PDST: A Peer Database Simulation Tool for Data Sharing Systems

Md Mehedi Masud
SITE, University of Ottawa
800 King Edward Road
Ottawa, Canada
mmasud@site.uottawa.ca

Iluju Kiringa
SITE, University of Ottawa
800 King Edward Road
Ottawa, Canada
kiringa@site.uottawa.ca

ABSTRACT

At present there are many simulation tools developed in order to simulate a peer-to-peer (P2P) system. All the tools are dedicated to P2P content distribution systems, simulate network systems for measuring the efficiency of the networks, and file sharing systems. In the last few years, steady progress has been made in research on various issues related to peer database systems. However, there is no software tool for evaluating a peer database system in a large P2P network. In this paper, we present a software tool that can simulate a peer database system in a large P2P network. The tool provides different facilities, for example, generates peers, databases with synthetic data, acquaintances, and mappings between peers. The tool also provides a general framework for executing queries and updates in a peer database system.

Categories and Subject Descriptors

H.2.4 [**Database Management**]: System-Distributed databases; I.6.3 [**Simulation and Modeling**]: Applications

General Terms

Experimentation

1. INTRODUCTION

In the past few years, the P2P technology has emerged as a new paradigm for distributed data sharing systems. In this technology, all participating computers (or peers) have equivalent capabilities and responsibilities, and exchange resources and services through pair-wise communication, thus eliminating the need for centralized servers. Until now, there are many domain specific P2P systems (e.g. Freenet, Gnutella, SETI@home, ICQ, etc.) that have already been deployed. With a few notable exceptions, currently implemented P2P systems lack data management capabilities that are typically found in a database management system (DBMS), such as query and transaction processing.

A peer database system combines both P2P and database management system functionalities. In the last few years, steady progress has been made in research on various issues related to peer database systems, such as peer database mediation methods [4], coordination mechanisms between peer databases [7], mapping data between peers [1]. The majority of the peer data management systems do not provide experimental results considering a large P2P network. Most of the systems are evaluated with few numbers of peers where databases, acquaintances, and mappings are created manually. Therefore, it takes time for evaluating a system with a large setting. Hence, there is a need to develop a software tool that can serve the peer database research community in order to simulate a system in a large P2P network.

In practice, there has been a clear separation between a simulation model of a P2P system and a real P2P system that operates with real resources (e.g. databases, mappings, queries, etc.). Simulation models are simplified abstractions of real systems that are formalized with the language and modeling concepts that a particular simulation paradigm and environment offers [11]. Particularly, a simulation model in a P2P system is developed in order to validate the scalability of the system. However, sometimes it is necessary to experiment the system with real peers in order to validate the functionalities of the proposed system. The functionalities of a real P2P system is usually provided by P2P applications that are formalized by means of application programming and that include every details of the intended system.

The main objective of this paper is to present a software tool developed for modeling the peer database systems. The tool provides facilities for experimenting peer database systems in a large P2P network. In brief, the tool has the following features.

- combines both the database systems and the P2P functionalities.
- automatically generates databases for each peer and populates databases with data. The tool also generates mappings (coordination rules, mappings [1]) between peers automatically.
- automatically creates acquaintances among peers based on the data in the peers.

The paper is structured as follows. Section 2 briefly describes some of the peer database systems. Section 3 explains the architecture of a peer and describes the framework

SIMUTOOLS 2008, March 03-07, Marseille, France
Copyright © 2008 ICST 978-963-9799-20-2
DOI 10.4108/ICST.SIMUTOOLS2008.2936

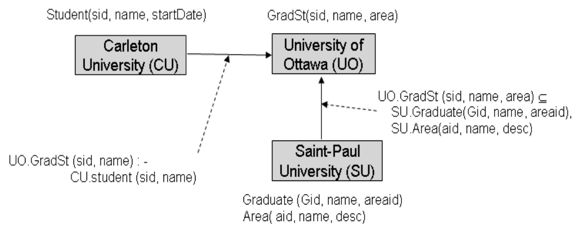

Student(sid, name, startDate)
GradSt(sid, name, area)
Carleton University (CU)
University of Ottawa (UO)
UO.GradSt (sid, name, area) ⊆ SU.Graduate(Gid, name, areaid), SU.Area(aid, name, desc)
UO.GradSt (sid, name) : - CU.student (sid, name)
Saint-Paul University (SU)
Graduate (Gid, name, areaid)
Area(aid, name, desc)

Figure 1: Peers with coordination rules

of the peer database simulation tool (PDST). In Section 4, we show the evaluation of the tool by modeling a transaction processing mechanism. Finally, the paper concludes with some future directions.

2. PEER DATABASE SYSTEMS

A peer database system (PDBS) is a decentralized system which consists of autonomous heterogeneous database sources, where the integration of databases is performed in a peer-to-peer fashion. That is, a network of peer databases is created either directly or indirectly integrated through a path of peer databases that are basically pairwise integrated. The decentralized nature of a PDBS is not transparent to the users. Moreover, a PDBS does gracefully handle the joining or leaving of sources. This decentralized and dynamic approach is in contrast to the traditional distributed and multidatabase systems, where a rigid global schema is used by all the underlying sources. This integration requires a prior knowledge of the schemas of all the databases to be integrated. In the following, we briefly describe some of the peer database systems for which the tool may be useful.

2.1 Piazza System

In Piazza [5], global-as-view and local-as-view mappings [6] are used to relate the schema of each peer with the schemas of other (acquainted) peers. Figure 1 shows an example of the system where peers correspond to major universities, each storing information about students. We assume that we access the network through our local UO peer. The mappings between the UO peer and the CU peer is formed by the global-as-view mapping, and the mappings between the UO peer and the SU peer is formed by the local-as-view mapping. During query answering, a query over the schema of a peer is rewritten to a query over the schemas of acquainted peers, where the nature of the rewriting process depends on the type of mappings available. In general, for global-as-view mappings, the rewriting is performed through view expansion, while for local-as-view mappings, an algorithm for answering queries using views is used. Piazza extends the above rewriting techniques by the combination of both types of mappings during the rewriting process [4]. The placement of mappings between two peers P_i and P_j creates an acquaintance between them. Here, P_j is called the *acquaintee* of P_i.

2.2 Hyperion

Hyperion [3] addresses heterogeneity by means of instance-level mappings. The instance-level mappings are created by mapping tables. This mapping is used where the establishment of coordination rules is not feasible. Situations like this arise when the contents of the acquainted peers, although related, belong in disjoint worlds. As a motivating example, consider the two peer sources: one storing information about genes and another storing information about proteins. The contents of the sources are related, since, in real life, genes produce proteins. However, the contents of the sources cannot be related as such using a coordination rule. In brief, Hyperion introduces a mapping table to associate attributes from the schemas of the two peers, and attribute values that appear in the instances of these schemas. The placement of mapping tables between two peers P_i and P_j creates an acquaintance between them. Here, P_j is called the *acquaintee* of P_i. In terms of query answering, Hyperion offers the mechanisms [2, 10] that rewrite queries between peers by considering the mapping tables.

2.3 coDB

coDB [13] introduces a general logical and computational characterization of peer database systems. In coDB, a network of databases, possibly with different schemas, are interconnected by means of GLAV mappings [14] or coordination rules. coDB mainly analyzed a distributed procedure for the problem of local database update in a network of database peers. In coDB, the answer to a local query may involve data that is distributed in the network, thus requiring the participation of all nodes at query time to propagate in the direction of the query node the relevant data for the answer, taking into account the (possibly cyclic) coordination rules bridging the nodes. Each node can be queried in its schema for data, which the node can fetch from its neighbors, if a coordination rule is involved.

2.4 Discussion

From the above discussion, we observe that PDBSs require databases and some forms of mappings between peers. Further, we notice that each system shows experimental results with a small size of network. For example, Hyperion considered only six to twelve peers, and coDB considered small number of peers to validate their systems. Unfortunately, none of the systems provides experimental results considering a large network since it is practically impossible and time consuming to create databases, mappings, and acquaintances manually for the large number of peers. Therefore, it is necessary to develop a tool that can create databases for as many peers as needed, and generate mappings and acquaintances, automatically. Moreover, the tool should provide the basic functionalities for developing a P2P network (e.g. create peers, communication between peers, provide statistics, etc.). If all the resources and functionalities are ready, then the users only need to incorporate the services they want to evaluate. In this paper, we present such a tool that creates databases for each peer automatically, and builds mappings, acquaintances, and topology for a large P2P network. Moreover, the tool provides the basic underlying framework for evaluating different services (e.g. query, update, etc.) in a peer database system.

3. DESCRIPTION OF PDST

PDST, developed using Java, is a simulation toolkit with a library of Java classes. It is developed using Java programming language for the portability and ease of extensibility. Each peer in PDST is a separate Java thread. The tool can be used for query, update, and transaction processing, which are the important services in a peer database system.

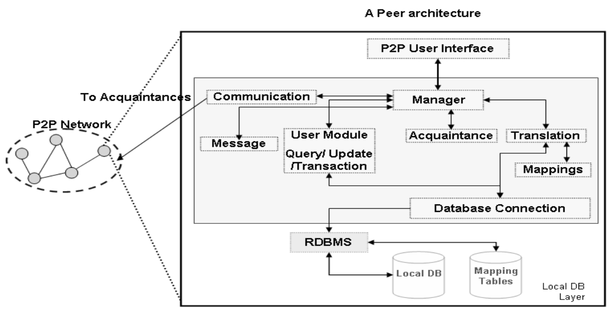


Figure 2: Architecture of a peer

In PDST, each service is implemented as a thread. The tool provides the real-time clock simulation capabilities. The real-time clock simulation, as the name suggests, involves using a real world clock (system clock time obtained using the Java function System.currentTimeMills) for simulating timing [16].

PDST is a message-level event driven simulation framework aimed at modeling peer database systems. It is event based rather than time-driven. Therefore, the simulation time is advanced by the occurrence of events instead of advancing the simulation time in fixed increments that is used in a time-driven simulation system. For example, a query service starts when a peer receives a query from a user or from another peer, and the service finishes when the query is executed in all the peers in the network relevant to the query. Before describing the framework of the tool, we first describe the architecture of a peer that is created by the tool for a peer database system. The architecture of a peer is shown in Figure 2. A peer in the system consists of the following components:

P2P User Interface: through this interface a user submits a service request (e.g. query, update, or transaction). There are two options to submit a service request: GUI or a text file. The file option allows a user to submit a batch of requests for monitoring the behavior of a system with different loads (e.g. queries arrival rate, different update size, number of concurrent transactions, etc.). Through the GUI, user can pose one request at a time to verify the functionalities of a system.

User Module: the user modules are plugged-in by the users according to the types of service the system needs to process. For example, a service may be for processing a query, an update, or a transaction. If the service needs to be executed in the local database, the database connection component is used. For processing the service in acquainted peers, the service is handled by the Manager component.

Manager: each peer has a Manager that handles the execution of services. The Manager takes care of the service requests that are received from the local as well as from remote peers. The Manager interacts with other components that are necessary for processing a service request. For example, if a service needs to be executed locally, it communicates directly with the local database system through the appropriate processing modules (e.g. query, update, or transaction). The service processing modules are externally plugged-in by the users according to the system requirements. If the service needs to be executed remotely, the Manager interacts with other components, such as, Translation, Acquaintance, Communication, and Message components.

Acquaintance: the Acquaintance component provides the acquaintance information to the Manager of the local peer. The acquaintance information is used by the Manager to translate a service request using mappings in terms of the vocabularies of the acquainted peers. The Manager interacts with the translation component for translating a request and the communication component for forwarding a request to the acquaintees of a peer.

Translation: this component translates a local request into a set of remote requests that need to be executed in the acquainted peers. The translation is performed using the coordination rules or mapping tables that exist between a local peer and its acquainted peers. The coordination rules are used for schema-level mappings, and mapping tables are used for data-level mappings. For data-level mappings, Database Connection component is used to connect the database. Users plug-in the Translation component in the system that is to be used by a peer to translate a request.

Communication: this module is used to send and receive messages in the network. In PDST, each peer implements a FIFO queue for sending and receiving messages. When a peer wants to send a message to an acquaintee, the peer enqueue the message to the corresponding queue of the acquaintee. A peer performs dequeue operation for receiving a message from the queue. For receiving message, a peer listens the queue for the incoming message. If a message is found, the appropriate action is performed. The Communication component uses the Message component to construct a message for a request. Later in Section 3.1, we show the message format. Once a message is formed, the Communication component sends the message to the acquaintees. Since the tool implements queue for message exchanges, there is no communication delay in the simulation time. On the other hand, some delays are introduced because of database access time.

Database Connection: this component is used by a peer for accessing the local database system to execute a service request. The tool supports different database systems (MySQL, PostgreSQL, Microsoft Access) connectivity. For a particular database system, the user needs to specify the database system in the environment setup phase. The tool is incorporated with different JDBC driver packages.

Local DB and Mapping Tables: each peer has a local database system and a set of mapping tables. The mapping tables are used to resolve data-level heterogeneity between peers. In the tool, we assume that each peer uses a relational database system (RDBS).

3.1 General framework of PDST

The overall framework of PDST is shown in Figure 3. In the following, we describe different phases to build a peer database network using the framework of PDST.

Environment Initialization: with this phase, users specifies different parameter values to setup a peer database system. For example, number of peers, maximum number of acquaintances per peer, number of relations per peer, etc. The tool provides an interface to the users for setting a system environment. The parameters that are used to build a system using PDST are illustrated in Table 1

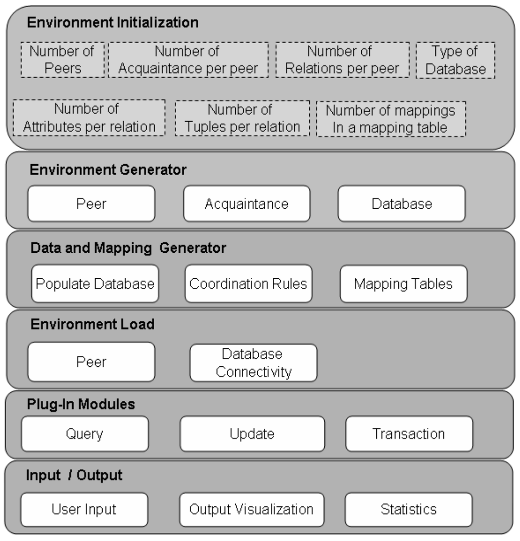


Figure 3: PDST general framework

Parameter	Value
Number of peers	1 to 500
Maximum diameter of a network	10
Number of acquaintances per peer	Min: 2; max: 5
Number of relations per peer	Min: 1; max:3
Number of attributes per relation	Min: 2; max: 4
Number of tuples	Min: 10; max: 100
Domain of the attributes	Integers from 1 to 1000
Number of mapping tables per peer for each acquaintance	Min: 2; max: 8
Number of data mappings in a mapping table	Min: 5; max: 25
Number of coordination rules	Min: 1; max: 3

Table 1: Parameters to build a peer database system

Environment Generator: through this phase, different resources are created for a system to be evaluated. The peer module is used to create peers as defined in the parameter settings. In the system, each peer is implemented as a distinct java thread with the functionalities illustrated in Figure 2. The ids of peers are integer numbers starting from 1. After creating peers, a database is created for each peer. When databases are created, different parameters are used. For example, number of relations, tuples per relation, and attributes. The database module mainly creates the schemas of a database. In the next phase, databases are populated, and the mappings are generated. The acquaintance module creates the acquaintances for each peer. When acquaintances are created, a logical peer database network is created. The acquaintance information is stored in a file. The number of acquaintances created for each peer is based on the parameter values.

Data and Mapping Generator: the databases are populated in this phase. More importantly, it creates coordination rules and mapping tables for each acquaintance. Integer is the domain of the attributes of each relation. Relation names follow the following convention:

$$peerid\_relation(attrbute1, attribute2, \cdots).$$

For example, if a peer "P1" has two relations then the relations are generated as follows:

$$P1\_r1(A1, A2, A3);\ P1\_r2(A4, A5, A6)$$

Consider that peer $P1$ has an acquaintance with peer $P2$. An example of a coordination rule generated from the mapping module is shown below:

$$P1 :: P2 :: P1\_r1(A1, A2, A3) : -P2\_r1(A1, A2, A3)$$

The naming convention of mapping tables is *peerid_m_peerid*. A peer may generate more than one mapping table for a single acquaintee. For example, if a peer $P1$ has two mapping tables for its acquaintee $P2$, then the following mapping tables are generated:

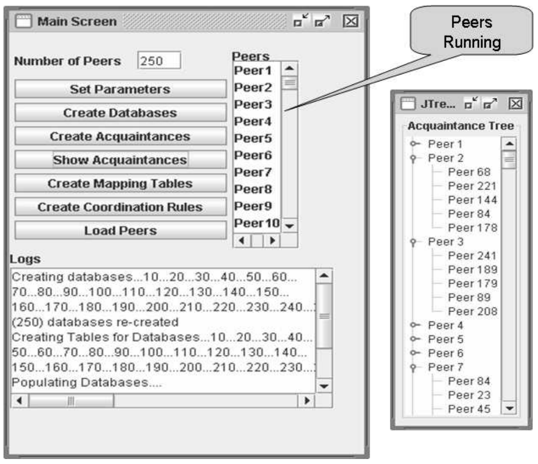
$$P1\_m1\_P2;\ P1\_m2\_P2$$

Environment Load: once the generation of peers, acquaintances, databases, and mappings are completed, this phase loads all the peers in the memory to run. When a peer runs in the system, a peer connects to its database and loads all the resources in the memory. Peers then wait for the events to process.

The simulation starts when a service request is initiated by a peer and needs to be executed in the system. Before forwarding a request to the acquaintees, the initiator constructs a message which is composed of (i) peer ID (PID), the sender of the message, (ii) a unique global identifier ($msgID$) of the message. $msgID$ is formed combining the identifier of the initiator and a message sequence number. We assume that each peer generates a unique sequence number in increasing order for each message it originates. If a peer receives a message with the same $msgID$ as seen before, then the request is rejected, (iii) the message itself, and (iv) type of message. There are three types of message: (a) query (Q), (b) response (RS), and (c) result (R). In order to create a message, the user needs to call the $createMsg$ method of the $message$ class with the required parameters. A message is sent to the peers using the method $sendMessage$ in the $message$ class. There are also other useful methods in the $message$ class.

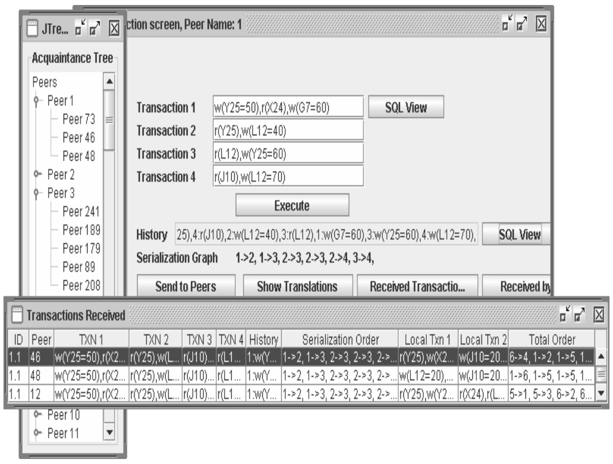
Plug-in Module: in this phase, users are allowed to include their own modules to extend the tools according to the requirements. For example, processing of queries, updates, and transactions.

Input / Output: this phase allows users different interfaces for handling the tool. The tool provides GUI to setup a system environment. Figure 4 shows the interface for creating a system environment. Through the interface, a user can create peers, databases, acquaintances, and mappings. The interface allows users to see the acquaintance graph created for the system. The window in the right side of the main screen in Figure 4 shows the generated acquaintances of a peer. In order to run all the peers in the system, the user needs to click on the 'Load Peers' button. On the right side of the main screen, it shows that the peers are running in the system. In order to initiate a request (query, update, or transaction) from a peer, the user selects a peer from the peers' list. When a peer is selected, a window is activated to work with that peer. For example, Figure 5 shows the input screen of peer "1" for submitting transactions. The screen also provides information to see the list of peers which receives the transactions and the results generated by peers. Users can also see the translated requests (e.g. transactions) for each acquainted peer. Note that each peer is a distinct thread of a class "peer.class". In the following, we present some of the important methods of the "peer.class".



```
Creating databases...10...20...30...40...50...60...
70...80...90...100...110...120...130...140...150...
160...170...180...190...200...210...220...230...240...
(250) databases re-created
Creating Tables for Databases...10...20...30...40...
50...60...70...80...90...100...110...120...130...140...
150...160...170...180...190...200...210...220...230...
Populating Databases....
```

Figure 4: Main screen of PDST



Transaction 1 w(Y25=50),r(X24),w(G7=60)
Transaction 2 r(Y25),w(L12=40)
Transaction 3 r(L12),w(Y25=60)
Transaction 4 r(J10),w(L12=70)
History 25),4:r(J10),2:w(L12=40),3:r(L12),1:w(G7=60),3:w(Y25=60),4:w(L12=70),
Serialization Graph 1->2, 1->3, 2->3, 2->3, 2->4, 3->4,

ID	Peer	TXN 1	TXN 2	TXN 3	TXN 4	History	Serialization Order	Local Txn 1	Local Txn 2	Total Order
1.1	46	w(Y25=50),r(X2...	r(Y25),w(L...	r(J10)...	r(L1...	1:w(Y...	1->2, 1->3, 2->3, 2->3, 2->...	r(Y25),w(X2...	w(J10=20...	6->4, 1->2, 1->5, 1...
1.1	48	w(Y25=50),r(X2...	r(Y25),w(L...	r(J10)...	r(L1...	1:w(Y...	1->2, 1->3, 2->3, 2->3, 2->...	w(L12=20),...	w(J10=20...	1->6, 1->5, 1->5, 1...
1.1	12	w(Y25=50),r(X2...	r(Y25),w(L...	r(J10)...	r(L1...	1:w(Y...	1->2, 1->3, 2->3, 2->3, 2->...	r(Y25),w(Y2...	r(X24),r(L...	5->1, 5->3, 6->2, 6...

Figure 5: A user input screen for peer 1

getPeerName(): returns the name of the currently selected peer.

getAcquaintances(): returns the names of all the acquainted peers of a peer.

getDBConn(): returns the database connection object of a peer.

listenPeerQueue(): continuously listen the queue for any incoming message. This a separate thread running in each peer. If any message is detected, the peer processes the request.

sendMessage(): sends message to the acquaintees.

4. EVALUATION

The first objective of the evaluations is to show the time required to build a peer database system with different size of networks. The time includes creation of peers, databases, acquaintances, and mappings. The second objective is to evaluate the tool by modeling an existing approach of query, update, or transaction processing. In this paper, we modeled the transaction processing system proposed in [9]. For all the evaluations, we use the tool in a single Windows XP machine with Intel Pentium 4 CPU 3.40GHz and 1GB of RAM. Since the tool is used in a single machine, there is no communication delay in the simulation time. On the other hand, some delays are introduced because of database access time. A future goal is to evaluate the tool considering proper network factors, for example, taking into account network communication delay and the amount of exchanged messages and data with different database systems. In our setting, we consider that each peer is connected to a MySQL 5.0 database, and all the peers have an equal average number of acquaintances. We provide a summary of the configuration parameters in Table 2. The results of creating different peer database systems are shown in Figure 6. We observe from the result that the time increases sharply when the number of peers increases. Through the analysis of the evaluation, we notice that 90% of the time is required to create databases and generate mappings.

Parameter	Value
Number of peers	100-500
Number of acquaintances per peer	3
Number of relations per peer	1
Number of attributes	4
Number of tuples	25
Number of mapping tables per peer for each acquaintance	4
Number of data mappings in a mapping table	8

Table 2: System parameters

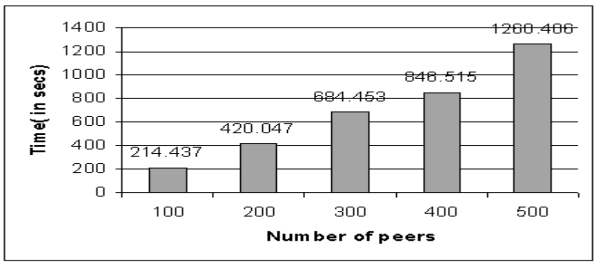


Figure 6: Time to generate peers with resources

We then use the tool to model a transaction processing system. For the evaluation, we consider only the time factor since we want to see how much time is required to process a request in the system. We consider transactions with different size. The size refers to the number of operations in a transaction. The transactions are chosen in such a way that the transactions are executed in each peer. The results of the experiment are shown in Figure 7. Consider a network size of 500 peers. From the result, we observe that the execution time of transactions with size 2 and 10 are 40.375 sec and 45.862 sec, respectively. The result shows that the execution time does not increase rapidly with the size of transactions. However, we notice from the result that there is a sharp increase of execution time of transactions. The result points that when a transaction has more write operations than read operations, then the execution time takes more in the system.

5. RELATED WORK

Authors in [21] discuss the present situation with respect to simulation usage in P2P research, and present the state of the art of P2P simulations. We found that most of the simu-

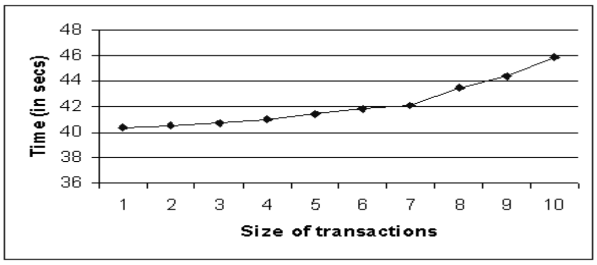


Figure 7: Execution time of transactions

lators are dedicated for content distribution and file sharing systems. In order to evaluate a peer database system, we need different resources, for example, databases, mappings between peers, and acquaintances. The existing tools do not support or provide these facilities. Moreover, tools should support database related actions (e.g. query, update, and transaction). Our goal is to present a tool that can be used to evaluate a peer database system which is unable with the existing tools. We describe some of the P2P simulators in the following.

NS-2 [17] is a popular network simulator that best suits for simulating packet switched networks and small scale networks. Adding new modules is not straightforward, because of it's complex module structure [18].

The simulator [19] is specialized to file-sharing simulations. It mainly models the content distributions, query activity, download behavior etc. Simulations proceed in query cycles representing the time period between issuing a query and receiving a response. Similar to our approach, queries are passed into a queue and handled on FIFO basis.

NeuroGrid [20] is single-threaded, Java-based simulator designed for supporting comparative resource search simulations between FreeNet, Gnutella, and NeuroGrid systems. It is basically single thread and non-parallel.

6. CONCLUSION AND FUTURE WORK

In this paper, we presented a tool for modeling peer database systems in a large P2P network where mappings between peers are established either by coordination rules or instance-level mappings. The tool supports the real database functionalities, for example processing queries, updates, and transactions. We have modeled the transaction processing mechanism [9] using the tool and presented some experimental results.

A future goal is to evaluate the tool considering proper network factors, for example, taking into account network contention and number of exchanged messages or data. Another goal is to investigate the query processing of the approaches presented in [5, 2, 12, 13] and show their comparison results. Moreover, we intend to extend the framework for supporting web interface, so that users can use the tool from any where through the internet.

7. REFERENCES

[1] A. Kementsietsidis, M. Arenas, and R.J. Miller. Mapping Data in Peer-to-Peer Systems: Semantics and Algorithmic Issues. In *SIGMOD*, 2003.

[2] A. Kementsietsidis and M. Arenas. Data Sharing Through Query Translation in Autonomous Sources. In *VLDB*, pages 468-479, 2004.

[3] M. Arenas, V. Kantere, A. Kementsietsidis, I. Kiringa, R.J. Miller, and J. Mylopoulos. The Hyperion Project: From Data Integration to Data Coordination. In *SIGMOD RECORD*, 2003.

[4] A. Halevy, Z. Ives, D. Suciu, and I. Tatarinov. Schema mediation in peer data management system. In *ICDE*, 2003.

[5] A. Y. Halevy, Z. G. Ives, J. Madhavan, P. Mork, D. Suciu, and I. Tatarinov. The piazza peer-data management system. In *IEEE Transactions on Knowledge and Data Engineering*, Vol. 16, no. 7, 2004.

[6] A. Y. Halevy. Answering Queries Using Views: A Survey. In *The VLDB Journal*, vol. 10, no. 4, 2001.

[7] L. Serafini, F. Giunchiglia, J. Molopoulos, and P. Bernstein. Local Relational Model:A Logocal Formalization of Database Coordination. Technical Report, University of Trento, 2003.

[8] M. Masud, I. Kiringa, and H. Ural. Update Propagation and Data Synchronization in Instance Mapped Peer Data Sharing Systems. In *InterDB*, 2007.

[9] M. Masud and I. Kiringa. Acquaintance Based Consistency in an Instance-Mapped P2P Data Sharing System During Transaction Processing. In *CoopIS*, 2007.

[10] M. Masud, I. Kiringa, and A. Kementsietsidis. Don't Mind Your Vocabulary: Data Sharing Across Heterogeneous Peers. In *CoopIS*, 2005.

[11] D. Hildebrandt, L. Bischofs, and W. Hasselbring. RealPeer-A Framework for Simulation-Based Development of Peer-to-Peer Systems. In *PDP*, 2007.

[12] W. Siong Ng, B. Chin Ooi, K. Tan, and A. Zhou. PeerDB:A P2P-based System for Distributed Data Sharing. In *ICDE* , 2003.

[13] E. Franconi, G. Kuper, A. Lopatenko, and I. Zaihrayeu. The coDB Robust Peer-to-Peer Database System. In *SEDB*, 2004.

[14] M. Lenzerini. Data Integration: A Theoretical Prespective. In *PODS*, 2001.

[15] W. Yang and N. Abu-Ghazaleh. GPS: a general peer-to-peer simulator and its use for modeling BitTorrent. In *MASCOTS* , 2005.

[16] R. S. Nair, J. A. Miller, and Z. Zhang. Java-based query driven simulation environment. In *WSC* , 1996.

[17] NS-2. http://www.isi.edu/nsnam/ns/.

[18] N. Kotilainen, M. Vapa, T. Keltanen, A. Auvinen, and J. Vuori. P2PRealm - peer-to-peer network simulator In *CAMAD*, 2006.

[19] M. Schlosser and S. Kamvar. Simulating a P2P File-Sharing Network In *Workshop on Semantics in Grid and P2P Networks*, 2002.

[20] S. Joseph and T. Hoshiai Decentralized Meta-Data Strategies: Effective Peer-to-Peer Search. In *IEICE Transaction on Communication*, vol. E86-B, no. 6., 2003

[21] S. Naicken, B. Livingston, A. Basu, S. Rodhetbhai, I. Wakeman, and D. Chalmers. The state of peer-to-peer simulators and simulations. In *ACM SIGCOMM Comp. Comm. Review*, vol. 37, no. 2, April 2007.