Herwig Unger, Saiho Yuen, Peter Kropf, and Gilbert Babin. "Simulation of Communication of Nodes in a Wide Area Distributed System."
in Eurosim 2001 Congress. Delft, The Netherlands. June 2001. Published on CD-ROM.

SIMULATION OF COMMUNITIES OF NODES IN A WIDE AREA DISTRIBUTED SYSTEM

Herwig Unger[1], Saiho Yuen[2], Peter Kropf[2], Gilbert Babin[3]

[1]University of Rostock, Rostock, Germany
hunger@informatik.uni-rostock.de
[2]University of Montreal, Montréal, Canada
{yuensaih,kropf}@iro.umontreal.ca
[3]HEC — Montréal, Montréal, Canada
Gilbert.Babin@hec.ca

ABSTRACT

The complex structure of the Web requires decentralised, adaptive mechanisms efficiently providing access to local and global capacities. To facilitate the development of such mechanisms, it seems to be reasonable to build clusters or grids of machines with similar structures and interests. In such a manner, communities of machines can be built. In a community, every machine contributes to the overall success through a division of management work and a respective collaboration. This article presents and analyses experimental results for algorithms optimising service response times in a community. We introduce a community simulation tool, which is used to experiment with optimisation algorithms. The experimentation results are presented and analysed.

KEYWORDS

Performance evaluation, Simulation and optimisation, Internet computing, Ubiquitous computing.

1 INTRODUCTION

To access the huge amount of hard- and software resources in the Internet and to maintain this under-utilized potential, new approaches are needed. The complex, heterogeneous and dynamic structure of the world wide network requires decentralised, adaptive mechanisms, that provide access to both local and global capacities in an efficient and transparent way. Since a correct load prediction for networks and servers are difficult (Adler, Feldman, and Taqqu 1998), self-organization and adaptation appear to be a promising way to make communication and computation in the Internet and large Intranets more effective (Milojicic 1999). The necessary data updates and the required computation and communication efforts are the reason why a lot of systems are based on decentralized management approaches (Foner 1997; Kramer, Minar, and Maes 1999; Menkov, Neu, and Shi 2000; Kropf, Unger, and Babin 2000).

In addition, it seems to be reasonable to build clusters or grids of machines with similar structures and interests. In such a manner, communities (Plaice, Svoboda, and Alammar 2000) of client and server machines can be built. These machines do not only share a common communication context, but also sets of similar parameters and interests. Mostly, every machine contributes to the success of such a community through a division of management work and a respective collaboration. The Web Operating System (WOS) (Kropf, Unger, and Babin 2000; Unger 2000) was built to support such communities. The WOS is an open middleware solution allowing for software services to be distributed over the Internet. Typically, the WOS infrastructure provides the tools to search for and prepare all the necessary resources that fulfil the desired characteristics for a service request (e.g., performance, storage, etc.).

In WOS, such a set of resources or nodes is called a *community* or WOSnet. Each node of a WOSnet maintains a warehouse to keep information about the community (i.e., characteristics of other nodes of the community). A WOSnet is dynamically formed, and nodes may join and leave a community. An initial bootstrap procedure (Babin, Kropf, and Unger 1998) is used by a WOS node to locate other WOS nodes in its vicinity. The WOSnet evolves from that basic information stored in the nodes' warehouses through the location and execution activities performed by the different WOS nodes. The warehouses are continously updated through these activities. For example, a service location request will leave its result in the warehouses of the visited nodes.

This article presents and analyses experimental results for algorithms optimising service response times in a community. In Section 2, a network community simulation tool is presented which is used to experiment with optimisation algorithms discussed in Section 3. The experimentation results are presented in Section 4. Finally, Section 5 concludes the paper with a discussion of the approach.

2 A NETWORK COMMUNITY SIMULATION

We have developed a tool which simulates a network by recreating the network's environment. A randomly generated graph placed in a 2D grid represents a WOSnet. Nodes act either as servers or clients, the actual proportion in the network being defined by a simulation parameter. Each node has a unique identifier. The initial network graph is constructed as follows: a certain number of node identifiers is assigned to each node and stored in its warehouse. These warehouse entries represent a directed valuated arc to the node represented by the identifier. In other words, a node is said to know the nodes identified by its warehouse entries. The valuation of the arcs depends on the distance between the nodes in the 2D grid of the network simulation. All nodes, clients and servers, are only connected to server nodes. The resulting directed graph represents a *community*. It corresponds to a virtual network structure superimposed onto the real network structure, where the virtual structure reflects the knowledge of each node about the others. We further assume that each node may potentially know each other node, as it is the case for IP networks. There are therefore no restrictions imposed as far as the construction of the virtual graph, or community, is concerned. However for constructing the initial community, the maximal number of warehouse entries (i.e. known nodes) is restricted by a simulation parameter. As the simulation progresses, the client nodes generate service requests to the server nodes they know. We define a request as the action of asking for data, like a file transfer. Therefore, when servers deliver a service, they deliver the requested data to the clients. There is only one kind of service, but the response time to a request may vary.

The request generation follows a sawtooth function, using the simulation time as its parameter. This way, we obtain request cycles, where the number of requests increase within a cycle. The simulation is run for a number of such request cycles. The simulation time is measured in units of times (ut) created by the simulation tool. Thus, each action of the nodes is measured using that simulated time scale.

During the simulation it may happen that clients are not satisfied with the response times of the currently used server nodes, or that a server node is overloaded. In either case, a community optimisation is performed which reorganises the initial virtual network with the goal to minimise the response times. A specific client seeking for a more suitable response time to his requests performs an optimisation process by recursively searching for server nodes it does not know yet by inspecting the warehouse entries. During this process the visited warehouses are updated, thus dynamically restructuring the virtual network or community. In this context, we define the *optimisation of the network* as the optimisation of the response time for every client node of that network. This means, that new arcs with better valuations are created while others will disappear.

The simulation tool records and monitors every action of the nodes and the algorithms. With the data collected, we are able to evaluate optimisation algorithms and the global behaviour of the nodes in the network. The following sections present two optimisation algorithms and experimental results.

3 OPTIMISATION OF NETWORK COMMUNITIES

The response time for a service request of client c to server s may be defined as :

$$t(c, s) = k(s) * t_{\text{cpu}} + d(c)/b(c, s)$$

where k is the CPU speed normalisation factor, t_{cpu} is the execution time, $d(c)$ is the amount of data to be transferred and $b(c, s)$ is the communication bandwidth of the channel used.

The computation time ($k(s) * t_{\text{cpu}}$) normally corresponds to the execution time of the client's request. However, to simplify our model, we only consider the response time of the communication layer (e.g., TCP/IP). Therefore, t_{cpu} is the average response time of the communication layer to reach the servers and $k(s)$ normalises the speed differences between machines. As a consequence, a strong bandwidth between two nodes yields a short response time for a request. To further simplify our model, we use the actual distance between two nodes in the 2D grid representing the network as the value for the bandwidth. Thus, when we optimise the response time for a node, we look for a server which is closer to the client than the server currently used by that node.

3.1 Optimisation algorithms

We experienced with two different optimisation algorithms: the *Whip* algorithm and the *Wanderer* algorithm (Unger 2000). Each algorithm uses a different strategy to locate potential server nodes. However, they follow the same pattern that may be divided into three distinct stages: initialisation, search, and update stages.

At the the *initialisation stage*, the algorithm is initialised with the information available in the node requesting the optimisation. We will refer to that node as starting node. The Whip algorithm uses a centralised approach, and therefore all the processing will be done by the starting node. The Wanderer algorithms uses mobile agents (Russell and Norvig 1995; Kramer, Minar, and Maes 1999). At this first stage, it creates an agent, the wanderer, that will "wander" in the network and collect information on other nodes.

At the *search stage*, both algorithms explore the network to gather useful information about potential servers. The exploration is done by requesting information from a certain number of nodes and by using their warehouse information to obtain more information about the whole network. Depending on the algorithm used, the strategy to determine the next node to query will be different.

For the *Whip* algorithm, the strategy is based on a random selection method. The search takes place along a chain of nodes beginning at the starting node. At each node in the chain, the next node to interrogate is selected randomly from information currently available to the algorithm (i.e. in the respective warehouses). One of the parameters of the Whip algorithm is the maximal number of nodes queried. Each time a node queries a server node, it records information about that server node. The search ends when the required number of queries have been performed.

For the *Wanderer* algorithm, the wanderer agent will do a complete search of the network using a depth first strategy. At each node, the wanderer adds server node information from the local warehouse to its list of nodes to visit. The agent therefore realizes a deterministic search in the community. It stops when there are no more nodes to visit.

At the *update stage*, both algorithms select a server node with a shorter response time than the response time of the server currently used by the starting node.

3.2 Performance Evaluation

We evaluate the performance of an algorithm with two parameters: effectiveness (E) and convergence time (t_c). Effectiveness (E) is defined as the ratio of the minimum response time available on the network (r_{avail}) over the minimum response time obtained by the algorithm (r_{min}):

$$E = r_{\text{avail}}/r_{\text{min}} * 100$$

Table 1. E and t_c for the Whip Algorithm vs # of nodes and diameter d of the graph (at most 5 warehouse entries; 2 % of server nodes)

# Nodes	d	# Queries = 4		# Queries = $d + 1$	
		E (%)	t_c (ut)	E (%)	t_c (ut)
500	4	98.58	240	98.49	260
1000	4	96.34	300	97.69	340
2000	5	89.03	380	92.16	440
3000	5	85.07	420	89.48	440
4000	5	83.45	420	88.04	460
5000	6	83.40	420	89.06	460

Table 2. E and t_c for the Whip Algorithm vs # of warehouse entries and # of queries (5000 nodes; 2 % of server nodes)

	# Entries = 3		# Entries = 5		# Entries = 8		# Entries = 10	
# Queries	E (%)	t_c (ut)	E (%)	t_c (ut)	E (%)	t_c (ut)	E (%)	t_c (ut)
2	63.91	360	75.11	360	79.03	360		
4	77.02	380	83.40	420	85.49	420		
6	82.51	380	88.11	380	89.12	400	90.01	400
8			89.56	400	90.16	400	92.14	420
Diameter	9		6		4		4	

The larger the value of E, the more effective is the algorithm. We define r_{avail} as the average response time of requests, if all the clients of the network launch exactly one request simultaneously and the clients are using the server with the strongest bandwidth in the network. We define r_{min} as the average response time of requests, if all the clients of the network launch exactly one request simultaneously and the clients are using the server with the strongest bandwidth that the algorithm was able to find.

Convergence time (t_c) is defined as the time required for reaching the minimum response time obtained by the algorithm (r_{min}). It is measured in the units of time (ut). The smaller t_c, the faster the algorithm.

4 EXPERIMENTATION

We tested both algorithms in randomly generated networks. For each experimental parameter, we ran 5 to 20 simulations. Each simulation was made on the same network and under the same condition.

For the Whip algorithm, we had three experimental parameters: network size, number of queries, and number of entries in a warehouse. For every simulation, the network generated had 2 % of nodes acting as servers and 98 % of nodes acting as clients. These percentages are arbitrary, but reflect typical situations. Each simulation was run for the duration of one cycle of the sawtooth function for the request generation. Table 1 presents the effectiveness and convergence time for varying network sizes. In each case, the number of queries (i.e. length of search chains) was chosen such that it was larger than the diameter of the graph, allowing each client to potentially reach each server node, though it is of course not assured, because the search procedure is not guaranteed to use the shortest paths for the searches. Table 2 shows the influence of the number of queries and the number of warehouse entries on the algorithm's performances (effectiveness and convergence time). It also shows the influence of the number of queries with regard to different diameters of the underlying graphs.

For the Wanderer algorithm, we used four experimental parameters: network size, percentage of server nodes, number of entries in a warehouse, and percentage of wanderer agents over the number of client nodes. Because the Wanderer algorithm is much slower than the Whip algorithm (due to the complete search made), we limited the network size to 2000 nodes (Table 3). Tables 4, 5 and 6 respectively show the effect of changing the percentage of nodes acting as servers, the number of entries in a warehouse and the percentage of wanderer agents over the number of client nodes.

Table 3. E and t_c for the Wanderer Algorithm vs # of nodes (2 % of server nodes; 5 entries; 100 % wanderer/client)

# Nodes	E (%)	t_c (ut)
500	100.00	260
1000	99.99	340
2000	99.81	440

Table 4. E and t_c for the Wanderer Algorithm vs % of server nodes (2000 nodes; 5 entries; 100 % wanderer/client)

% Servers	E (%)	t_c (ut)
1	99.96	400
2	99.81	440
3	99.30	400

Table 5. E and t_c for the Wanderer Algorithm vs # of entries (2000 nodes; 2 % server nodes; 100 % wanderer/client)

# Entries	E (%)	t_c (ut)
4	99.74	440
5	99.81	440
6	99.91	440
8	99.77	440

Table 6. E and t_c for the Wanderer Algorithm vs % wanderer/client (2000 nodes; 2 % server nodes; 5 entries)

% wanderer/client	E (%)	t_c (ut)
33	99.72	6000
66	99.66	600
100	99.81	440

5 DISCUSSION

5.1 Whip Algorithm

A number of conclusions may be drawn from an analysis of the Whip algorithm. The Whip algorithm works better for a small network; as the results show (Table 1), the bigger the network, the poorer the effectiveness and the larger the convergence time.

By increasing the number of queries or the number of the entries in the warehouse, we increase the effectiveness of the Whip algorithm (Table 2). This increase is significant until the number of queries is equal to the number of warehouse entries. After that point, however, the increase in effectiveness is small. The only way to increase effectiveness is to increase both parameters simultaneously.

We experimented with a small number of queries and a small number of entries in the warehouse. It turns out that the diameter of the networks generated by the simulation tool is always smaller than 10, implying that on average two nodes are separated by at most 5 nodes. Hence, the algorithm is able to reach most of the server nodes in the network in less than 4 queries. In addition, increasing the number of entries in the warehouse decreases the diameter, since the number of entries is directly related to the number of nodes currently known by a certain node. Recall here that knowledge of a node is reflected by a directed arc to that node. There is thus no need to have large values for these parameters. A small value for the number of queries also has the advantage of decreasing the computational resources required and the convergence time of the algorithm.

Because of its simplicity, the Whip algorithm is inexpensive in terms of computing resources, compared to the Wanderer algorithm.

5.2 Wanderer algorithm

From our experiments, we can see that the Wanderer algorithm is very effective, even when varying all the simulation parameters. It turns out, however, that its convergence time changes greatly depending on these same parameters. For instance, we observe that the convergence time increases when the number of nodes increases. This can be explained by the “complete search” nature of the algorithm. We also observe such an increase when the ratio of wanderer agents per client nodes increases. By limiting the number of wanderer agents in the network, we limit the number of client nodes performing an optimisation at a given time. Furthermore, as the ratio is reduced, client nodes spend more time verifying whether or not they can launch an optimisation, which degrades the convergence time even more. The convergence times necessary to achieve the 99% effectivenesses shown in Table 6 are larger than those of the other simulations. Indeed, the number of cycles of simulation was 20 and 3 for the 33% and 66%

wanderer/client ratios, respectively, as opposed to the 1 cycle simulations in all other cases.

Finally, convergence time is not the only drawback of the Wanderer algorithm. The algorithm is very expensive in terms of computational resources; since a wanderer agent is mobile, it uses resources on every node it visits. The quantity of resources required is proportional to the number of wanderer agents active in the network. In a larger network, that number can be extremely high. As a consequence, the Wanderer algorithm also runs into the possibility of congesting the network.

5.3 Final Remarks

In this paper, we demonstrated the possibility of optimising any network structure by using the experimented algorithms. Both algorithms have their advantages and disadvantages. The Whip algorithm is fairly efficient (E is fairly high). It is simple and almost costless. But the optimisation is limited by the size of the network; the larger the network, the poorer the performance will be. The experiments conducted so far indicate that the Wanderer algorithm can adapt more easily to changes in the network than the Whip algorithm. The Wanderer algorithm has an almost constant efficiency, no matter the size of network or the number of entries in the warehouse, but the high demands on computational and communication resources as well as time must be carefully observed. We are therefore further investigating the Wanderer algorithm to fully understand its limitations and to address those limitations with specific changes in the algorithm. Preliminary results obtained with variations of the Wanderer algorithm indicate that ressource, performance and congestion problems can be satisfactory allevited.

REFERENCES

Adler, R. J., R. E. Feldman, and M. S. Taqqu (1998). Heavy-tailed probability distributions in the World Wide Web. In *A Practical Guide to Heavy Tails: Statistical Techniques and Applications*, pp. 3–25. Birkhäuser.

Babin, G., P. Kropf, and H. Unger (1998). A Two-Level Communication Protocol for a Web Operating System (WOS). In *IEEE 24th Euromicro Workshop on Network Computing*, pp. 934–944. Västerås, Sweden.

Foner, L.N. (1997). YENTA - A Multi-Agent Referral System for Matchmaking. In *First International Conference on Autonomous Agents (Agents '97)*. Marina del Rey, CA.

Kramer, K., N. Minar, and P. Maes (1999). Mobile software agents for dynamic routing. *Mobile Computing and Communications Review 3*(2).

Kropf, P., H. Unger, and G. Babin (2000). WOS: an Internet Computing Environment. In *Workshop on Ubiquitous Computing*, PACT 2000 (IEEE International Conference on Parallel Architectures and Compilation Techniques), pp. 14–22. Philadelphia, PA.

Menkov, V., D.J. Neu, and Q. Shi (2000). Ant world: A collaborative web search tool. In P. Kropf et al. (Ed.), *Distributed Communities on the Web (DCW 2000)*, LNCS 1830, pp. 13–22. Quebec, Canada: Springer.

Milojicic, D. (1999). Operating systems - now and in the future. *IEEE Concurrency 7*(1), 12–21.

Plaice, J., P. Svoboda, and A. Alammar (2000). Building Intensional Communities Using Shared Contexts. In P. Kropf et al. (Ed.), *Distributed Communities on the Web (DCW 2000)*, LNCS 1830, pp. 55–64. Quebec, Canada: Springer.

Russell, S. and P. Norvig (1995). *Artificial Intelligence: A Modern Approach*. Prentice Hall.

Unger, H. (2000). Distributed Resource Location Management in the Web Operating System. In SCS A. Tentner (Ed.), *High Performance Computing 2000 (ASTC)*, pp. 213–218. Washington, DC.