A P2P Approach for Global Computing

Wen Dou[1] Yan Jia [2] Huai Ming Wang [3] Wen Qiang Song[4] Peng Zou [5]

[1,2,3,5] Dept. of Computer Science, Changsha Institute of Technology, 410073, China
[4] Computing Center of 3rd Military Medical University, 400038, China

Abstract

We describe a peer-to-peer self-organizing overlay network for our global computing system. The preliminary simulation results show that the network has some small-world characteristics such as higher clustering coefficient and short path length, which leads to an efficient heuristic task scheduling algorithm on which any volunteer peer with limited knowledge about the global network can dispatch its excrescent computation tasks to powerful nodes globally, in a way contrary to the current global computing system in which a global broker is responsible for the task scheduling. We argue that our approach is a starting point to eliminate the broker component which makes current global systems unscalable.

1 Introduction

This paper describes a peer-to-peer self-organizing overlay network on which we build a java based general-purpose global computing system, Paradropper. The goal of Paradropper project is to implement an unified computing network on which anyone can submit (contribute) his/her computing jobs (resources) in an unified overlay network via unified user-friendly GUI (see Figure 1). The most desirable feature of our system is its peer-to-peer architecture. Current global computing systems such as Javelin++[1] and Bayanihan[2] are essentially center-based. The centralized architecture leads to some problems in scalability and accessibility. Firstly, in systems such as Javelin++ and Bayanihan, there is a broker component which is responsible for registering, task dispatching, job submitting, load balancing, and synchronizing. When there are many computing jobs managed by the broker, it becomes a bottleneck of the whole computing network and a potential single point of failure. Though the two pro- jects both prompt a broker network [1] (in Bayanihan, a similar concept is called server pool) approach to avoid this situation, we argue that the maintaining cost of the broker network cannot be neglected. Secondly, in most other general-purpose global computing systems (including above two systems), there must be one or more well-known sites to host the brokers, non-professional clients and volunteers need firstly to know these brokers address for submitting their computing jobs or registering their computers. This in a sense will frustrate those volunteers who do not want to know any troublesome details of how to contribute their computation resources. So how to eliminate the broker component (or weaken its functions?) is a key issue of the system scalability and accessibility. The peer-to-peer overlay network has been widely used in file-sharing and data-sharing applications [3, 4, 5], many researches [6, 7, 8, 9] has proved that a well-designed peer-to-peer topology would make the applications built on it more scalable and assessable. In this paper, we investigate the possibility of using a pure peer-to-peer network to construct computing network. Our preliminary work show that, by constructing a small-world network with clustering characteristic, any volunteer peer with limited knowledge about global network can dispatch its computation tasks to the more powerful nodes globally, without the help of a global broker component. In another word, our work lighten the broker's overload by unleashing its scheduling and registering responsibility, and any peer can join the computing network just by starting the (Paradropper) program without any knowledge about broker (like the situation in some pure decentralized file-sharing application). Our ongoing work is dealing with eliminating other responsibility of the broker component, such as heartbeat detecting, termination detecting, load balancing, and synchronizing, etc. The final goal of our project is to eliminate the whole broker component completely.

The paper is organized as follows: in section 2 we discuss construction of Paradropper network. In section 3 we prompt a heuristic task scheduling algorithm. In section 4 a conclusion is given.

0-7695-1926-1/03/$17.00 (C) 2003 IEEE
Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS'03)

2 The Constructing of Paradropper Network

Many researches [10, 11, 12, 13, 14] have shown that peer-to-peer applications such as the Kazaa decentralized file-sharing network, have some self-organizing network characteristics, which contribute to peer's easily and effectively searching for data. These characteristics

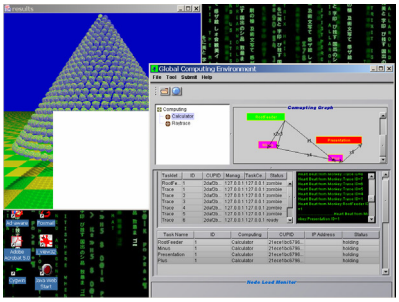


Figure 1. **Paradropper screenshot**

include (1) high clustering coefficient, (2) short path length and (3) power-law distribution of node degree, etc. All these characteristics help peers relying on limited local knowledge to make routing decisions rather than depending on global knowledge. After Milgram[15] prompted the famous 6- degrees of separation notion by a social experiment, Watts et al. [16] started by looking at graphs and the metrics for graphs common in social networks. Two key ideas they focused on were clustering and path length. The first concept is a formal way to state how "cliquish" a graph is, i.e., how tightly the links of a node in the graph are connected to each other. For convenience, we give the formal graph theory definitions about these concepts:

Definition 1 *The connectivity of a vertex v, k_v, is the number of attached edges.*

Definition 2 *Let d(i, j) be the length of the shortest path between the vertices i and j, then the characteristic path length, L, is d(i, j) averaged over all C_n^2 pairs of vertices.*

Definition 3 *The neighborhood of a vertex v, $\Gamma_v = \{i : d(i,v) = 1\}$ so $(v \notin \Gamma_v)$*

Definition 4 *The local clustering coefficient, C_v, is: Cv* $= \dfrac{|E(\Gamma_v)|}{C_{k_v}^2}$ *,where $|E(\bullet)|$ gives a sub graph's total number of edges.*

Definition 5 *The clustering coefficient, C, is C_v averaged over all vertices.*

Watts et al. have discovered that almost all the self-organizing networks have two basic characteristics:(1) higher clustering coefficient than the random network (2) short path length, and the (2) lead to the famous small-world phenomenon in social network and many self-organizing networks. Unfortunately, Watts's network constructing approaches are based on regular network (lattice), and not suitable for dynamic network constructing and maintaining. There are other researches [17] also prompt some approaches which are more suitable for dynamic network constructing, but these approaches seem to be dedicated to constructing only low diameter network without more concerning about clustering characteristic which is important in our system. In this paper, we prompt a very simple construction approach, and the simulation show that our approach is very suitable for constructing small-world network with clustering characteristic.

2.1 The Construction of Paradropper Computing Network

We use the following approach to construct Paradropper network:

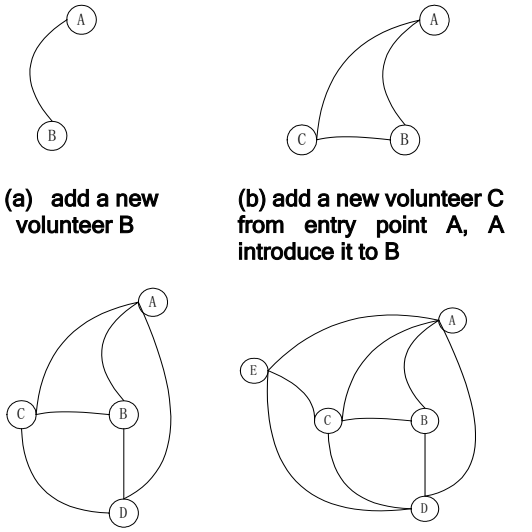
At bootstrapping stage, every new volunteer computer interacts with an entry point cache, which randomly selects an in-network volunteer (we call it network entry point) as response to the new participator.

(1) The new volunteer then send a message to the entry point, if the entry point's neighbor number is beyond the upper bound *Z*, it will accept the new node as its new neighbor, otherwise, refuse it. In the later situation, the new node will ask the cache again for a new entry point.
(2) If the entry point accepts the new node, it will send notify messages to part of his old neighbors. Let the number is *K*.
(3) These *K* neighbors who received the notify message will also try to build a neighbor relationship with the new node.

In social terms, when you have a new friend, you would likely introduce him (her) to part of your old friends. We argue that it is the main reason why in social network the probability of our friends know each other is very high. The clustering characteristic emerges when we introduce new friends to our old friends. The re-wiring behavior described by Watts [16] can also be explained by our model: some nodes have friends in several clusters. These nodes play the re-wiring roles just like those nodes selected randomly in Watts's lattice network.

Practically, a volunteer computer can only have limited neighbors. Suppose when an entry point accepts a new node as a new neighbor, it will introduce the new node to maximum *K* number of his old neighbors. In Paradropper, we let the upper bound of a volunteer can have neighbors is *2K*, i.e., *Z=2K*. Sometimes, not all the *K*

neighbors notified will accept the new node as a new neighbor because of the upper bound *Z*. Let us suppose every in-network volunteer computer knows how many neighbors his neighbors have. The notify message will first send to neighbors whose neighbor number is below the upper bound *Z*. The reason why we take *Z* as *2K* is empirically: we find when *1*<*K*<*Z*-1(If K = Z-1, the network will become a complete graph and stop to grow as the nodes number in network reach Z), the constructed network appears almost the same properties with slightly decreasing in clustering coefficient when $K \rightarrow Z\text{-}1$ and slightly increasing in path length when $K \rightarrow 1$. More details about *K* and Z are discussed in our technical report [18].The Figure 2 illustrate the construction of 5 nodes network.



(a) add a new volunteer B

(b) add a new volunteer C from entry point A, A introduce it to B

(c) add a new volunteer D from entry point B, B introduce it to A, C

(d) add a new volunteer E from entry point C, C introduce it to A, D

Figure 2. **A Simple illustration of Paradropper Network Construction**

2.2 Paradropper Network Simulation

We use simulation approach to investigate the properties of Paradropper network, i.e. clustering coefficient and path length. A simulator (Figure 3) was build with java. We use undirected graph to represent Paradropper network and vertex to represent volunteer computer. A failure simulation thread make some links disable (every 500ms) randomly, and another recovery thread periodically check if there is node with no neighbor (every 1000ms), if so, give it an entry point randomly and make it re-connect to the network again. We have tried scale of 100,400, 600, 1000 nodes, with K = 2, 4, 8, 10, 20. The results are shown in Figure 4, 5. The simulation result show that our construction approach leads to a self-organizing overlay network, with the clustering coefficient

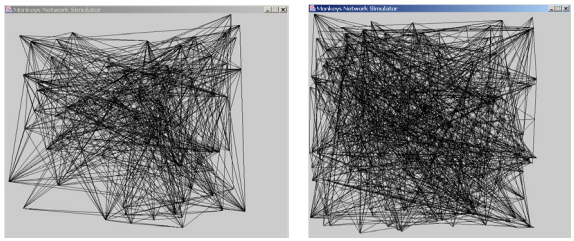


Figure 3. **(a) a Paradropper simulation network with K=10, Z =20, 100 nodes; (b) a Paradropper simulation network with K=10, Z =20, 400 nodes**

is very high and the path length decrease sharply with limited K. For example, when scale is 100, K=4, the clustering coefficient $C \approx 0.65$, and when the scale became 1000, C just has some slight change($C \approx 0.67$).

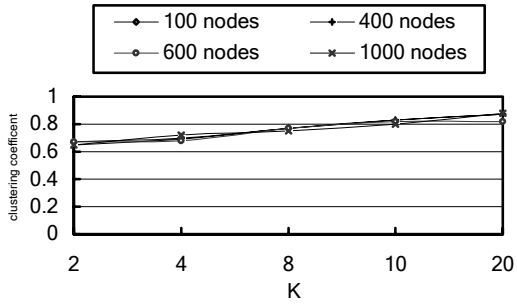


Figure 4. **Simulation result of clustering coefficient**

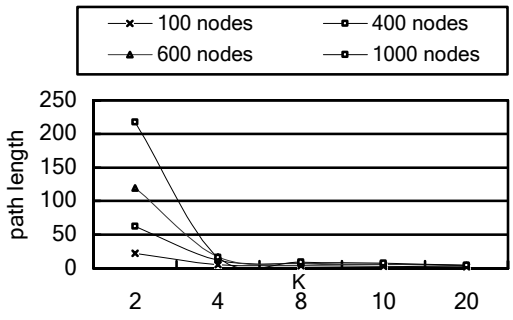


Figure 5. **Simulation result of path length**

The path length decrease sharply with the increasing of upper bound Z, while the clustering coefficient seems get

little affected. In fact, the C gets slightly increased when the nodes have a higher upper bound Z. This phenomenon can be explained easily: when Z becomes larger, the whole network gets more tightly. In an extreme situation, when Z=N-1(N is the scale of the network) and K=Z, the whole network will become a complete graph gradually with C=1 and the path length L=1. The failure simulation thread and

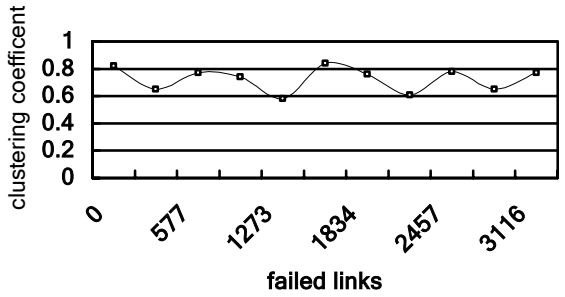


Figure 6. **Simulation result of clustering coefficient in random links failure, 1000 nodes**

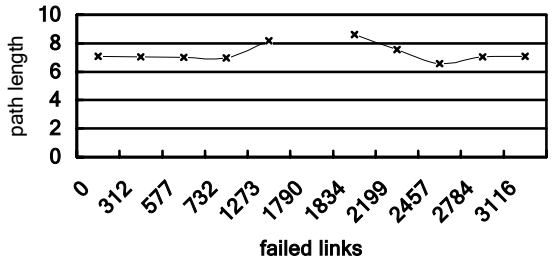


Figure 7. **Simulation result of path length in random links failure, 1000 nodes**

recovery thread are used to investigate the robustness of Paradropper network in an ad-hoc environment, which is the nature of internet. The simulation results are shown in Figure 6, 7. With the interaction between the failure thread and recovery thread, the clustering coefficient waves at a slight scope, and the network partitioned occasionally but could be repaired at last. The explanation is intuitive: when a node lost some of its neighbors, the local clustering efficient C_v will decrease and the global clustering coefficient C trends to decrease with a lot of nodes lost their neighbors. But as we discuss in section 2.1, when a node finds itself lost all its neighbors, it will asks the entry point cache to give it a new entry point to re-join to the network, so its C_v will increase and the global C will recovery with the work of the recovery thread. The path length is in the same situation, when a node lost some of its neighbors, it means that node lost some possible shortest path to some other nodes. In an extreme situation, this will make the whole graph "partition". But the recovery algorithm will merge these partitions as long as there is enough running time.

3 Scheduling Algorithm

Here scheduling means routing tasks to volunteer peer efficiently, i.e., with a high probability, tasks are more likely to be routed to the powerful peers in Paradropper network. In a random network, deciding to whom to send the task is sightless. Because Paradropper network is close to a self-organizing network with high clustering coefficient and short path length, we could implement a very simple but efficient heuristic scheduling algorithm.

As we discuss above, every Paradropper peer has limited neighbors. In each routing hop, the idlest neighbor is selected as the target. In every peer, there is a variable *WorkLoad* recording the current workload of the peer. Whenever the peer accepts a task, its Workload increase by 1, and whenever the peer finishes a task, its Workload decreases by 1. Here we use the number of tasks the peer accepted to represent the workload of a peer. It's a coarse granularity workload representation, but we thought it is enough to explain our scheduling algorithm here.

A new volunteer has the workload 0. When the workload of a peer gets changed (accept a task or finish a task), the peer will notify all its neighbor using *Load Change Report Message (LCRM)*. We give the scheduling algorithm as follow:

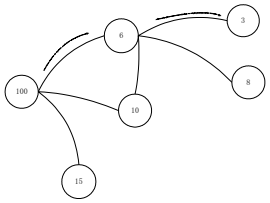
Algorithm (pseudocode)

(1) receive a task (or the sponsor who first schedule the task);
(2) ***if****(ttl>0){*
Select a neighbor whose workload is the smallest in its neighbors;
(3) ***if*** *the selected neighbor 's workload low than the peer's*
(4) ttl decrease by 1;
(5) Send the task to the neighbor;
(6) ***else***
(7) Accept itself;
(8) ***endif***
(9) ***else***
(10) Accept it itself.
(11) ***endif***

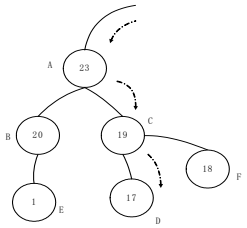
The *ttl* is used to limit the hops of a task message. Because Paradropper network has a short path length, in most case, a peer could reach any peer in limited hops (in our system, we set the *ttl* to 11). The procedure is shown in Figure 8-a.

Can a random network do the same thing? No. For example, a random network can not deal with the following case shown in Figure 8-b. In figure 8-b, when node A accepts a task, according to our scheduling algorithm, the task will be send to node D, while the node E has the lowest workload. In Paradropper network, because the clustering coefficient is very high, which

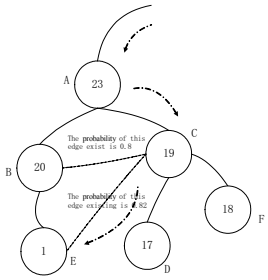
means the probability of a peer's neighbors knowing each other is very high. Suppose the clustering coefficient is ***P,*** then the probability of B and C have a neighbor relationship is ***P,*** and the probability of which E and C have a relationship is P^2. Consider our simulation results, the average clustering coefficient is 0.8, then the probability that there is an edge between E and C is 0.5+0.5x0.64=0.82. So the task is likely to be sent to C with a high probability (see Figure 8-c).



(a) task scheduling in Paradropper network



(b) in a random network, scheduling is sightless



(c) in a network with high clustering coefficient, tasks will be routed to those powerful nodes with high probability

Figure 8. Task scheduling in Paradropper network

A simulation is done to show a large scale scheduling result. In our simulation, we setup a 1000 nodes network (K=10), every node has a *superfactor*. When a node accepts a task, it increases its workload by its *superfactor*. More powerful node has the smaller *super- factor*. The distribution of the nodes is shown in Figure 9(which is close to a Zipf distribution). A thread every t1 (1-200) ms randomly submit a random number of tasks, and hold t2 (1-2000) ms to mimick the executing procedure. After t2 ms, the workloads of these nodes decrease by their superfactor. After enough time, we randomly select 4 sampling point to check the accepted tasks of different kinds of nodes, the result is shown in Figure 10. As the result indicated, those powerful nodes (with lower superfactor) have got more tasks than those weaker nodes. It implies that our scheduling algorithm can not only be used to dispatch tasks, but also can be used to balance loading of the whole computing network. When a node get overload, it can re-send its excrescent computation tasks to the network. With high probability, the re-sent tasks will be accepted by those powerful nodes (nodes with low workload). Someone may argue that even if in a random network we could do the same thing. As we discuss above, because of the poor clustering characteristic, scheduling in a random network will miss many nodes which are more suitable for those tasks. For a more accurate discussion, we prompt the concept of *scheduling efficiency* to evaluate and compare the scheduling capability between our network and random network. From figure 11 we can conclude that the scheduling efficiency with our network is far superior to random network.

Suppose that there is a super node in the Paradropper network, its initialized workload and superfactor are always 0, and the initialized workload and superfactor of other nodes are value large then 1. We randomly submit N tasks to the network and count the number of tasks accepted by the super node, let the number is W. We define E=W/N as the *scheduling efficiency*. In scales of 100, 600 and 1000 nodes, we get the scheduling efficiency E which is shown in Figure 11.

Superfactor	distribution	Superfactor	distribution
1	1	9	81
2	4	10	100
3	9	11	121
4	16	12	144
5	25	13	169
6	36	14	181
7	49		
8	64		

Figure 9. Node distribution with different "superfactor"

As we see in Figure 11, the E can reach very high level in Paradropper network with limited TTL, compared to the low level in random network. Figure 10 and 11 shows that, even if without any global knowledge of the computing network, the topology of Paradropper network and our heuristic scheduling algorithm make it possible for any volunteer to route tasks anywhere and anytime to the most powerful nodes in the network with high probability. This means that we need not a global broker to schedule the tasks dispatching or node registering in our computing network expressly.

4. Conclusion

In this paper, we prompt a simple and practical approach to build our peer-to-peer global computing net- work, the simulation results show that, our approach lead to a

network with higher clustering coefficient and short path length, which has been used to implement our heuristic scheduling algorithm. Our experimental results also suggest that the global broker component existing in current global computing system could be eliminated (or be weakened) from our peer-to-peer computing network in order to make system more scalable and accessible. Our current work proves that at least the scheduling and registering responsibility of broker component could be unleashed.

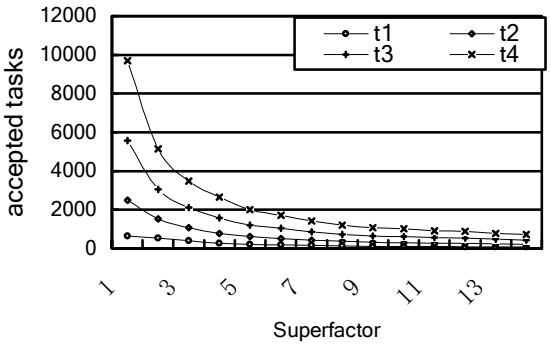


Figure10. **Accepted tasks using Paradropper scheduling algorithm,1000 nodes, K=10**

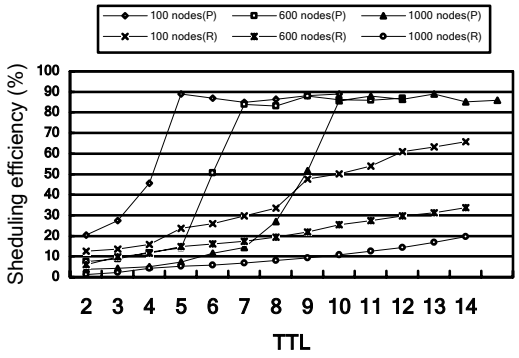


Figure 11. **Scheduling efficiency of Paradropper simulation network and random network. (P: Paradropper, R: random network)**

5. References

[1] M. O. Neary, S. P. Brydon, P. Kmiec, S. Rollins, P. Capello, "Javelin++: Scalability Issues in Global Computing," Proceedings of the ACM Java Grande 1999 Conference, June 12-14, 1999, San Francisco, California.

[2] Luis F. G. Sarmenta, "Volunteer Computing", Ph.D. Thesis, MIT Department of Electrical Engineering and Computer Science, March 2001.

[3] http://www.gnutella.co.uk.

[4] I. Clarke, O. Sandberg, B. Wiley, and T.W. Hong. Freenet: A distributed anonymous information storage and retrieval system in designing privacy enhancing technologies. Inter- national Workshop on Design Issues in Anonymity and Unobservability, LNCS 2009, 2001.

[5] S. Ratnaswamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content-addressable network. ACM SIGCOMM, 2001.

[6] H. Zhang , A. Goel, R. Govindan. "Using the Small World Model to Improve Freenet Performance. Proceedings, IEEE Infocom., 2002

[7] A. Montresor, "Anthill: a Framework for the Design and Analysis of Peer-to-Peer Systems", 4th European Research Seminar on Advances in Distributed Systems (ERSADS '01), Bertinoro, Italy (May 2001).

[8] A. Montresor, Hein Meling, Ozalp Babaoglu, "Towards Self-Organizing, Self-Repairing and Resilient Peer-to-Peer Systems", Proceedings of the 1st International Workshop on Future Directions in Distributed Computing, Bertinoro, Forlì, Italy, June 2002.

[9] http://www.fasttrack.nu

[10] L. Adamic, The Small World Web,Technical Report, Xerox Palo Alto Research Center,2000

[11] J. Kleinberg. Navigation in a small world. Nature, 406, 2000.

[12] K.Aberer, M.Punceva, M.Hauswirth, R.Schmidt, Improving Data Access in P2P Systems. IEEE Internet Computing 6(1): 58-67. January-February, 2002.

[13] M. Faloutsos, P. Faloutsos, and C. Faloutsos. On power-law relationships of the Internet topology. ACM SIGCOMM, 1999.

[14] A. Oram Peer-to-Peer: Harnessing the benefits of a disrupt -tive technology (O'Reilly, 2001).

[15] S. Milgram: The Small World Problem, Psychology Today 1(1), 60-67 (1967)

[16] D. J. Watts and S. H. Strogatz. Collective dynamics of 'small-world' networks, Nature 393, 440--442 (1998). Net- works, Nature 393, 440--442 , 1998.

[17] G. Pandurangan, P. Raghavan, E. Upfal, Building Low-Diameter P2P Networks, Proceedings of the 42nd Annual IEEE Symposium on the Foundations of Computer Science (FOCS), 2001

[18] http://paradropper.sourceforge.net