An Extensible, Distributed Simulation Environment for Peer Data Management Systems

Katja Hose, Andreas Job, Marcel Karnstedt, and Kai-Uwe Sattler

Department of Computer Science and Automation, TU Ilmenau,
P.O. Box 100565, D-98684 Ilmenau, Germany

Abstract. Peer Data Management Systems (PDMS) have recently attracted attention by the database community. One of the main challenges of this paradigm is the development and evaluation of indexing and query processing strategies for large-scale networks. So far, research groups working in this area build their own testing environment which first causes a huge effort and second makes it difficult to compare different strategies. In this demonstration paper, we present a simulation environment that aims to be an extensible platform for experimenting with query processing techniques in PDMS and allows for running large simulation experiments in distributed environments such as workstation clusters or even PlanetLab. In the demonstration we plan to show the evaluation of processing strategies for queries with specialized operators like top-k and skyline computation on structured data.

1 Introduction

In recent years research concerning peer-to-peer (P2P) systems has mainly dealt with P2P applications based on environments that are much more sophisticated than those simple file sharing systems that they originate from. Especially Peer Data Management Systems (PDMS) are the focus of current work. They consider the problems arising from peers with different local schemas but appear to be one virtual database.

Research activities in this area do not only include the behavior and topology of a P2P system itself but also query processing strategies, routing indexes[1], schema correspondences[2], etc. All approaches and ideas have to be thoroughly evaluated. For this purpose, simulation environments are built. Such implementations have to meet several requirements in order to allow for reasonable evaluations. In the context of P2P research the primary requirements are: scalability, dynamic behavior, performance, extensibility, arbitrary experimental settings, repeatability, traceability/logging, and control (as mentioned in [3]).

Usually, research groups build their own environments using different data sets, programming languages, application areas and so on. This results in a couple of problems:

- Many environments are implemented rather hastily and have several shortcomings like limited network size, firm index structures and query processing strategies, or even include bugs. Other aspects are simplified and therefore cannot be examined.
- Results gained with different systems are usually not comparable. This leads to unsatisfying evaluations or high reimplementation costs.

Y. Ioannidis et al. (Eds.): EDBT 2006, LNCS 3896, pp. 1198–1202, 2006.

- Existing environments are normally based on network simulators like *ns-2* that are too low-level for our purpose.
- Most environments do not have an intuitive user interface let alone a graphical one. Configuring and using the software is difficult. The output is rather cryptic and does not allow for an intuitive result analysis.

In this paper we present SmurfPDMS (SiMUlating enviRonment For PDMS), a simulator that meets all the requirements mentioned above. Furthermore, it tries to overcome the introduced shortcomings of existing simulators. In the following sections we present its architecture (Section 2) and the aspect of extensibility (Section 3). Afterwards, in Section 4, we present the graphical user interface that makes it easy to operate. Finally, in Section 5 we point out what features we are planning to show at the conference.

2 Architecture

SmurfPDMS logically consists of two parts that we implemented using Java: the *simulation engine* and an *environment for configuration and experimenting*. The latter uses a graphical interface based on Swing and JGraph (*http://www.jgraph.com*) for visualizing configuration parameters, statistics, networks and so on. Moreover, this environment can generate initial settings like the network topology based on user specified parameters like the number of peers and the average number of connections per peer.

The *simulation engine* simulates peers whose states are represented by peer objects. These objects have local caches and message queues to improve query processing, mappings to describe how to translate queries and data into a neighbor's schema, and indexes to describe a neighbor's data. The simulation environment does not only allow for running simulations locally on a single computer, it also gives the opportunity to run simulations involving a couple of computers - improving scalability. Communication between participating computers is carried out by sending and receiving messages using the JXTA [4] protocols. Simulating only one peer per machine enables the simulator to act like a "real" P2P application. In future work we intend to use *PlanetLab* (*http://www.planet-lab.org*) as the underlying framework.

Each participating computer manages several peer objects that can be connected to others residing on other computers. The localization of peer objects (local or remote) does not have any influence on the implementation of other components like query processing strategies. A central clock mechanism allows for gaining comparable results even in resource-limited environments. Apart from the system architecture like introduced above, Figure 1 shows the implementation architecture of SmurfPDMS. It consists of three layers: the graphical user interface, the simulation layer, and the network communication layer. This architecture considers several central concepts: (i) managers that control the simulation or communication, (ii) messages for information exchange, (iii) hierarchies of inherited objects, and (iv) the distinction of participating JXTA peers between multiple *participators* and one *coordinator*. The *coordinator* runs on a designated computer and coordinates the simulation. Its most important tasks are:

- Determine the setup including calculating a network topology, assigning the peers to the participating computers, calculating data partitions, etc.
- Choose queries and determine peers to initiate them

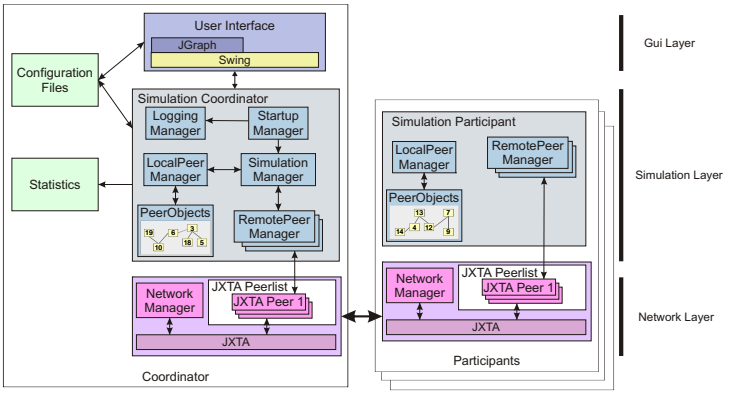

Fig. 1. Architecture

- Determine and choose peers to leave or join the network.
- Simulate communication and processing delays, log the initial setup (in order to achieve repeatability) as well as results and statistics.

3 Extensibility

We achieve extensibility by a modular architecture. Each of the above mentioned classes (manager, strategy, message, peer, etc.) represents a base class. Thus, the logical *simulation engine* of SmurfPDMS can be easily extended. For example, we have derived queries and answers from messages. If we want to introduce a new message type, we just have to derive a new class from the base class. In order to process such messages correctly, we could extend existing strategies or we could as well introduce a new strategy class as an extension of the corresponding base class.

Just like the *simulation engine*, we can also extend the *environment for configuration and experimenting*. SmurfPDMS has several algorithms for initializing a simulation. Assume we want to test different topology construction algorithms or algorithms for distributing the initial data among peers. Then all we have to do is adding an implementation of these algorithms to the concerned classes. All other parameters can remain the same. This allows us to examine the influence of these algorithms under the same environmental settings.

4 Running Experiments

First of all, the simulator has to be configured (*arbitrary simulation settings*). The user has the chance to load configuration files, to reconfigure individual parameters, or to have the simulator compute settings like the network topology, the data distribution, or the query mix. All these parameters and settings can be written to configuration files and reloaded for the next simulation (*repeatability*). Together with the log files we can easily

reproduce and reconstruct entire simulations or reuse partial settings like the topology (*traceability/logging*).

Afterwards, the simulator looks for other JXTA peers in the network. The user can select among them those that he or she wants to participate in the simulation (*scalability, performance*). After having selected the JXTA peers the simulation can be started with the computer that the user currently operates on as coordinator (*control*).

Once the simulation has been started, simulations can be halted, canceled, and executed stepwise. Moreover, the *graphical user interface* provides the user with further features for visualization like messages and their details or peers' local statistics. Figure 2 shows the basic principles of visualizing. The simulation either ends after a user

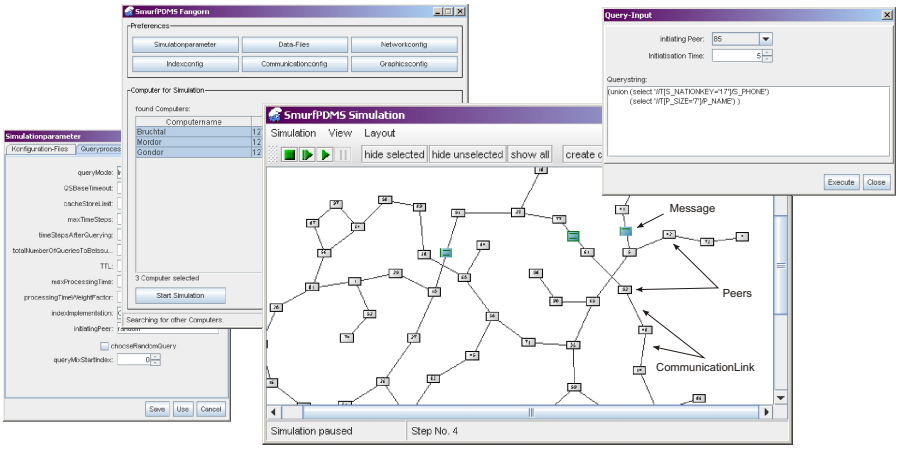

Fig. 2. Simulation window

defined number of simulation steps or when no peer has any actions left to perform. In both situations the coordinator sends the break signal to all participating JXTA peers and thus ends the simulation. At the end of a simulation the global statistics are displayed to the user and a file containing that data is created. These results can be used for creating charts like we did for example in [5] and [6].

5 Demonstration

We plan to focus our demonstration on the aspect of how to use the simulator for conducting experiments with arbitrary settings. This includes presenting algorithms for automatic topology generation as well as algorithms for distributing data among peers. Furthermore, we will show how to formulate queries by hand or how to have them generated automatically by the simulator. We will also show how to compare different query processing strategies by means of their results and collected statistics. Finally, we will show how we can do all this using a graphical user interface for both configuring the simulator and visualizing the results.

References

1. Crespo, A., Garcia-Molina, H.: Routing indices for peer-to-peer systems. In: 22nd Int. Conf. on Distributed Computing Systems. (2002) 23–32
2. Tatarinov, I., Halevy, A.: Efficient query reformulation in peer data management systems. In: SIGMOD'04. (2004) 539–550
3. Buchmann, E., Böhm, K.: How to Run Experiments with Large Peer-to-Peer Data Structures. In: IPDPS'04. (2004)
4. Sun Microsystems, I.: JXTA v2.3.x: Java Programmer's Guide – http://www.jxta.org/docs/JxtaProgGuide_v2.3.pdf (2005)
5. Karnstedt, M., Hose, K., Sattler, K.U.: Query Routing and Processing in Schema-Based P2P Systems. In: DEXA'04 Workshops, IEEE Computer Society (2004) 544–548
6. Karnstedt, M., Hose, K., Stehr, E.A., Sattler, K.U.: Adaptive Routing Filters for Robust Query Processing in Schema-Based P2P Systems. In: IDEAS 2005, Montreal, Canada, 2005. (2005)