Paradropper: A General-Purpose Global Computing Environment Built On Peer-to-Peer Overlay Network

Liu Zhong, Dou Wen, Zhang Wei Ming and Zou Peng
phillipliu@vip.sina.com
Department of Management Science, Department of Computer Science,
Changsha Institute of Technology, P.R.China

Abstract We describe Paradropper, a general-purpose global computing system based on a peer-to-peer self-organizing overlay network. In Paradropper, the peer is organized into an overlay network with "small world" characteristics, and workload is well balanced among the peers according to their workload in a decentralized manner. Particularly, it not only supports embarrassing parallel computing mode, but also allows correlativity between tasks. Moreover, the master/worker roles are extended in Paradropper, which makes it more flexible in computing environment configuration.

Keyword:Paradropper; general-purpose global computing system; small world

1 Introduction

This paper describes Paradropper, a general-purpose peer-to-peer global computing system. Paradropper has three distinguishable features compared with other global computing systems. The first feature is its peer-to-peer architecture. Current global computing systems such as Javelin++[1] and Bayanihan[2] are essentially center-based, when there are too many computing jobs managed by the center server, it becomes a bottleneck of the whole system, which may lead to single-point failure. Secondly, in most of other general-purpose global computing systems, users need to register first to contribute their resource and know addresses of other peers to submit jobs. On the contrary, the objective of Paradropper is to implement a unified computing network on which any one can submit (contribute) his/her computing jobs (computing resources) without knowing the underlying details described above. Finally, Paradropper is not only an embarrassingly parallel computing environment, but also supports pipeline parallel and cooperative computing.

2 Roles In Paradropper System

There are some different roles between volunteer peers who participate in the computing job. Paradropper extends roles flexible enough to adapt to various computing environment and computing devices. The one who submit the computing job is called *Submitter*, which loads a computing descriptor, divides it into many task descriptors and sends them to Paradropper network. The one who devotes to do tasks is called *Worker*. The one who is responsible for feeding computing data to workers is called *Rootfeeder*, and the one who presents the result is called *Presenter*. Because submitter just sends task descriptor to volunteer nodes, when these nodes accept these task descriptors, it will load tasklet through a network classloader from a remote node, which is called *Taskcenter*, though taskcenter can be any volunteer node, generally, it is the node on which the developer put his applications, the developer could select any node as the taskcenter. The one who is responsible for managing a concrete computing job is called *Manager*, it is different from the broker in center-based global computing system, because it is temporary, and only has limited information about those volunteers who participate in *a* *concrete* computing job. Actually, a peer can have many roles simultaneously in a certain computing job.

Generally speaking, in a concrete computing job, the rootfeeder, presenter, taskcenter and manager have fixed address and worker can be any volunteer node in Paradropper network. The rootfeeder usually works with a database or other application data sources, and these data sources sometimes have fixed IP address. So is the presenter, the result is usually presented on a fixed node. As we mentioned above, the taskcenter node usually is the developer's node on which the developer put his applications. The worker nodes devoted to do divided computing, i.e. tasks, and seldom need a fixed address. In a large scale computing job, the manager node usually has a heavy load, which acts as a controller of this computing job, and is responsible for heartbeat detecting, re-schedule when a participator failure and termination detecting, etc, thus it usually becomes a powerful node beside the user in order to monitor the status of the procedure of the computing job at any moment.

3 System Architecture

We first give some explanations about several terms appeared in this paper. *Computing*: A computing job is called a *Computing* in Paradropper system. Statically, a computing is a computing descriptor; dynamically, a computing is an object instance, which has some attributes and methods. *Task*: A sub-job is called a *Task* in Paradropper system. A computing consists of many tasks, which in turn are divided into sub-jobs. *Tasklet*: The entity that is used to execute a task is called a *Tasklet*. Tasklet is another form of task. Before a task is sent to a volunteer computer, it is called a task. Once a task is accepted by a volunteer computer, the volunteer will assemble it into a tasklet. Generally, a task has no address, no execute body (the java class), on the contrary, a tasklet has an address and some loaded java class. In other words, computing and task is logic executing unit and tasklet is the actual executing unit. For instance, once a volunteer accepts a task descriptor, it will assemble a tasklet object according the descriptor.

The Paradropper framework employs a highly object-oriented design. It defines a set of interacting components and uses them to build Paradropper system such as the one shown in Figure 1.

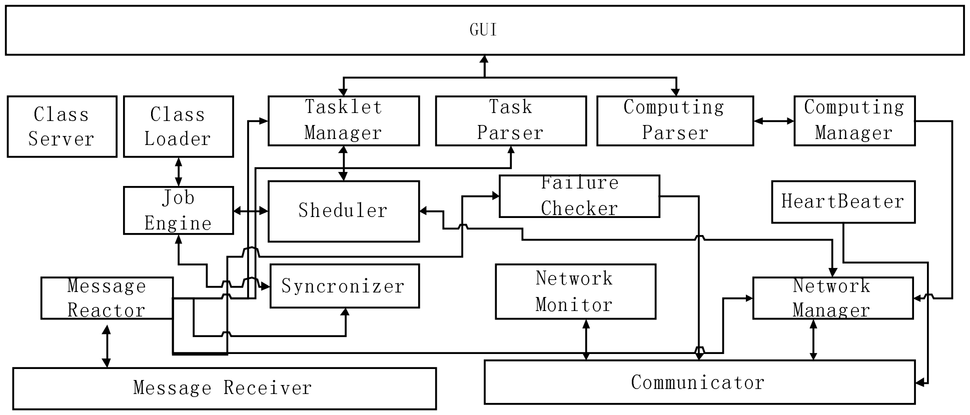

Figure 1 Paradropper Peer Architecture

4 The Constructing Of Paradropper Network

Many researches[3~5] have shown that such peer-to-peer applications as the Gnutella decentralized file-sharing network have some self-organizing network characteristics, which contribute to peer's easily and effectively searching for data. These characteristics include (1) high cluster coefficient (2) short path length and (3) power-law distribution, etc. All these characteristics help peers rely on limited local knowledge to make routing decisions rather than depend on global knowledge. Stanley Milgram[6] prompt the famous 6-degrees of separation notion by a social experiment. Watts and Strogatz[7] 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 which lead to the famous small-world phenomenon in almost every self-organizing networks. Unfortunately, Watts and Strogatz's network constructing approaches are based on regular network, not suitable for dynamic network constructing and maintaining. Other researches also prompt some approaches which are more suitable for dynamic network constructing, but these approaches seem to be dedicated to constructing low diameter network and without more concerning about cluster coefficient which is important in our system. In Paradropper, we develop a simple constructing approach, in which the entry point server randomly select an in-network volunteer address as the entry point for the new joining node, which in turn introduces the new node to part of its neighbors with relatively low outdegrees. The detailed algorithm is covered in another paper. Simulation result shows that, the constructing approach leads to a self-organizing network with high cluster coefficient and short path length.

In accordance with the "small-world" character of the underlining overlay network, we develop a simple static scheduling algorithm to search peer with less load efficiently. Suppose every peer knows its neighbor's workload via *Load Change Report Message (LCRM)*, the pseudocode of scheduling algorithm is as following:

```
receive a task (or the sponsor who first schedule the task);
    if(ttl>0){
        Select a neighbor whose workload is the smallest in neighbors;
        if the selected neighbor 's workload low than the peer's
            ttl minus by 1;
            Send the task to the neighbor;
        else
            Accept itself;
        endif
    else
        Accept it itself.
endif
```

Because Paradropper network has a short path length, in most cases, a peer could reach any peer in small hops (in our system, we set the ttl to 11). Simulation result shows that the workload is well balanced in the whole system.

5. Relationship-based Cooperative Computing

Different from other embarrassingly parallel computing environment, Paradropper is also a cooperative computing environment. When one of the tasklet fails, the manager will not just re-send the failure tasklet, but also re-send the related tasklets or re-get computing data. So it is necessary for a computing manager has relationship map of all tasklets who participate in the computing. Computing parser does this when a manager accept a computing descriptor. By computing descriptor, a concrete computing manager could build a relationship map among all the tasklets, a tasklet also could know which tasklets have an output or input relationship with it.

Generally, When a manager finds that a tasklet is dead, it will:

(1) *retrieve its relationship with other tasklet from the relationship map*
(2) *re-send the failed task*
(3) *send a Re-Prepare Data Message(RPDM) to any tasklet who have input relationship with the failed tasklet. The input relationship means this tasklet is one of the data source of the failed tasklet.*
(4) *When a tasklet accepts a RPDM message, it will fetch the relative data from its write cache, and begin trying to send the data to the re-spawned tasklet.*

6.Discussion

Apart from the above discussions, there are some trivial issues in Paradropper implementation, including eager scheduling, termination detecting etc. here we give some brief discussions. Besides static scheduling, Paradropper also uses an eager scheduling algorithm to balance the load of every peer in a computing dynamically. The workload information in a certain computing task is maintained by the manager via heartbeat message, and peer also uses Load Change Report Message (LCRM) to notify neighbors its workload. Accordingly, when a peer finishes its tasks, it will send a Request Work Message (RWM) to manager for new task or get task directly from its neighbor. On the other hand, We implement a 2-phase terminating algorithm which is closer to Bertsekas and Tsitsiklis approach to deal with the cooperative computing task. The first phase is running phase and the other is zombie phase. The manager is responsible for the whole computing termination detecting, every tasklet has four status: ready, running, zombie and halt. When a tasklet has finished it tasks, it will be put into Zombie queue by scheduler. Zombie tasklet is not active, but it does not release any memory resource, and can re-send related data to the new incarnation of failed tasklet. When all the tasklets become zombie, they at last transit to halt state.

7 Applications

we implemented a parallel primality test, which is used to search for Mersenne prime numbers. As we shall see, this type of application is well suited to Paradropper, since it is very coarse-grained with a high computation-to-communication ratio when testing large Mersenne primes. A Mersenne prime is a prime number of the form 2^p-1, where the exponent itself is prime. In our current implementation, we developed a Java application to test for Mersenne primality, given 5 numbers: 44241, 44237, 44253, 44267, 44243, we use 1,2,3,4,5 PIII 733 256M intel desktop PC as volunteer computers respectively. The speedup is shown in Figure2.

We also developed a sequential raytracer, and implemented a white board application to demonstrate the ability of paradropper of cooperative computing.

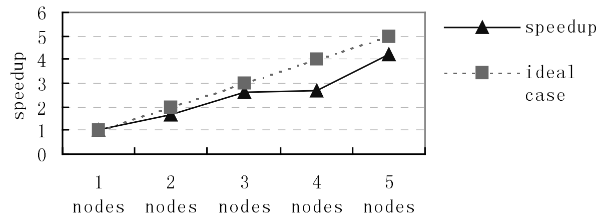

Figure 2. Mersenne Prime Application Speedup Test

8 Summary

As far as we know, Paradroper is the first general-purpose peer-to-peer global computing system which aims at providing a unified computing overlay network and unified GUI interface, and it supports more computing mode and is not restricted to embarrassingly parallel computing. It partly overcomes the bottleneck problem of whole computing network, but there still is computing center within a concrete computing (the computing manager), though any computing failure will not affect another computing processing in the network. Our next work will try to exploit this problem.

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

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

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

[6] Stanley Milgram: The Small World Problem, Psychology Today 1(1), 60-67 (1967)

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

IEEE COMPUTER SOCIETY