BALLS Simulator: Evaluator of a Structured Peer-to-peer System with Integrated Load Balancing

Viet-Dung Le
Department of Computer Science
and Operations Research
University of Montreal
C.P. 6128, Succ. Centre-Ville,
Montréal, (Québec)
Canada H3C 3J7
Email: levietdu@iro.umontreal.ca

Gilbert Babin
Information Technologies
HEC Montréal
3000, ch. de la
Côte-Sainte-Catherine,
Montréal, (Québec)
Canada H3T 2A7
Email: Gilbert.Babin@hec.ca

Peter Kropf
Institute of Computer Science
University of Neuchâtel
Rue Emile Argand 11,
CH 2000 Neuchâtel,
Switzerland
Email: Peter.Kropf@unine.ch

***Abstract*— Simulation is an efficient way to evaluate new peer-to-peer models. It requires two implicit properties: large scale and high dynamicity. In the context of our work that proposes a peer-to-peer structure based on partitioning a de Bruijn graph and its load balancing algorithms, we developed a simulator for evaluation purposes. This paper introduces a three-layer architecture of the simulator. This architecture allows to support simulations in two modes: centralized (where all peers are simulated on one physical machine) and decentralized (where the peers run on separate machines communicating through the underlying network).**

I. Introduction

The recent appearance of structured peer-to-peer (P2P) systems (e.g., [1]–[6]) proposed interesting solutions to the routing problem resulting from the lack of a centralized control. Such a system manages objects by distributing the responsibility of the object keys (usually produced by hashing the object id) over the available peers. The peers connect according to the set of keys each one holds.

The distribution of the key responsibility in a structured P2P system introduces a problem of load unbalance, which critically affects the system's performance. In this context of load balancing in a structured P2P system, we have proposed a P2P structure and algorithms for the index management load balancing and the storage load balancing. The index management load represents the rate of bandwidth consumption on each peer for routing, whereas the storage load implies the usage of each peer's resources for object accommodation. The proposed structure (namely BALLS - Balanced Load Supported P2P Structure) partitions a de Bruijn graph layered over the P2P network. It achieves the simultaneous index management load balancing and storage load balancing by separating the peer id, object key, and storage location. Details of this system were described in [7].

The practical performance of our proposed P2P system must of course be evaluated. A P2P system usually displays two properties: large scale and high dynamicity. Therefore, using simulators is an efficient method for evaluating P2P systems.

There exist P2P simulators. Peersim [8] supports simulation of several P2P protocols as part of the BISON (Biology-Inspired techniques for Self-Organization in dynamic Networks) project. The simulator manages an array of peers and performs the peer operations sequentially. It provides the following main Java classes and interfaces: Node, CDProtocol, Linkable, Observer, and Dynamics to model respectively the peer, protocol, network, data collection, and dynamicity.

Another P2P simulator is the NeuroGrid simulator [9], which was originally designed to compare the Freenet, Gnutella, and NeuroGrid [10] systems. This simulator supports abstract classes intended to be generic for different P2P simulations. The classes Network, Node, Document, and Keyword represent, respectively, the P2P network, peer, document, and keyword. Nodes interact through instances of class Message.

The p-sim simulator [11] was developed in C. It is based on the event-driven simulation technique. The supported events (including Peer Arrival/Departure, Search Query, and Evaluation) have a time-stamp. The simulation executes the events sequentially according to the order of their time-stamp. The simulation components include topology, peer dynamics, file search protocol, and evaluation metrics.

He et al. [12] introduced a framework for P2P simulation that takes into account the details of the underlying network. Each peer simulated by this framework consists of three layers: Network Simulator (providing packet transfer service), PeerAgent (supporting message forwarding and processing), and PeerApp (performing application actions such as query processing and peer relationship maintenance).

FreePastry [13] is a Java implementation of the Pastry network for deployment in the Internet. The latest version supports two transport communication modes: Direct and Socket. The Direct mode emulates the whole system with Pastry nodes in one Java VM without a physical network. The Socket mode, however, uses TCP for communication (except liveness checks) between Pastry nodes.

The ongoing effort to provide a diversified evaluation of our proposed P2P structure and load balancing algorithms requires centralized simulations (in one physical machine) and decentralized simulations (in a real network). Centralized simulations evaluate the system with a high number (thousands) of

peers and high dynamicity. Decentralized simulations, on the other hand, take into account the real-time and parallel factors. The above simulators (except FreePastry) are generic for evaluating different P2P systems but do not support decentralized simulations. FreePastry can answer this requirement due to its two communication modes. It also includes a common API package [14] for developing different structured P2P overlays. However, the API is suitable for P2P structures having peer ids belonging to the object key space (e.g., [3]–[5]). Our P2P structure separates the peer id from the object key to enable index management load balancing. Therefore, basing our simulator on this API is not advantageous.

We therefore implemented the BALLS simulator for our proposed P2P system that can perform simulations under both modes, centralized and decentralized. The present paper describes the architecture of the BALLS simulator. It includes three layers:

1) the network layer provides functions of peer communication and system configuration. This layer consists of two parts supporting, respectively, the centralized and decentralized simulation modes;
2) the peer layer provides functions that support peer actions such as joining, departing, routing, load balancing. This layer is common for the two simulation modes;
3) the evaluation layer, on top of the two others, provides evaluation functions. Like the network layer, the evaluation layer is divided into two parts because the evaluation strategies are different across the simulation modes.

The three-layer architecture also facilitates the extension of the BALLS simulator for other P2P structures and protocols. We can do it by simply extending and developing classes in the peer layer without affecting the other layers.

The rest of this paper is organized as follows. Section II outlines the objectives and architecture of the BALLS simulator. Section III summarizes the BALLS structure and algorithms as the simulated objects. Section IV goes into details of the simulator layers. Discussion in Section V validates the simulator. Finally, the last section concludes the paper.

II. Generic architecture of the BALLS simulator

Evaluating the BALLS structrure and the load balancing methods requires a large-scale and dynamic environment, which involves a large number (e.g., thousands) of peers that continuously join and leave. Running such a system in a real network, where each peer occupies a physical node, is very expensive and often only possible in closed experimental environments. As an alternative we use centralized simulations in which all peers operate on the same machine. In such a way, evaluation of a large and dynamic system becomes feasible and efficient. Unlike centralized simulations, a decentralized simulation implements the peers on separate machines communicating through the underlying network. Evaluation employing decentralized simulations allows us to take into account the effect of real network and real-time factors. Therefore, one objective of the BALLS simulator is to support both centralized and decentralized simulation modes.

The decentralized simulations supported by the BALLS simulator are not real experiments yet, although they run on a real network. The objects and routing requests in simulations are generated by scenarios selected a priori but not by real users. If in a real experiment, the peer properties (e.g., storage capacity, traffic capacity) are set based on the host machine configuration, in simulations, they are chosen according to simulation setting. Although being a simulator, this framework is first step of the future implementation of the BALLS system.

The evaluation of a P2P system usually considers the routing cost and the maintenance cost. The routing cost means the number of hops involved in routing. Whereas, the maintenance cost denotes the number of messages exchanged to maintain the P2P structure when a peer arrives or departs. The maintenance cost is usually related to the degree of the peers. The BALLS simulator must be able to explore the routing cost, maintenance cost, and peer degree.

On the other hand, evaluating the proposed load balancing methods requires the simulator to provide functionalities that measure the loads (index management load and storage load) and compute the corresponding overloads (see Sect. III for the definition). This observation allows us to quantify the load balancing performance.

In order to fulfill the above objectives, we design the BALLS simulator in three layers: network, peer, and evaluation. The **network layer** is responsible for system configuration and peer communication. These tasks function differently across the two simulation modes. In centralized simulations, the whole system with multiple peers can be initiated at the same time on one physical machine. The peers communicate by direct access to each other. However, in decentralized simulations, each peer is initiated on a separate machine. The peers therefore communicate through network services provided by the underlying network. For this reason, we divide the network layer into two parts: centralized and decentralized. Each part of this layer supports the network functionalities for the corresponding simulation mode.

The **peer layer** provides the operational functions of a peer: joining, departing, routing, object managing, and load balancing. Since the functionality requirement of a peer does not change across the centralized and decentralized simulations, we design this layer commonly for the two simulation modes.

The **evaluation layer**'s task consists of conducting simulations and observing the simulation data. The two simulation modes require different evaluation strategies. In centralized simulations, global measurements can be efficiently computed by local functions. Whereas, they become more difficult in the decentralized mode where additional network communications are needed to perform aggregate operations. Consequently, this layer involves two parts: centralized and decentralized. The centralized part supports evaluation of the peer degree, routing cost, arrival cost, departure cost, index management load balancing, and storage load balancing. To ensure the computational efficiency, the decentralized part of the current simulator version only evaluates the load balancing methods.

III. BALLS AND LOAD BALANCING ALGORITHMS

This section briefly introduces the BALLS structure and load balancing algorithms (see more in [7]), the evaluated objects of the simulator. It helps the reader understand the simulator's requirements and concepts that are related in later sections. We define the **index management load** of a peer as the bandwidth consumed in a unit of time for the routing task. The **storage load** of a peer is the total size of the objects it actually stores. To enable the simultaneous index management load balancing and storage load balancing, we proposed the BALLS structure that separates the concerns of peer id (address), object key, and storage location.

Due to its fascinating properties (low node degree and diameter), the de Bruijn graph appears extensively in the literature of interconnection networks (e.g., [15]–[17]). We apply the de Bruijn graph in the P2P structure for the goals: low-cost maintenance, efficient routing and load balancing.

The BALLS partitions a binary de Bruijn graph in the P2P network. The key space is identical to the de Bruijn node id space. Each peer p holds (is responsible for) a non-empty key interval, denoted $[p.b, p.e]$. Any two peers p and q connect if $[p.b, p.e]$ and $[q.b, q.e]$ are connected by a de Bruijn arc or adjacent in the circular key space. Such p and q are called neighbours. A peer maintains a *neighbour list* consisting of a triple $(q.a, q.b, q.e)$ for each neighbour q, where $q.a$ is the address (e.g., TCP/IP) of q.

The routing follows appropriate routing paths in the de Bruijn graph. Because of the de Bruijn graph's low diameter, the routing efficiency is guaranteed. The BALLS applies simple peer arrival and departure protocols. When a new peer p joins, it lookups for the root of a random key via a known peer. The found root splits its key interval and transfers one half to p. When a peer q departs, it selects among its two ring neighbours (i.e., peers holding a numerically adjacent key interval) the one holding the shortest key interval to transfer $[q.b, q.e]$. If the transfer is done, q notifies its neighbours about the departure and quits after receiving their confirmation.

The index management load balancing aims at minimizing the system's global overload. Given a peer p with index management capacity C_p and index management load T_p, its overload is $O_p = (T_p - C_p + |T_p - C_p|)/2$. Whenever $O_p > 0$, p transfers some portion of $[p.b, p.e]$ to an appropriate ring neighbour (say q) so as to reduce the combined overload $O_p + O_q$ the most. The separation between peer id and object key means that p can dynamically change either $p.b$ or $p.e$. It thus enables the key transfer among peers. The reduction of $O_p + O_q$ contributes to the minimization of the global overload.

Each peer p has a space boundary D_p to limits the storage load S_p. In addition to the storage space, object accommodation also spends network bandwidth for object access and migration. For a proper operation, we define another boundary namely the desired storage capacity $\overline{D}_p$, with $\overline{D}_p \leq D_p$. S_p never exceeds D_p but can temporarily exceed $\overline{D}_p$. The storage overload is defined as $W_p = (S_p - \overline{D}_p + |S_p - \overline{D}_p|)/2$. The storage load balancing aims at minimizing the global storage overload while ensuring $S_p \leq D_p$ on every peer p.

Our solution is to separate the object key and object location. An object can reside on a peer other than its root (the peer responsible for the object's key). To keep reference, an object and its root maintain pointers to each other. This separation allows objects to migrate among peers. As long as $W_p > 0$, the storage load balancing algorithm on peer p exchanges appropriate objects with another peer (say q) to minimize their combined overload $W_p + W_q$. Obviously, this exchange contributes to the global overload minimization.

IV. THE LAYERS OF THE BALLS SIMULATOR

Section II gave an overview of the BALLS simulator's architecture, which comprises three layers: network, peer, and evaluation. The present section describes the layers in more details. The implementation of this architecture uses Java. Figure 1 depicts the principal classes supported as well as their relationship. In centralized simulations, the peer layer operates with the centralizied parts of the network and evaluation layers. In decentralized simulations, the peer layer and the decentralized parts of the other layers are used.

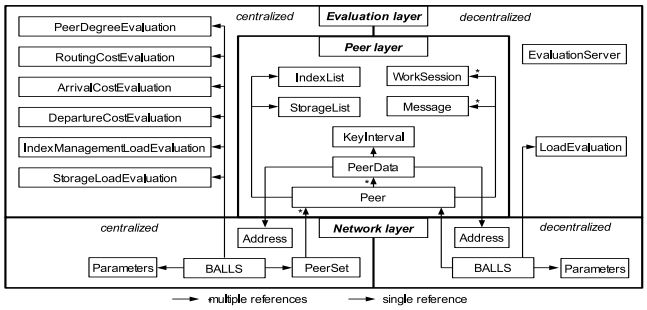

Fig. 1. The simulator architecture

A. Network layer

As we have introduced, this layer consists of two parts: centralized and decentralized. While the centralized network layer constructs the whole system on one machine and realizes communication using direct access, the decentralized network layer initiates only one peer at a time and supports communication via the real network.

The centralized network layer supports the following classes: BALLS, Parameters, PeerSet, and Address. Class BALLS represents the P2P system. It keeps an instance of Parameters (for the configuration) and an instance of PeerSet (for the access to the peers).

The Parameters class is responsible for reading/setting the parameters from/to a configuration file and keeping the parameter values to conduct the system operation. Principal examples of parameters are: the key length, initial number of peers, arrival and departure rates, and simulation scenarios.

The PeerSet class is an array of Peers (representing peers) sorted in ascending order of addresses. This class provides the peer insertion/deletion (in the array) and access functions

using binary search. The peers run in a circular schedule. The peers' execution order follows their order in the array.

A complete execution round of all peers constitutes a simulation cycle. In the centralized mode, the simulator does not rely on the computer clock to calculate the time because the load of concurrent processes affects the operation speed. Instead, we use simulation cycles for the time calculation.

The Address class represents the peer address, which is an integer in the centralized mode. We use it as the key for sorting the peers in the array and for binary search. Sending a message to a peer involves locating it in the array using the given address and inserting the message to its message queue.

The decentralized network layer supports classes: BALLS, Parameters, and Address. The BALLS class here represents the system aspect of one peer. The Parameters class provides similar functionality to that of the centralized part. However, since it configures one peer, the address (TCP/IP) of the peer must be specified. Moreover, it allows to set the address of the EvaluationServer (which we will introduce in Sect. IV-C).

The Address class now specifies the TCP/IP address of the corresponding peer. In communication, the BALLS objects of the peers exchange messages using TCP/IP and push them to the message queue of the receiving peers.

B. Peer layer

The peer layer implements the classes supported for a peer's functionality. The classes are common for both simulation modes. The principal classes includes: Peer, PeerData, KeyInterval, IndexList, StorageList, Message, and WorkSession.

The role of class Peer is to perform the peer activities. Each execution cycle of a peer consists in treating the new messages, performing the ongoing protocols (WorkSessions), and generating random actions (e.g., start of routing, insertion of new objects) depending on the simulation requirement.

Class PeerData represents the triple $(p.a, p.b, p.e)$ of a peer p. Peer p uses a PeerData instance to identify its own $(p.a, p.b, p.e)$ and a set of PeerData instances for the neighbour list. Class KeyInterval represents a key interval. It also provides the necessary calculations in the circular de Bruijn node space, e.g., union, intersection, neighbourhood verification, distance.

In order to manage the objects, a peer maintains an instance of class IndexList and an instance of class StorageList. An IndexList is the list of the pointers to the objects under the current peer's responsibility. It sorts the items in ascending order of (object key, object id). The StorageList is the list of the objects stored on the current peer. It sorts the items based on the object id. The lists are sorted to support binary search.

We develop the Message class for peer communication. The current BALLS simulator version supports 24 extensions of Message to use in different protocols. The principal message types includes: routing message, joining message, departure message, interval notification, interval transfer message, storage transfer message, storage notification, and root notification. In the decentralized simulation mode, the peer periodically sends a message called load notification to the EvaluationServer (see Sect. IV-C) for aggregate measurements.

Class WorkSession implements a protocol on a peer. The present simulator version develops 13 extensions of WorkSession. Principal work sessions includes: routing, joining, departure, interval transfer, interval notification, index management load monitoring, index management load balancing, storage load monitoring, and storage load transfer. Each peer keeps instances of the ongoing work sessions. An execution cycle resumes the sessions by calling their continueSession method. When a session ends, the peer frees the its instance.

C. Evaluation layer

The evaluation layer supports measurement of different metrics so as to evaluate the system performance. As explained before, this layer is divided into two parts: centralized and decentralized to server different evaluation strategies.

The centralized evaluation layer provides the evaluation classes: PeerDegreeEvaluation, RoutingCostEvaluation, ArrivalCostEvaluation, DepartureCostEvaluation, IndexManagementLoadEvaluation, and StorageLoadEvaluation. The first four classes measure the topology performance. The last two, however, evaluate the load balancing methods. An evaluation instance starts with the simulation, calculates the desired metrics at intended moments, and saves the metric values for later analyses.

PeerDegreeEvaluation measures the average and distribution of peer degree in function of system size. A measurement takes place when an arrival or departure of peer finishes successfully. RoutingCostEvaluation measures the average routing cost. It registers the number of hops taken by each routing when the routing finishes. ArrivalCostEvaluation and DepartureCostEvaluation measure the number of messages sent in, respectively, peer arrival and departure in function of system size. The number is counted by the joining and departure sessions.

IndexManagementLoadEvaluation and StorageLoadEvaluation evaluate the load balancing methods. They measure the corresponding global overloads, loads, and capacities along the simulation life. To enhance the evaluation quality, the simulator allows to set simulation scenarios. A scenario defines the moments to start or stop load balancing functions, the capacities' utilization, the key popularity dynamicity, etc.

This layer also supports different patterns of load and capacity distribution so as to achieve the most realistic simulations. As suggested by numerous research (e.g., [18]–[21]), the Zipf distribution is appropriate for the peer's capacities and the routing skewness while the log-normal distribution is appropriate for the object size. Table I shows the distribution patterns supported by the BALLS simulator.

Distribution	Random	Zipf	Log-normal
Index management capacity	√	√	
Routing source probability	√	√	
Routing target probability	√	√	
Desired storage capacity	√	√	
Object size	√		√

TABLE I

SUPPORTED DISTRIBUTION PATTERNS

The decentralized evaluation layer of the current BALLS simulator version supports the index management load balancing and storage load balancing evaluations. In order to facilitate aggregating measurements, we use a process called EvaluationServer. The EvaluationServer runs as a server that aggregates measurement values from the peers, periodically calculates the desired global measurements (including the overload, load, and capacity), and writes the results to files.

In the peer part, we group both supported evaluations in one class called LoadEvaluation. It periodically sends a load notification (Sect. IV-B) containing the current index management load, storage load, and the peer's capacities to the EvaluationServer. With the limitation of current decentralized testbeds (e.g., no more than 594 nodes in PlanetLab) and the small size of load notifications, this client/server evaluation cannot cause a computational catastrophe.

V. Discussion

The previous section has described the design of the BALLS simulator. We now review whether the functionality of the simulator conforms to what we expect. We validate the simulator in the two simulation modes.

A. Validating the centralized simulations

The centralized simulations are required to observe the peer degree metrics (average and distribution), the arrival cost, the departure cost, the routing cost at different system sizes, as well as the global index management and storage overloads, loads, and capacities along the simulation life.

The measurement of the peer degree is based on the peers' neighbour list size since it reflects the number of connections among the peers. We only count the degree of the peers that actually join the system. The measurement takes place after a successfull peer arrival or departure - the moment of a change in number of peers. It thus ensures the valid observation of the peer degree relative to the varying system size.

The evaluations of peer arrival and departure determine the number of messages sent (to maintain the P2P structure) when these events occur. Because the arrival and departure involve transferring some key interval from a peer to another, the cost is the number of notification messages sent by the giving peer (say q) and the receiving peer (say p). If the transfer succeeds, p replies an acceptance message to q. It also sends the number of its messages to q via this reply. q registers the total number of messages as the cost. In the case of departure, q also counts the number of confirmation messages from its neighbours before leaving. Each completed arrival or departure makes a change in system size. It ensures the valid arrival/departure cost corresponding to all system sizes in a simulation.

Each routing message maintains a list of the peers it passed. When routing finishes, the length of this list reflects the routing cost. The routing cost evaluation registers this value at the end of every routing. If the routing frequency is higher than the frequency of peer arrivals and departures, we can observe the routing costs relative to all the system sizes.

The index management load balancing evaluation and the storage load balancing evaluation measure and register the corresponding overload, load, and capacity of the whole system at every simulation cycle. This ensures observing the effect of the balancing methods at all simulation moments.

Centralized simulations usually require a huge amount of computer resources to accommodate the peers and to execute their operations. We assessed the scalability of the simulator using two measures: simulation size and simulation time. The simulation size includes the population of peers and the population of objects managed by the peers. The simulation time denotes the time (in milliseconds) needed to run a certain number of simulation cycles. We measured these metrics on a computer with a Pentium 4 CPU, 3.4 GHz, 2GB RAM, running OS Red Hat Fedora Core 3 and a computer with the same hardware configuration but running Windows XP.

For measuring the simulation size, we executed experiments with different numbers of peers and different numbers of objects. Figure 2 shows the numbers of objects a simulation can manage corresponding to system sizes: 2^9, 2^{10}, 2^{11}, 2^{12}, 2^{13}, and 2^{14} peers. The number of peers satisfies the requirement of large simulations (thousands of peers). It is comparable to the system sizes of numerous experiments on P2P, such as 4096 in [22], 2250 in [23], or 2048 in [11]. On the other hand, the object population can attain hundreds of thousands. It is comparable to the numbers of objects used in numerous P2P experiments, e.g., 40000 in [24], 3000 in [25], or 30 per peer in [26]. Denoting the peer population as x and the object population as y, we find that they have a linear replationship: $y = -29.22x + 672222$. This reason is due to the limited memory size, which is split between the peers and the objects. In all experiments, Windows XP yields a little higher object population.

To evaluate the simulation time, we executed experiments that measured the time of 30 simulation cycles running in system sizes: 2^9, 2^{10}, 2^{11}, 2^{12}, and 2^{13} peers. In all experiments, the load balancing and the evaluation activities were activated. From the results in Figure 3, we can see a linear relationship between the time and the peer population. Based on the data collected, we calculated the angular coefficient of this relationship as 333.33 in Windows XP and 219.66 in Fedora Core. The execution time of Windows XP is longer than that of Fedora Core. However, even with 2^{13} peers, 30 cycles take only 2674641 ms. This allows a 300-cycle simulation to be completed in no more than 8 hours.

B. Validating the decentralized simulations

In the decentralized simulation mode, evaluation of the load balancing methods is also required to observe the corresponding global overload, load, and capacity during all the simulation life. However, global measurements cannot be locally calculated. As described in Section IV-C, an EvaluationServer aggregates load notifications from the peers and periodically registers the measured values to files. The EvaluationServer maintains a table keeping the load and the capacity of the currently live peers. Each reception of a load notification

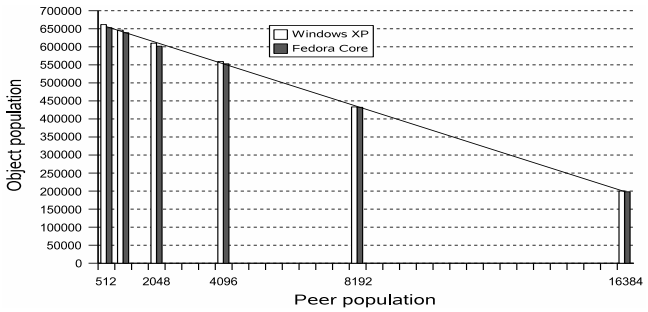

Fig. 2. Simulation size evaluation

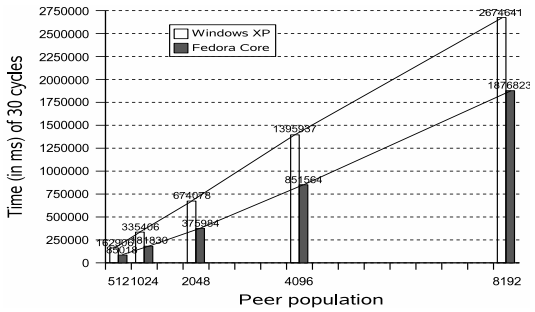

Fig. 3. Simulation time evaluation

brings about an update of the corresponding entry in the table. Since the peers periodically post load notifications, if the EvaluationServer does not receive a peer's notification for some time, the corresponding entry is deleted. Peer arrivals and departures also entail updates in the table. The EvaluationServer produces the global measurement (of overload, load, and capacity) relying on the current state of this table.

This measurement method allows to observe the load balancing metrics of the whole system along the simulation life. The more frequently the peers send load notifications, the more exact results we observe. Since the decentralized mode takes into account the real time factor but not the large-scale factor, our intended experiments will use a modest number of peers (no more than one hundred). This small simulation size will not cause any overload problem to the network system.

VI. Conclusion

Being developed in three independent layers, the BALLS simulator allows us to perform simulations in two environments: centralized and decentralized. The common use of the peer layer for both simulation modes reduces our effort in upgrading the simulator. Moreover, the simulator having this architecture is easy to be extended for other P2P systems. The network and evaluation layers are generic for most P2P structures. An extension therefore is made on the peer layer. We can also add more evaluations by developing new evaluation classes in the evaluation layer.

The evaluation classes provide the ability to explore most performance aspects of the P2P model: peer degree, routing cost, arrival and departure costs, and load balancing. The supported distribution patterns: Zipf and log-normal (for the peers' capacities, the routing source, the key popularity, and the object size) were confirmed by numerous research. Applying them enables us to imitate the most realistic distributions.

We have shown that the measurement methods supported by the simulator allow it to generate valid evaluation data. On the other hand, experiments have verified the simulator's capability in two aspects: simulation size and simulation time.

References

[1] M. F. Kaashoek and D. R. Karger, "Koorde: A simple degree-optimal distributed hash table," in *IPTPS'03*, Feb. 2003.

[2] M. Naor and U. Weider, "Novel architectures for p2p application: the continuous-discrete approach," in *ACM SPAA'03*, June 2003.

[3] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, "A scalabale content-addressable network," in *ACM SIGCOMM'01*, Aug. 2001, pp. 161–172.

[4] A. Rowstron and P. Druschel, "Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems," in *IFIP/ACM Middleware'01*, Nov. 2001.

[5] I. Stoica, R. Moris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for internet applications," in *ACM SIGCOMM'01*, Aug. 2001, pp. 149–160.

[6] B. Y. Zhao, J. Kubiatowicz, and A. D. Joseph, "Tapestry: An infrastructure for fault-tolerance wide-area location and routing," University of California Berkeley, Tech. Rep. UCB/CSD-01-1141, Apr. 2002.

[7] V. D. Le, G. Babin, and P. Kropf, "A structured peer-to-peer system with integrated index and storage load balancing," in *Innovative Internet Community Systems (I2CS) 2005*, Paris, France, June 2005.

[8] "Peersim peer-to-peer simulator. http://peersim.sourceforge.net/," 2005.

[9] S. Joseph, "An extendible open source p2p simulator," *P2PJournal*, Nov. 2003.

[10] ——, "Neurogrid: Semantically routing queries in peer-to-peer networks," in *International Workshop on Peer-to-Peer Computing*, 2002.

[11] S. Merugu, S. Srinivasan, and E. Zegura, "p-sim: A simulator for peer-to-peer networks," in *IEEE MASCOTS'03*, 2003, pp. 213–218.

[12] Q. He, M. Ammar, G. Riley, H. Raj, and R. Fujimoto, "Mapping peer behavior to packet-level details: A framework for packet-level simulation of peer-to-peer systems," in *IEEE MASCOTS'03*, 2003, pp. 71–78.

[13] "Freepastry. http://freepastry.rice.edu/freepastry/," 2005.

[14] F. Dabek, B. Zhao, P. Druschel, J. Kubiatowicz, and I. Stoica, "Towards a common api for structured peer-to-peer overlays," in *IPTPS'03*, Feb. 2003.

[15] J.-C. Bermond, R. Dawes, and F. Ergincan, "De bruijn and kautz bus networks," *Networks*, vol. 30, pp. 205–218, 1997.

[16] F. Chung, J. E.G. Coffman, M. Reiman, and B. Simon, "The forwarding index of communication networks," *IEEE Transactions on Information Theory*, vol. IT-33, no. 2, pp. 224–232, Mar. 1987.

[17] F. auf der Heide and B. Vöcking, "Shortest-path routing in arbitrary networks," *Journal of Algorithms*, vol. 31, no. 1, Apr. 1999.

[18] A. B. Downey, "The structural cause of file size distributions," in *ACM SIGMETRICS'01*, 2001.

[19] Z. Ge, D. Figueiredo, S. Jaiswal, J. Kurose, and D. Towsley, "Modeling peer-peer file sharing systems," in *IEEE INFOCOM'03*, 2003.

[20] Q. Lv, S. Ratnasamy, and S. Shenker, "Can heterogeneity make gnutella scalable?" in *IPTPS'02*, Mar. 2002.

[21] S. Saroiu, K. Gummadi, R. Dunn, S. Gribble, and H. Levy, "An analysis of internet content delivery systems," in *OSDI'02*, 2002.

[22] A. Rao, K. Lakshminarayanan, S. Surana, R. Karp, and I. Stoica, "Load balancing in structured p2p systems," in *IPTPS'03*, 2003.

[23] A. Rowstron and P. Druschel, "Storage management and caching in past, a large-scale, persistent peer-to-peer storage utility," in *ACM SOSP'01*, Oct. 2001.

[24] K. Gummadi, R. Dunn, S. Saroiu, S. Gribble, H. Levy, and J. Zahorjan, "Measurement, modeling, and analysis of a peer-to-peer file-sharing workload," in *ACM SOSP'03*, Oct. 2003.

[25] E. Tanin, A. Harwood, and H. Samet, "A distributed quadtree index for peer-to-peer settings," in *IEEE ICDE'05*, 2005.

[26] W. Yee, D. Jia, and O. Frieder, "Finding rare data objects in p2p file-sharing systems," in *IEEE P2P2005*, Sept. 2005.