A Dynamic Hierarchical System for Large Scale Distributed Applications

Junichi Uekawa
Graduate School of
Doshisha University,
Kyoto, Japan
email: dancer@mikilab.doshisha.ac.jp

Tomoyuki Hiroyasu
Doshisha University, Kyoto, Japan
email: tomo@is.doshisha.ac.jp

Mitsunori Miki
Doshisha University, Kyoto, Japan
email: mmiki@mail.doshisha.ac.jp

Yusuke Tanimura
Graduate School of
Doshisha University,
Kyoto, Japan
email: tanisuke@mikilab.doshisha.ac.jp

ABSTRACT
A system for creating a dynamically generated hierarchical communication structure is designed. It is an hierarchical system with mechanisms to make the system more robust. The system has low network load due to restricted nature. Applications for system monitoring and solving optimisation problem are implemented to demonstrate the applicability of the system.

KEY WORDS
P2P, distributed computing, hierarchy, Genetic Algorithm

1 Introduction
The aim of the research is to create a robust computer network on which multiple computers can cooperate, and as a whole, create reliable output. A network structure to operate applications on top of with hierarchical structure as its logical construct is designed to accomplish the goal. By creating an hierarchical structure on top of a flat TCP/IP network, communication can be more concentrated between local machines, possibly decreasing overall network data flow. To compensate for the weakness of hierarchical structure, especially for the dependency to intermediate nodes of a tree, hierarchy is dynamically modified. The aim is to allow generic optimisation computation on top of this structure.

Many problems exist with high computation demands. There is still use for larger parallel computers. Parallel computers are computers that have more than one computational unit, such as machines that have more than one CPU. Parallel computers function with more than one arithmetic computation unit that exist independent of each other cooperating with others to solve a problem. This achieves higher performance than it is possible by designing a single high-performance CPU. PC Clusters[1][2] are parallel computers manufactured from mass-produced consumer PCs, interconnected with commodity networking.

The main characteristics of such systems are that they have a relatively high computational ability compared to their low network connection. PC Cluster nodes often have a significant physical distance between each nodes, and it is usually quite difficult to maintain a good shared memory model. The PC Clusters mentioned in this paper are the models based on distributed memory model.

A hierarchical logical network structure is a system where information packet from a node can only be sent to a restricted member of the network. The packet is not freely sent to random hosts. When the restriction is done in such a way as described in fig. 1, it forms a logical tree structure. Machines in TCP/IP network have the ability of sending messages to any host within the network, but the distance and ping time may vary. By restricting communication to certain hosts and modifying the hierarchy according to the real network structure, the most frequent communication can happen between the closer nodes. Each node processes the input, and sends the processed data to another node. These hosts are called servents, because they are a server to one and a client to the other. By being distributed like thus the central server is not overloaded with individual requests, nd for calculations which can work with collection of partial solutions, it is possible to calculate partial solu-
tions in servants, and accumulate the results at the central server. It is more limiting compared to P2P[7][8][9][4][3] model where anyone can communicate with anyone else in the network. However, it has the advantage of optimising network traffic to local communication that may be less costly, and resulting in more scalability.

In this research, a software to solve optimisation problems using the proposed network structure is presented. To consider the advantages of the hierarchical system, genetic algorithm[8] (GA) is considered. For GA, there is a model suited for distributed execution called Distributed Genetic Algorithm[9] (DGA).

GA is an algorithm where problems are applied to logic of genes and evolution, where parameters are what the genes represent, and individuals that have closer value to the optimal value have better chance of survival. DGA is an algorithm for operating GA, using a model of islands, GA operations are done within islands, and some individuals are communicated between them, modelling the real-life situation where far-away islands receive an occasional migrant visitor.

2 Background

In current age of computing powers, there are many desktop computers that have idle computing resource. It is possible to use the idle resource for doing extra calculation. To utilise the computational power in projects, it is necessary to devise a method to make them available to users of computational resources, and also a method to make the resources to cooperate is required. It is easy to use the resource singularly, but using them for doing parallel computing, combined to do a large task was somehow difficult.

To use many resources that may come and go, a fault-tolerant method of distributing a job is sought. Projects such as SETI@home[10] have managed to perform with such aim, and succeeded. However, there is no generic enough framework for such project, and systems are too specific for the problems.

When large number of machines are employed in the computation process, the server machine, the master node that coordinates the computation may become overloaded. To lessen the load, a method of distributing the calculation or the load is required. Constructing a hierarchical structure is one way of tackling such problem. Each system will serve few other systems and collect information from them, where each of them in turn serve other few systems each and collect information from them. This scheme is especially useful when problem can be split into subproblems, and main problem can be solved using the results of solving the subproblems.

Distributed Genetic Algorithm is a model where population are spread through islands. The islands communicate infrequently with each other. The result of search done by the genetic algorithm can be communicated as bit-data, and only a very little amount of data, namely the genetic information of few individuals, need to be communicated.

3 The system for using dynamical and hierarchical network structure

The system is constructed around the idea of creating a dynamically changing hierarchical network structure. This is a logical network structure, as opposed to physical actual wiring for the network. The physical network structure can be such that network packets are able to communicate to any host within the network. It is logically made hierarchical by creating restrictions on what host to send message to.

The topology construction should be scalable, and it is ideal if the formation of tree can be done with least communication of current topology.

3.1 Topology formation

The initial topology is given by the user. It can be a server-client type set up where all the clients are connected to one master system. From this state, systems can use reconnection algorithms to form a tree structure. This way, without much user consideration, the topology can be rearranged.

Manually defined hierarchy can be made optimal using human hands, but such an hierarchy is weak against damage done to intermediate hosts. Nodes that relay information exist in an hierarchical topology, and that host is the weakest part in this network. However, by creating measures to reconstruct network structures removing the defunct system, from the hierarchy it will be possible to have a network that survives node failures.

Each host sends information packet (fig. 2) tagged with hostname at the top, and tag-data pairs. Tag consists from start of line delimited by a colon, and the rest of the line is treated as data.
Figure 3. Information flow of Route-To: and Seen-By: message

```
while link (car(Route-To)) fails do
    Route-To = cdr(Route-To)
```

Figure 4. Algorithm to relink to uplink, when uplink is lost

3.2 Reconnecting Algorithms

To achieve the network structure, information about how the network is constructed is given. The information used here are Seen-By, Route-To, and Data-Seen.

Route-To information goes from uplink to downlink. This information is used to determine what hosts exist in the uplink side of a specific host. Each host adds itself to the Route-To information as the information is passed on.

Seen-By is a reverse of the Route-To. It is sent from the downlink to uplink, relayed upwards. Each host adds its own hostname to the Seen-By information. Each host will then have a collection of Seen-By information, which has a list of hosts that the information packet has passed through. It is then possible to reconstruct the current network structure. The flow of information is as shown in fig. 3.

Data-Seen is a flag that each host has against the hosts it serves, and its downlinks. Any information has this flag, and tagged as “Seen” if it has been propagated to the uplink. So that it would be possible to know that this information was sent up to the uplink once.

Applications reconnect using the algorithm shown in fig. 4 and fig. 5. In fig. 4, hosts detect that a connection to uplink is failing. Each host utilises the available Route-To information to find out available alternative hosts to connect to. The topmost item in the Route-To information is the direct uplink which stopped responding, that item is dropped, and connection to the grandparent node is attempted. If grandparent node is not available, the uplink to that node is attempted, proceeding one-by-one through the Route-To information until a host that responds is found. It will fail at the root node, if no host responds.

There might be problems with too many hosts linking to the same uplink when algorithm in fig. 4 is applied, since the algorithm always tries to connect hosts to upper links when failure happens. Hosts higher in the tree will need a way to lessen the load. In the current implementation, each host has a limit on how many nodes it serves. When the number of direct downlinks exceeds the limit, two arbitrary downlinks A and B, are selected. Then a fake Route-To message is sent to host A from the uplink, pretending that A is talking to B. Host A then will consider the uplink to be B, and will connect to host B from the next iteration. The algorithm is shown on fig. 5. This method allows decreasing the number of hosts connecting to the same host at the same time. To avoid conflicting simultaneous operation which can cause looping network, only one relink from a host is allowed at one time.

4 Distributed Genetic Algorithm

To demonstrate that some useful calculation can be done upon this system, a system to perform computation using Distributed Genetic Algorithm was implemented. Genetic Algorithm is a optimisation method. It is based on theory of evolution, where individuals which have higher fitness to the environment survive, and cross over, to produce offsprings, eventually evolving into individuals which are fit to live in the environment after several generations. It is a probabilistic searching method with multiple point search. It is known to work well with optimisation problems when its parameters are independent of each other.

4.1 Conventional DGA

A typical conventional DGA forms some random topology such as a ring topology, where hosts communicate individuals to the consecutive member of the ring. The individuals are communicated between random islands. Migration occurs every few generations, and almost half of the individuals migrate to other islands. This assumes that the network system is homogeneous, and cost of communicating is not high.

4.2 DGA on hierarchical network

On top of the tree structured system, a variant of DGA is constructed. Each servant represents an island, and individuals are distributed on each node. Each servant per-
forms GA operation. Communication of individuals are done through uplinks as shown on fig. 6.

In this system applications run on each node, communicating directly with uplink. Each node has a pool of individuals. Random individuals are communicated to the uplink. The uplink pools the individual, indexed with the hostname, and forwards them to the uplink. When there are old individuals from the last migration from the same host, they are overwritten with the new individual. Each node polls the uplink for individuals, and the pooled individuals reach the downlink.

Each child node sends their individual to uplink on every migration interval, and tries to retrieve individual information from the uplink. The logic is summarised in fig. 7. The uplink pools the received individuals indexed on hostname, and keeps only one copy of the individual per each servent host. Uplink responds to retrieval request with sending back the individuals that were received from the downlink. The uplink node logic is summarised in fig. 8. Nodes communicate with each other through their uplinks, as shown in fig. 6. Uplink relays information to further uplinks, and individual information is replicated. The interaction of the hierarchy is briefly shown on fig. 6.

5 Experiments

Some experiments for verifying the network system functionality was done by running DGA process on the system. More than 200 computers connected via 100BASE-TX connection was used in carrying out the experiments.

The test function used in the DGA experiment is a Rastrigin function, coded in 5 variables of 32-bits using bit-coding method, totalling in 160 bits. The problem was to find the maximum value of the function, which is $x_i = 0$ for all $i$. The range of the variable $x_i$ was $[-5.12, 5.12]$. The function used is shown in (1).

$$f = -\left(10n + \sum_{i=1}^{n} x_i^2 - 10\cos(2\pi x_i)\right)$$  \hspace{1cm} (1)$$

There are 100 individuals per island. Every individual is operated through one-point cross over operation in one generation, and mutation happens at 0.1%.

5.1 Genetic Algorithm Hierarchy

In the hierarchical system, DGA was ran on multiple machines. The topology was left to form with using the rule that only 4 downlinks allowed on each system.

To verify the scalability of this system, the DGA system was ran on 10 to 160 nodes. The average fitness values of the individuals after 2000 iterations is shown in fig. 9. Migration was done on every generation. The results suggest that the DGA process gained better results with more nodes available. With more nodes, the search is done with greater number of individuals, namely $100 \times n$, where $n$ is
the number of nodes. The result suggests that the effect of increasing the number of nodes to the result may be saturated after 60 nodes.

To see the effect of migration frequency an experiment was done on a 5-node set up with differing migration intervals. Result is shown in fig. 10. The result shows that migration interval less than 10 is significantly better than the rest. The result shows that migration rate affects the hierarchical DGA result. This suggests that frequent migration is having a good effect on the DGA process.

An experiment was done to verify the effect of the topology to the DGA operation. The topology was manually created and fixed as shown in fig. 11, using 6 servents and 1 master node. The result is shown in fig. 12. It shows that none of them reached the optimal value, and was not good enough to reach the optimal value reliably. The value shown is the average value of 10 trials, and although it is not visible from the result shown, they do reach the optimal result sometimes. The result suggests that topology 3 is quite fast in converging to the solution while topology 2 and 1 is not very good. This result shows that the balanced tree is not necessarily the optimal structure in this hierarchical model of DGA.

5.2 Fault-tolerance results

To experiment with fault-tolerance, system was put to an endurance test. 7 hosts were used and evaluation was done with 6 islands. The line marked as “fitness value without node failure” in fig. 13 shows a result of such set up. DGA process was run for 2000 generations, and terminated. It was then resumed for another 2000 generations using fresh DGA process, which used the information left over from previous iterations on the servents. Another such run was done. In total 6000 generation of calculation was done. This shows that DGA search is able to pick up a good value from system, and continue searching.

Another experiment was done, with 3 iterations of 2000 generations each. This time, an artificial network problem was introduced, so that servent was not available at one of the nodes in the middle 2000 generation. Out of 6 servents, the servent process running one node was explicitly killed, and DGA process was ran while it was down. It can be seen that without network interaction, where simple GA iteration was performed, the fitness value did not reach a good value. However, when the system is reconnected to the network, it was possible to attain a better fitness value.
5.3 Analysis of results

The results in fig. 9 and fig. 10 suggest that the system is useful in applying DGA. Migration process and having more nodes affects the outcome of DGA operation in this system, meaning that migration is contributing to the DGA search. The system shows some actual fault tolerance as in fig. 13. The result in fig. 12 suggests that the current re-linking mechanism is not necessarily suboptimal, and that current re-linking algorithms shown in fig. 4 and fig. 5 are useful.

6 Conclusion

An implementation of DGA was executed on top of the dynamic hierarchical system, and the scalability and fault tolerance of the system was demonstrated.

This dynamical hierarchical system is fault tolerant and creates hierarchy in a dynamical manner.

The shortcomings of this system is that although the application interface is simple there are not many other applications yet. This system does not try to produce very low level infrastructures such as process migration. The logical hierarchical network structure can be dynamically reconfigured to be optimal for the physical network structure through benchmarking, but the actual implementation of such system is yet to be researched.

References


