

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, "DNAS", is designed. DNAS is a middleware for Grid applications that need communications. It is a hierarchical system with mechanisms to make the system more robust. The system has low network load due to restricted nature. An application for genetic algorithm is implemented to demonstrate the applicability of the system.

KEY WORDS

Internet Computing, Fault Tolerance, Hierarchy, Genetic Algorithm

1 Introduction

Recent advancement in networking has made more machines available to the network, with much excess computing power. The grid computing is a kind of infrastructure like the power grid. The grid computing is the operation form of the application based on a network. In the grid computing, a lot of computers and huge super computers are connected by network, especially by the NET. Users can use these computational resources as a system. Therefore, they can treat tremendous data and can operate huge application. However, there is a big network latency between nodes, while computational power of each node is large. Therefore, not all applications but only some of them are suitable for the grid computing. The characteristics of these applications which match the characteristics of the grid, and have the potential to be ran effectively on the grid can be summarized as follows,

- Jobs can be divided on multiple parts, and the individual portions can be executed independently from each other on different nodes.
- The divided jobs running on each nodes may require some communication between them, but the require-

ment is modest.

- Low cost for continuing the job when some nodes are removed, and the job will utilise new nodes added to the system.

We call applications that have these characteristics as " Grid Oriented Computing Application (GOA) ".

As an example of GOA, there are projects like SETI@home[1], and Folding@home[2]. There are also Grid RPC systems, such as NetSolve[3], Ninf[4], Nimrod/G[5] and so on. Some applications can be worked on these Grid RPC. However, these projects target at problems that do not require communication between the individual nodes or assume a low latency high-speed interconnect. There is currently a lack of projects which consider high latency network communication between nodes on running GOA type applications.

In this paper, for running GOA type applications, a system called Distributed Network Application System (DNAS) is proposed.

DNAS has a hierarchical logical network structure. 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. It is more limiting compared to P2P[6][7][8][9][10] model where anyone can communicate with anyone else in the network. However, it has the advantage of optimizing network traffic to local communication that may be less costly, and resulting in more scalability. Some researches on applications using hierarchical network exist. DNS is a hierarchical protocol to provide name to number matching service. It handles a mostly static data with caching technology. It has been fine-tuned for name services, but it is not suited for running applications on them. IRC[11] is an hierarchical network but it has been tailored for data broadcasting technology. There are dynamic routing protocols but they do not consider running applications on them. DNAS is different in that its aim is in

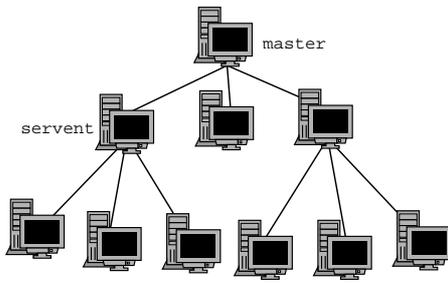


Figure 1. A logical tree-shaped network structure

running a GOA. The interfacing between GOA and DNAS is strong, but DNAS itself remains simple in nature.

To consider the advantages of the hierarchical system, genetic algorithm[12] (GA) is considered. For GA, there is a model suited for distributed execution called Distributed Genetic Algorithm[13] (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 performed for several generations within islands. After the generations, some individuals are communicated between them. This is the modeling the real-life situation where far-away islands receive an occasional migrant visitor. Since the network communication between the islands does not happen frequently, GA is one of GOA. Through the simulation, the operation of DNAS is confirmed and results of DGAs on GOA are discussed.

2 DNAS

Our goal is to create a middleware system for running GOA to utilize the excess computing powers. On this system, GOA that needs communication between nodes can be applied. In this paper, a system called Distributed Network Application System (DNAS) is implemented. The requirement for DNAS is as follows:

- It allows GOA to run on multiple machines
- Automatic determination of the computational network topology
- Avoid overloading a machine when other jobs are running on a node.

2.1 Data communication in DNAS

A hierarchical logical network structure forms a logical tree structure like fig. 1. DNAS requires a central server to exist, which is called a DNAS master system, and the child system called downlinks, which can serve any number of children themselves. Each node processes input, and sends the

```
Info-packet :- meta-header each-host-info*
each-host-info: host-name host-info
host-name:- ":hostname:"
host-info:- tag-data*
tag-data :- "tag: data"
```

Figure 2. Information packet

processed data to another node closer to the master node, called uplinks. 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 sever is not overloaded with individual requests. Calculations can work with the collection of partial solutions, therefore, it is possible to calculate partial solutions in servents and the results are accumulated at the central server.

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.

2.2 Topology formation

The initial topology is given. It is an arbitrary set up which can be a server-client type where all the clients are connected to one master system. From this state, DNAS servents use reconnection algorithms to form a tree structure.

Manually defined hierarchy can be made optimal using human hands, but such an hierarchy is weak against changes. Nodes that relay information exist in an hierarchical topology, and that the network is weak against removal of such nodes. However, by creating measures to reconstruct network structures by removing the defunct system from the routing, it is possible to have a network that survives node removals.

Each host sends information packet (fig. 2) with host-name at the start, followed with tag-data pairs. Tag and data are text delimited with a colon, and paired in a line-based manner.

2.3 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.

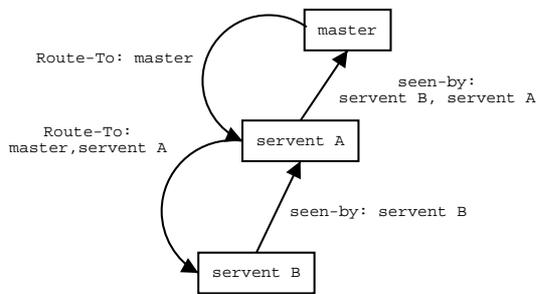


Figure 3. Information flow of Route-To: and Seen-By: message

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

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

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 along with other tag-data pairs. 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 an internal flag that each hosts have for each served hosts, 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 information is already 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 closer to the root of tree will need a way to lessen the load. In the current implementation, each host has a limit on how many nodes it serves.

```
D=list of active downlinks
A=random member from D
B=random member from D
if A!=B then
  set Route-To[A]=B+Route-To[A]
```

Figure 5. Algorithm to relink on too many downlinks

When the number of direct downlink 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 operations which can cause looping network, only one relink from a host is allowed at one time.

3 DGA on DNAS

To demonstrate that some useful calculation can be applied upon the DNAS system, a system to perform computation using Distributed Genetic Algorithm was implemented. Genetic Algorithm is a optimisation method and can be an example of GOA with suitable implementation. It is an algorithm based on theory of evolution, where individuals which have higher fitness to the environment survive to cross over, to reproduce offsprings, eventually evolving into individuals which are generally fit to live in the environment after generations. It is a probabilistic searching method with multiple point search. It is known to work well to find a global optimum and easy to apply to several types of optimization problems.

3.1 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.

3.2 DGA on hierarchical network

On top of the DNAS system, a variant of DGA is constructed. Each servent represents an island, and individuals are distributed on each node. Each servent performs GA operation. Communication of individuals are done through uplinks as shown on fig. 6.

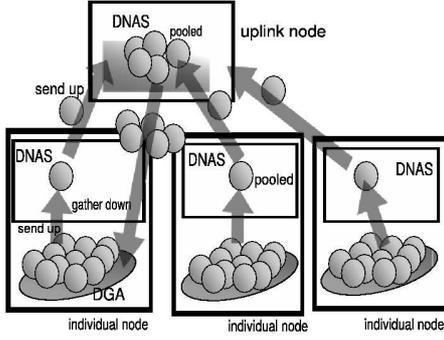


Figure 6. Implementation of DGA

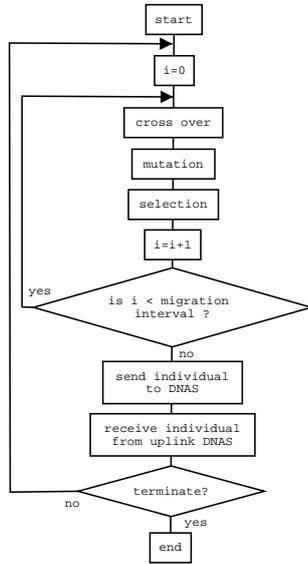


Figure 7. Flow chart for client side logic for the DGA model

In this system the GOA runs on each node, communicating directly with DNAS. Each node has a pool of individuals. Random individuals are communicated to the uplink, and the uplink passes the information upward. The uplink pools the individual, indexed with the hostname. 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 the local DNAS on every migration interval, and tries to retrieve individual information from the uplink DNAS. The logic is summarised in fig. 7. The uplink DNAS pools the received individuals indexed on hostname, and keeps only one copy of the individual per each server host. DNAS responds to retrieval request with sending back the individuals that

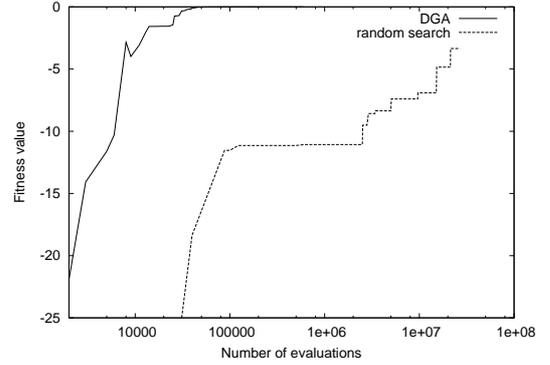


Figure 8. Searching progress of a random search program and a DGA program running on 10 nodes on a DNAS system

were received from the GA process. Nodes communicate with each other through their uplinks, as shown in fig. 6. DNAS relays information to further uplinks, and individual information is replicated.

The operations are fully asynchronous and do not depend on states of other nodes. The constructed DGA system satisfies the defined qualities for being a GOA and can be considered an example of GOA.

4 Experiments for DGA/DNAS

Some experiments for verifying the network system functionality was done by running DGA process on the system. In this system, 256 computers connected via 100BASE-TX connection was used in carrying out the experiments. In this paper, this system is simulating the grid computation environment.

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).

$$f = - \left(10n + \sum_1^n x_i^2 - 10\cos(2\pi x_i) \right) \quad (1)$$

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

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 see the effectiveness of DGA, a random-search program which searches the optimal design variables using a random number generator was ran on 10 nodes. The result of the run is compared with DGA program running on

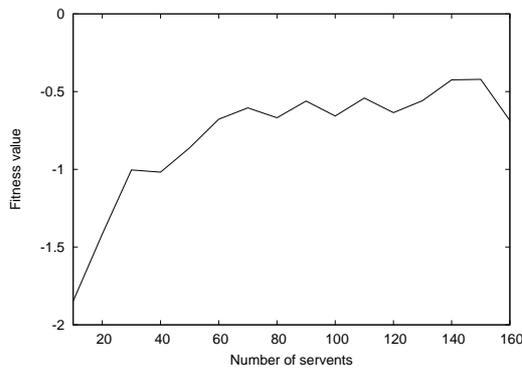


Figure 9. Fitness value after 2000 generation running with 10 to 160 nodes.

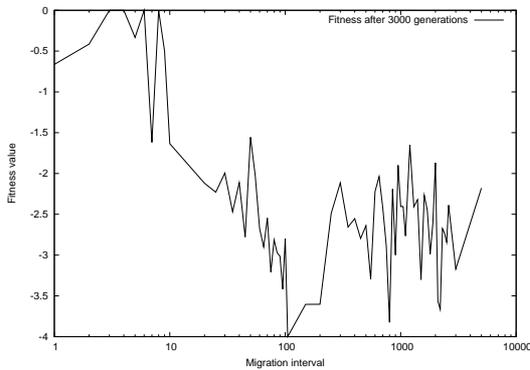


Figure 10. Fitness value after 3000 generations on 5-node set up, with 4 nodes representing islands of 100 populations each. Comparison is done based on different migration rates. Average of 5 tries.

10 nodes. The application was run on DNAS system, and observed from remote network through the master node. The result in fig. 8 shows DGA performing better.

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 performed on every generation. The results suggest that the DGA process gained better results with more nodes available. With more nodes, the search is performed 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.

An experiment was executed on a 5-node set up with differing migration intervals to see the effect of migration. Result is shown in fig. 10 which shows that migration interval less than 10 is significantly better than the rest. The result shows that migration rate does have an effect. This suggests that frequent migration allows earlier arrival to a better fitness value.

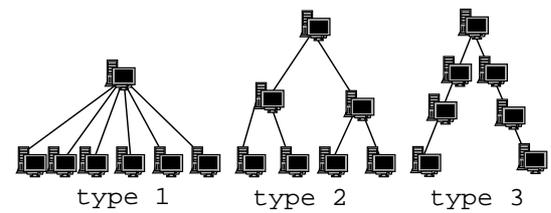


Figure 11. Three different topologies

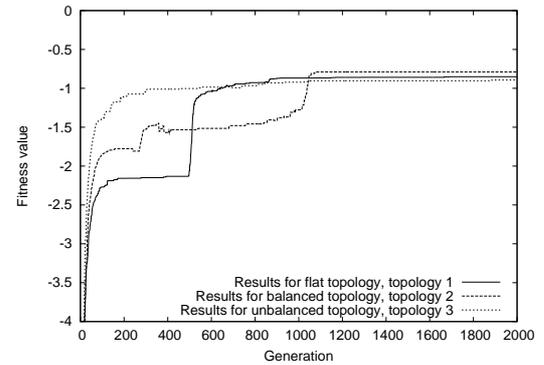


Figure 12. System having different static topologies, and running DGA process for 2000 generations

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 servers 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.

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 servers. 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 server was not available at one of the nodes in the middle 2000 generation. Out of 6

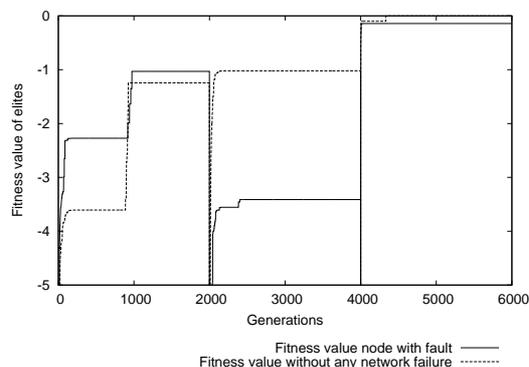


Figure 13. Comparison of DGA with resume at 2000 generations, running total of 6000 generations. Average value of fitness on all hosts. 7 hosts are used, with 6 islands. Comparing the effect of node with fault. One trial with manual fault in one host, and another without. Average of 10 tries.

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.

The results in fig. 9 and fig. 10 suggest that the DNAS system is useful in applying DGA. However, the scalability of the GOA stops at about 60 nodes. It is required that GOA that comes on DNAS to be with a better scalability.

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 relinking mechanism is not necessarily suboptimal, and that current relinking algorithms shown in fig. 4 and fig. 5 are useful.

According to these results, GOA is able to run even when DNAS has failed to communicate with other nodes.

5 Conclusion

DNAS, a system to construct hierarchical computing network for executing GOA was constructed. Some experiments were done and DNAS proved to be useful in running a GOA application, DGA. In the process, requirement for GOA became apparent, that the DGA process used in the experiment did not have enough scalability, but DNAS itself proved to be quite tough against failures. The experiment was performed on PC clusters consisting of 256 nodes. The experiment in the real grid computation environment is the future work.

Acknowledgements

This work was supported by a grant to RCAST at Doshisha University from the Ministry of Education, Culture, Sports, Science and Technology.

References

- [1] SETI@home: search for extraterrestrial intelligence at home, <http://setiathome.ssl.berkeley.edu/>.
- [2] Folding@home: Folding@home Distributed Computing, <http://folding.stanford.edu/>.
- [3] Arnold, D., Agrawal, S., Blackford, S., Dongarra, J., Miller, M., Seymour, K., Sagi, K., Shi, Z. and Vadhiyar, S.: Users' Guide to NetSolve V1.4.1, Innovative Computing Dept. Technical Report ICL-UT-02-05, University of Tennessee, Knoxville, TN (2002).
- [4] Sato, M., Nakada, H., Sekiguchi, S., Matsuoka, S., Nagashim, U. and Takagi, H.: Ninf: A Network Based Information Library for Global World-Wide Computing Infrastructure, *HPCN Europe*, pp. 491–502 (1997).
- [5] Rajkumar Buyya, David Abramson, J. G.: Nimrod/G: An Architecture for a Resource Management and Scheduling System in a Global Computational Grid, *The 4th International Conference on High Performance Computing in Asia-Pacific Region (HPC Asia 2000)*, IEEE Computer Society Press (2000).
- [6] Stoica, I., Morris, R., Kaashoek, F. and Balakrishnan, H.: Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications, *SIGCOMM '01, August 27-31, 2001, San Diego, California, USA* (2001).
- [7] Sun Microsystems: *JXTA*, <http://www.jxta.org>.
- [8] Wego Systems, Inc.: *Gnutella*, <http://gnutella.wego.com> (1999).
- [9] Napster: *Napster*, <http://napster.com> (2001).
- [10] Freenet Project: *Freenet*, <http://freenetproject.org>.
- [11] Oikarinen, J.: RFC1459: Internet Relay Chat Protocol, <http://www.ietf.org/rfc/rfc1459.txt> (1993).
- [12] Goldberg, D. E.: *Genetic algorithms in search, optimization and machine learning*, Addison Wesley, Reading, Massachusetts (1989).
- [13] Tanese, R.: Distributed Genetic Algorithms, *Proc. 3rd International Conference on Genetic Algorithms*, pp. 434–439 (1989).

The paper is presented in :
Proceedings of the 14th IASTED International Conference,
Parallel and Distributed Computing and Systems, pp. 422–
427 (2002)