing distributed systems; however, this complexity has been tamed in the past, even since the very first day of TPC [2] transaction performance benchmarking. Another reason lies in the apparent diversity of P2P platforms:

- The underlying communication layer ranges from sockets, to Java, through XML message-based interaction.
- The network structure can vary widely, from structured networks (which include DHTs) to unstructured networks [3, 17], hybrid systems [10] etc.

We propose to demonstrate *P2PTester*, the first platform to systematically measure the performance of P2P content management systems. P2PTester is a Java-based application which *wraps around*, and *interfaces with*, any arbitrary P2P system (see assumptions on the underlying system in the next section). P2PTester allows the user to:

- launch, in a supervised manner, the construction of a peer network of controlled size and complexity
- measure the space and time costs of the process of indexing the data from this set of peers, within the P2P network
- measure the space and time costs associated to processing specific queries in each system
- trace the communications spawned from the processing of each specific query or search issued by a peer. This information is useful as a hint on the communication complexity of the system, but may also prove valuable for the system implementor, by helping him trace (and perhaps debug) his platform.

Existing P2P deployment and testing projects [13, 15] provide a standalone implementation (or simulation) of an overlay networks, on top of which users can specify P2P data management applications and gather simple statistics on their behavior, such as e.g. the number of exchanged messages. P2PTester departs from this approach in two major ways. First, P2PTester is meant to be used *in conjunction with (and on top of) existing complete P2P data*

management systems, giving it better chances to measure the real performance of a real, complete deployed platform. Second, P2PTester allows gathering much more (and more detailed) statistics, such as the size of exchanged messages, size of partial results shipped by each peer involved in answering a distributed query etc.

We plan to demonstrate how P2PTester can be used to measure a target application (see Section 3) issued from the context of the eDOS R&D European project [18].

2 P2PTester outline

The goals of P2PTester can be summarized as follows.

Genericity: our first and primary goal is that our system be usable to measure a wide range of P2P platforms.

Scalability: the tester application must be ready-to-deploy at a large scale, since we want to test the performance of distributed systems with a large number of "real" peers. The tester's own overhead should be low, to avoid influencing the overall performance of the system measured.

Modularity: P2PTester must allow performing fine-grained measures of various components of a P2P data management application. Thus, it may be interesting to grasp the performance of a P2P system's locating function *only*, or of its indexing component only, of its distributed query processing operations only etc. This essential feature also enables the testing of complex, hybrid systems, for instance XML or RDF data management layers deployed alternatively on a DHT and on an unstructured network etc.

2.1 P2PTester's interaction with a P2P system

P2PTester is structured in four independent layers, schematically presented in Figure 1.

Communication. The first layer offers a trusted communication infrastructure to exchange messages between peers. For generality, we provide a common interface that must be implemented by any module that provides basic communication functionalities. In the first tester release, we offer a basic socket-based communication module.

Application. Application-specific modules belong to this layer, which includes the peer entity, mainly composed of the indexing, routing and query processing modules. *The indexing module* is responsible for propagating the (system-specific) information used to locate data items and to process queries. *The routing module* exploits such information to locate in the P2P network peers which may contribute to answering a query. Finally, *the query processing module* (if it exists) performs query processing operations, and initiates communications whose aim is to ship data between peers (which may or may not use the routing modules).

Test. The test layer is ***interspersed with and between the other layers***, in order to attain our genericity and modularity goals. Given the variety of platforms and implementation details, we devote particular attention to ensure that

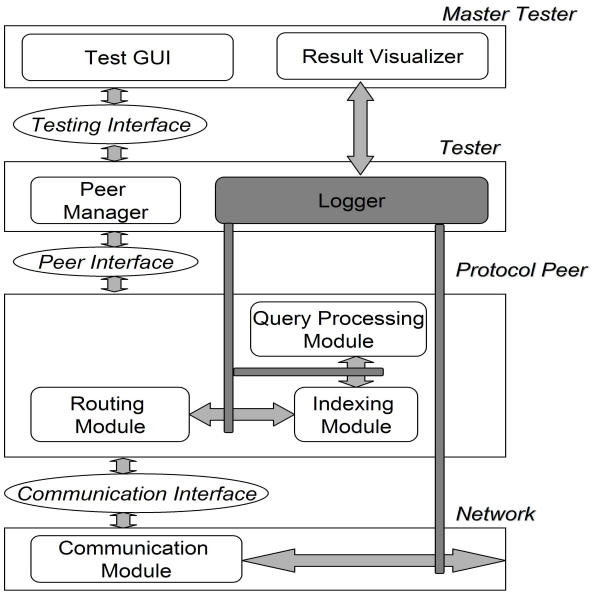

Figure 1. P2PTester Architecture

once defined, a test is general enough to be run on several similar, yet different architectures. The test layer includes a *peer manager*, responsible for launching and stopping peers of the system tested, and a *test control interface*, which receives and processes tests to be run. To gather and interpret (distributed) test results, a *distributed logger* is present in this layer, which records the details of each event in the evolution of the network. Finally, a *test monitor* coordinates the modules in this layer, and automates the production of test results.

Test Generation. To help the user devise and run tests, we provide a (graphical) interface where users can define tests by specifying several basic parameters: the number of peers in the test, the type of peer overlay that must be used, the data sets to be indexed, the queries to be executed, the duration of the test, how the measure results are to be gathered and structured etc. The test results are presented with the help of an interactive visualization tool.

As a supplementary testing help, geared more specifically towards the P2P platform *developers*, *P2PTester* provides a set of basic P2P application modules, which can be plugged together with specific, user-provided data management layers. For instance, P2PTester provides its own DHT implementation. The purpose is not to compete with existing DHTs, but to enable fast prototyping of a running system, and to allow testing the extent to which a system's performance depends on a specific DHT (by trying it alternatively with the one provided by P2PTester).

2.2 P2PTester Interaction with a P2P System

The tester offers a common API for writing P2P test scenarios for a large spectrum of P2P system architectures.

Each method in this API corresponds to (wraps) one of the methods provided by the P2P system under test. Thus, a call to the *join* method, which normally allows a peer to join a P2P system, is intercepted by P2PTester, which logs it, then redirects it to its rightful destination peer, all the while measuring its response time, the communications engendered by the *join* implementation provided by the system etc. Other frequent calls such as *leave* (disconnecting a peer from the network), *publish* (publishing indexing/catalog information in the network), *remove* (withdrawing published data), *leave* (disconnecting a peer from the network), and finally *query* (processing locate or more complex query requests) are intercepted and logged by P2PTester similarly.

P2PTester is deployed as a distributed Java application, as follows. Assume the intended P2P deployment architecture (in the absence of testing) consists of N_L *logical peers* running a given P2P data management software, deployed on N_ϕ *physical peers* (or machines), where $N_L \geq N_\phi$. Deploying the same architecture while testing it with P2PTester involves deploying N_ϕ P2PTester instances, one on each physical machine, and using each instance to start the corresponding logical peers and measure them.

The parameters measured by P2PTester during a test run include the following:

- Number, and size, of messages required for: (i) joining/leaving, (ii) publishing and (iii) querying.
- Size of index/routing data stored at a given peer
- Size of the data currently published in the network
- Query result sizes
- Query processing time, broken down (whenever the underlying system allows it) into:
  - Locate time, or the time it takes to identify the peers in the network holding useful data;
  - Pre-processing time, such as the time to filter out some of the located peer and/or to chose the ones to contact;
  - Processing time, spent in the data transfer and processing operations specific to query processing in the system under test;
  - Post-processing time, such as the time to rank results, aggregate them etc. (in short, all operations that the query peer may perform before presenting results to the user).

3 Demonstration Scenario

In this section, we outline the application scenario, and the P2P systems we plan to demonstrate P2PTester on.

3.1 Target Application

We plan to demonstrate P2PTester on a distributed application dedicated to the collaborative production, testing, integration and distribution of free software, inspired from the eDOS R&D project currently ongoing [18].

Many kinds of users participate in such a system. Most users are *writers*: they contribute successive versions of specific software packages and/or their documentation. Other users are *testers/integrators*: they need to have up-to-date versions of software packages on their sites (possibly automatically pushed by the system as part of a subscription), in order to test and integrate these packages among them. A few sites have a *publisher profile*: periodically, they publish large-scale integrated software suits, together with their documentation etc. Other sites serve as *mirrors*: they only replicate published suits, with the purpose of making them available faster to downloading users scattered all over the world. Finally, a large majority of participants only *download* software, either integrated suites or individual packages under test.

We have chosen this application as representative mainly due to its distributed nature, and to the dynamicity of all peers involved: an arbitrary peer can get involved in such a free software development and exploitation effort, and similarly, any peer can leave at any time. A further interesting aspect of the application is the variety of read/write profiles of the participating peers, which should allow to test the suitability of a P2P system for a large spectrum of real-life applications.

3.2 P2P Systems Under Test

KadoP [1, 19] is a P2P system built on the support offered by a DHT. The system can be used for publishing and querying XML documents. Published documents are indexed using an extension of the DHT API. The KadoP query language is based on tree pattern queries. For testing KadoP we use the tester DHT and we integrate the specific query processing modules in the architecture of the tester.
PIER [8] is a relational query processor adapted to a massively P2P architecture. As KadoP, PIER is based on a DHT structure that is used for indexing and querying data. In PIER queries are expressed in a relational language (e.g. SQL) and transformed in optimized execution plans evaluated over the DHT.

Distributed NIAGARA We plan to offer a P2P evaluation of the NIAGARA system [5] as presented in [7]. The system is built over a Chord implementation of a DHT table. XML documents are indexed in the DHT based on data summaries. The system can handle XPath queries of the form $q = /p_1[p'_1]/p_2[p'_2]/\ldots/p_n[p'_n]\ op\ const$, including wildcards, and the // navigation operator. Queries are answered by locating XML documents satisfying the structural conditions based on the DHT XML index, and value

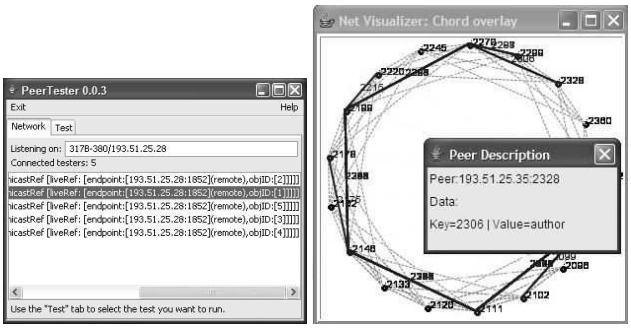

Figure 2. P2PTester screen shots

predicates are matched based on value summaries.

XPEER [8] is a self-administrating P2P XML database system built around a hierarchical overlay of peers with indexing capabilities named super-peers. Each peer makes data available to other peers by publishing XML views of local data, indexed by the super-peer network. XPeer supports the FLWR core of XQuery without respecting the document order in the query results. Queries are executed in two phases: first, using the hierarchical indexing network the relevant sources are discovered; then query execution plans are optimized and executed by directly contacting peers containing relevant data.

The tested systems offer varied, yet comparable, storing and querying functionalities. Our demo will offer a comparative evaluation by testing the sample application deployed over the three systems. For each query/update of the application scenario, and system, we will show: (i) the graph of communications between the peer where the query/update is received, and the other network peers; (ii) the number of overlay network messages involved in processing the query; (iii) the number and total size of data transfers resulting from query processing; (iv) the overall, and individual, processing times incurred at every step during the evaluation.

3.3 Application Deployment

The EDOS application will be deployed on about a hundred logical peers running in our three laboratories, and on laptops at the conference site. All P2PTester instances offer a Web interface, which we will use to show the measured results. Figure 2 shows two sample screen shots of P2PTester: the list of logical peers connected to a P2PTester instance (left), and the distribution of index data in a Chord-like DHT network (right).

4 Conclusion

The current abundance and complexity of P2P architectures makes it extremely difficult to assess their performance. P2PTester is the first tool devised to interface with, and measure the performance of, existing P2P data management platforms. We isolate basic components present in current P2P platforms, and to insert ”hooks” for P2PTester to capture, analyze and trace the interactions taking place in the underlying distributed system. P2PTester allows developers to gather useful feedback on their system, and P2P research and development to profit in general from a thorough, across-the-board comparative analysis.

References

[1] S. Abiteboul, I. Manolescu, and N. Preda. Constructing and querying peer-to-peer warehouses of XML resources (demo). In *ICDE*, 2005.

[2] Transaction Processing Performance Council. http://www.tpc.org.

[3] A. Crespo and H. Garcia-Molina. Routing indices for peer-to-peer systems. In *Proc. Int'l. Conf. on Distributed Computing Solutions (ICDCS'02)*, 2002.

[4] F. Dabek, B. Zhao, P. Druschel, J. Kubiatowicz, and I. Stoica. Towards a common API for structured P2P overlays. In *Proceedings of IPTPS*, 2003.

[5] D. DeWitt, D. Maier, and J. Naughton. http://www.cs.wisc.edu/niagara/.

[6] P. Druschel and A. Rowstron. Past: Persistent and anonymous storage in a peer-to-peer networking environment. In *Proc. 8th IEEE Workshop on Hot Topics in Operating Systems*, 2001.

[7] L. Galanis, Y. Wang, S. R. Jeffery, and D. J. DeWitt. Locating data sources in large distributed systems. In *VLDB*, 2003.

[8] R. Huebsch, J. Hellerstein, N. Lanham, B. T. Loo, S. Shenker, and I. Stoica. Querying the internet with pier. In *VLDB*, 2003.

[9] H. V. Jagadish, B. C. Ooi, and Q. H. Vu. BATON: A balanced tree structure for peer-to-peer networks. In *VLDB*, 2005.

[10] B. T. Loo, R. Huebsch, I. Stoica, and J. M. Hellerstein. The case for a hybrid P2P search infrastructure. In *Proc. 3rd Int'l. Workshop on Peer-to-Peer Systems (IPTPS)*, 2004.

[11] T. Neumann, M. Bender, S. Michel, and G. Weikum. A reproducible benchmark for P2P retrieval. In *Proc. 1st Int'l. Workshop on Performance and Evaluation of Data Management Systems (EXPDB)*, 2006.

[12] M. T. Ozsu and P. Valduriez. *Principles of Distributed Database Systems*. Prentice-Hall, 1991.

[13] J. Pujol, R. Mondejar, H. Tejedor, M. Sanchez, P. Garcia, and C. Pairot. http://planet.urv.es/planetsim.

[14] S. Ratnasamy, M. Handley, R. Karp, and S. Shenker. Application-level multicast using content-addressable networks. In *Proc. 2nd Int'l. Workshop of Network Group Communication (NGC)*, 2001.

[15] K. Shudo. The Overlayweaver project. http://overlayweaver.sourceforge.net/.

[16] I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. *Conf on Applications, Technologies, Architectures, and Protocols for Comp. Communications*, 2001.

[17] I. Tatarinov, Z. Ives, J. Madhavan, A. Halevy, D. Suciu, N. Dalvi, X. Dong, Y. Kadiyaska, G. Miklau, and P. Mork. The Piazza peer data management project. *ACM SIGMOD Record*, 2003.

[18] The EDOS project web site. http://www.edos-project.org/xwiki/.

[19] The KadoP project web site. http://gemo.futurs.inria.fr/projects/KadoP/.