PAPER *Special Section on Parallel and Distributed Computing and Networking*

# A Swarm Inspired Method for Efficient Data Transfer

**Yutaka KAWAI**[†a)], ***Student Member***, **Adil HASAN**[††b)], **Go IWAI**[†c)], **Takashi SASAKI**[†d)], ***and*** **Yoshiyuki WATASE**[†e)], ***Nonmembers***

**SUMMARY** In this paper we report on an approach inspired by Ant Colony Optimization (ACO) to provide a fault tolerant and efficient means of transferring data in dynamic environments. We investigate the problem of distributing data between a client and server by using pheromone equations. Ants choose the best source of food by selecting the strongest pheromone trail leaving the nest. The pheromone decays over-time and needs to be continually reinforced to define the optimum route in a dynamic environment. This resembles the dynamic environment for the distribution of data between clients and servers. Our approach uses readily available network and server information to construct a pheromone that determines the best server from which to download data. We demonstrate that the approach is self-optimizing and capable of adapting to dynamic changes in the environment.

***key words:*** *data grid, ACO, swarm intelligence, grid computing, file system, iRODS*

## 1. Introduction

Swarm intelligence [1] is inspired primarily by observations of the collective behavior of social insects in addressing complex distributed problems. The basic idea is that each member of the swarm has simple rules that govern its behavior, but the interaction among the members of the swarm can be used to tackle problems that are difficult to solve with complicated numeric methods. In this paper we investigate the problem of data distribution between a client and server in a dynamic environment. We regard each download from the server to the client as a single member in a swarm. The member's behavior is simply to reliably download a data file. Each member can communicate with other members to allow the swarm to settle on the best set of servers to download the data from based on the current status of the environment.

Some research work generates large numbers of small files and then interacts with the generated small files in groups. For example, the T2K ND280 [2], [3] group consists of hundreds of researchers in 12 countries and 62 research institutes, all of whom must reliably download their own data files. They are using iRODS (The Integrated Rule-Oriented Data System) [4], [5], which provides each user with a virtual file-system that maps to distributed storage systems. iRODS has been developed by the Data Intensive Cyber-Environments (DICE) [6] team and collaborators and is based on more than a decade of experience with distributed data management systems ([4]). Different iRODS installations can be federated together to provide a larger virtual-file-system while allowing each member of the federation complete control over access and management of their own iRODS. This approach also allows client applications to interact with the data. Our implementation uses iRODS i-commands to download files from each system.

In Sect. 2 we describe the type of swarm algorithm we have experimented with and Sect. 3 describes our specific problem. We refer to some previous studies in Sect. 4. We define our pheromones in Sect. 5. The algorithm to select the best server by using the pheromones is described in Sect. 6. Section 7 describes our simulation and its results and Sect. 8 describes the implementation and test results. Section 9 summaries our work and describes some next steps.

## 2. Ant Colony Optimization

Ant Colony Optimization (ACO) algorithms [7] are a type of swarm intelligence. They are based on the behavior of foraging ants in which individual ants search in a seemingly random manner for food. As an ant searches it leaves pheromone or scent that records on its discovery of a food source and the path used during its return to its nest. The amount of the pheromone reveals information about the nature of the food source. Subsequent ants follow the pheromone trail and also reinforce it when they return to the nest with food. There may be multiple trails to the food source, but after some time the ants will converge on the most direct path between the source and the nest. This is due to the evaporation of the pheromone, since longer paths will have weaker intensities of pheromones and will be less likely to be followed.

In solving complex problems ACO algorithms use computational agents (representing the ants) that perform simple tasks. Each agent constructs a candidate solution that is communicated to other agents via a probability (the pheromone element) that is based on the components used to

Manuscript received January 6, 2012.
Manuscript revised April 19, 2012.

[†]The authors are with Computing Research Center, High Energy Accelerator Research Organization (KEK), Tsukuba-shi, 305–0801 Japan.
[††]The author is with the Dept of English, University of Liverpool, Liverpool, UK.
a) E-mail: yutaka.kawai@kek.jp
b) E-mail: adilhasan2@gmail.com
c) E-mail: go.iwai@kek.jp
d) E-mail: takashi.sasaki@kek.jp
e) E-mail: yoshiyuki.watase@kek.jp

DOI: 10.1587/transinf.E95.D.2852

construct the solution. For example, in the travelling salesman problem the probability is based on the edges between cities. Each probability contains a weight based on the heuristic information for the current problem. The weight represents the evaporation factor and reduces the probability for each ant's solution by a defined amount. The role of the weight is to eliminate local or intermediate solutions and reinforce the global or true solution.

## 3. The Data Distribution Problem

A common problem in almost any field that requires the processing of quantities of data is the movement of data from the storage systems to the computational systems where the data can be processed as quickly and reliably as possible. The problem is compounded by the dynamic nature of the environment in which the client is operating. The activity of each server can vary over time, the network activity can vary over time and the activity of each client can vary over time. In some cases network status information is coupled with server information through a broker service to guide the client to the best server [8]. However, these services require each server to publish the necessary information in order for the clients to make decisions. Since the servers cannot anticipate all of the needs of each client, it is possible that crucial information for a client will not be published by the server.

We argue that such a priori in formation, although necessary, is actually encoded in the 'full' transfer rate from client to server. The 'full' transfer rate is simply the time taken for the client transfer application to complete a transfer. This includes the overhead of staging the data onto a disk on the server and finalizing the transfer on the client (such as calculating a check-sum for the downloaded data). We believe that this is a better metric since the client is often interested not only in transferring the data as quickly as possible, but also in using the data as quickly as possible.

## 4. Related Work

### 4.1 ACO Related Work

The main area where swarms and ACO algorithms have been used is in optimizing distributed computational processing [9]–[11]. In such cases Particle Swarm Optimization (PSO), which is based on the flocking behavior of birds can optimize computational job submissions to the most efficient and least loaded nodes. Ant Clustering Algorithms (ACA) have been used to address the problem of clustering data in which related data should be clustered together (or co-located) for more efficient access (see [12], [13]). Data clustering is crucial for data mining where the data is studied for patterns and relationships.

As in the case of ACO an ACA agent possesses simple behaviors and the interactions between agents allow complex problems to be solved. The ACA was based on studies of ant cemeteries where worker ants sort the deceased ants according to their size and function. Each ant works individually to arrange the dead ants in its local vicinity into a uniform group. Global sorting is done by the deceased ants on the edges being sorted by the neighboring worker ant. The ACA works by having each agent sort the data within a restricted vicinity (typically a $3 \times 3$ grid) so that all the data within that vicinity is of the same nature (where the nature is defined by the current problem). The data on the edges between neighboring agents is sorted first by one agent and then by the other (and then by their neighbors until they match a pile). The final result is clusters of data with similar properties.

In the area of data distribution using swarms the work by Peterson and Sirer [14] investigates the problem of data distribution in a peer-to-peer network. Peer-to-peer networks operate in a non-privileged manner where there is no central server and each client is also a server of data. The paper described the development of Antfarm, a system that manages the bandwidth usage of each server for the optimal download rates by a swarm of clients. The Antfarm system consists of coordinators that use information from seeders and peers to control the bandwidth for the peers downloading data such that the data is downloaded to members of the swarm in the most efficient manner possible. The system also encourages downloads between peers to distribute the bandwidth requirements. This paper differs in that the main focus of this work is the problem of optimizing upload and download performance in a client-server environment.

Ant Colony Optimization has been studied in peer to peer networks by Wang Zhao and Hu [15] who looked at the problem of data replication optimization so that the data would be replicated to the peers that could make the most efficient use of the available resources. Each agent used the host latency, storage space and bandwidth as ingredients in the pheromone to determine the best placement for all of the data on all of the available hosts. The ACO then globally optimized the placement of the replicas by allowing each agent to choose a placement based on the previous agent's attempt. The placement was governed by the strength of the pheromone at each site. The optimization finished when the agents did not return a better arrangement. The placement of replicas has similarities with the work described in this paper except that the global optimization was done only one time.

### 4.2 Compared with Other Services

There are some other services for redundancy mechanisms in distributed data systems. The Contents Delivery Network (CDN) [16] and load-balancing are well-known examples.

A typical CDN application tries to find hosts are located at the fewest number of hops from the client and it selects the best host to optimize the download performance. A typical application of load-balancing is to provide a single Internet service from multiple servers. However, both cases require installations of software and services on the server side to manage the client load. Our approach does not require the servers install any software. Our approach is client-based system and there is no impact or changes on the server-side. Also, our approach imposes no overhead on the server-side, but it offers advantages to the clients to get the data more efficiently when it resides in a number of different locations. As long as a user has access to the data (either through iRODS or any other file system) then the user can use the ACO to obtain the data in an optimal way (as long as there are multiple copies of the data).

## 5. Pheromone Definition

The essential component of the ACO is the pheromone. Ants collect their food using their pheromone. The environment around ants is similar to the environment of clients collecting data from servers (Fig. 1).

The pheromone indicates to the agents which are the more promising paths to use in constructing a solution to a problem. In our case the pheromone is a metric of the viability of the server to serve the data to the client in as short a time as possible. To encourage a quick convergence to a solution we first determine the server availability to handle requests to download data. The availability is dependent on the load on the server and on the network.

A 'ping'-like application that sends a light-weight query to the server can assess the viability of the server (We describe an example of a 'ping'-like application in Sect. 8.1). The servers would then be ranked according to their responses to the 'ping'. It is important to point out that the application should ping the server application that serves the data and not the server itself since the application may be overloaded or down whilst the server is only moderately loaded or up and still able to respond quickly to a ping request.

However, ranking servers according to their response times to a lightweight query is not sufficient to optimize the download performance. It is possible that a server may respond quickly to a lightweight query, but may be either unable to serve the data due to some component of the storage system being offline (in the case of a compound storage system with a disk cache and a tape store where it is possible the tape store may be offline), or the storage resource being very busy (possibly due to high fragmentation in the case of disk storage systems). To address such situations we devised a pheromone element based on a transfer rate metric.

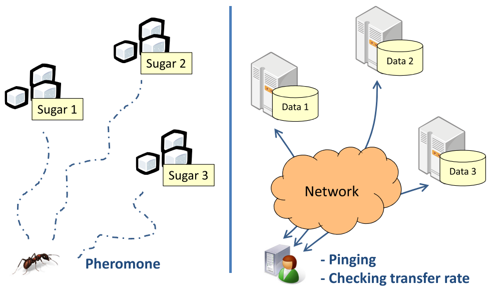

**Fig. 1** Environments of ants-foods and clients-data.

The rate is the inverse value of the complete download time measured from the time that the download application starts to the time that it finishes. This rate is necessarily smaller than the actual transfer rate because it includes the download time for the server to fetch the data from its system, serve it to the client, and any time required for the client download application to prepare the data for use. In this paper, we do not care about the actual transfer rate in the network but we only consider the inverse value of the complete download time. Therefore, we redefine the inverse value of the complete download time as the 'TransferRate'.

### 5.1 Pheromone Element

A pheromone's current value is based on the historical pheromone values so the base pheromone element must be defined first. We define the set $\mathcal{S}$ that includes all of the servers we want to use. $M$ is defined as the number of servers in $\mathcal{S}$, and $p_i$ expresses the pheromone element of $s_i \in S, 0 \le i \le M$. The pheromone element is given by:

$$n = 1:$$
$$p_i(1) = \frac{(CurrentTransferRate)_i}{\sum_{i=0}^{M}(CurrentTransferRate)_i} \tag{1}$$
$$n > 1:$$
$$p_i(n) = \frac{(CurrentTransferRate)_i}{\sum_{i=0}^{M}(PreviousTransferRate)_i} \tag{2}$$

where *CurrentTransferRate* is the rate used by the download application to start and complete for a given server. The *PreviousTransferRate* is the rate taken by the download application for previous transfers. The $n$ corresponds to the number of files downloaded from a given server. The first time a file is downloaded no prior history exists and the pheromone element $p_i(1)$ appears as a weight of the current TransferRate as shown in the first equation (Eq. (1)).

The pheromone element value is calculated immediately after downloading a file and is stored in a information file. This will be explained in Algorithm 2 in the next section.

### 5.2 Pheromone

Now we can define the signature of the pheromone elements. We simply call it "pheromone". The $h$ is given as the number of $p_i$ histories. The capital $P_i$ expresses the pheromone and it is given by:

$$1 \le n \le h:$$
$$P_i(n) = \sum_{k=0}^{n} p_i(k) \tag{3}$$
$$n > h:$$
$$P_i(n) = \sum_{k=n-h}^{n} p_i(k) \tag{4}$$

The pheromone value is calculated by reading the information file just before downloading a file. This is described in Algorithm 1 in the next section.

**Table 1** The example of an information file (i.e. n = 10, h = 4).

| $i$ | Server Name | iping Time | Download TransferRate | $p(6)$ | $p(7)$ | $\cdots$ | $p(10)$ |
|---|---|---|---|---|---|---|---|
| 0 | `Host01.kek.jp` | 3.4437897 | 42.4323076923 | 0.244369558 | 0.244369558 | $\cdots$ | 0.244369558 |
| 1 | `Host02.kek.jp` | 5.16568455 | 28.2882051282 | 0.172496159 | 0.172496159 | $\cdots$ | 0.172496159 |
| 2 | `Host03.kek.jp` | 4.2385104 | 34.47625 | 0.209459621 | 0.209459621 | $\cdots$ | 0.209459621 |
| 3 | `Host04.kek.jp` | 4.7683242 | 30.6455555556 | 0.148102762 | 0.148102762 | $\cdots$ | 0.148102762 |
| 4 | `Host05.kek.jp` | 8.10615114 | 18.0267973856 | 0.2255719 | 0.2255719 | $\cdots$ | 0.2255719 |

## 6. Algorithm

Our approach selects the best server using pheromone information before a client tries to download a file from a server. It also requires an information file to record each server's information and to update the information file immediately after the download. Our ACO agent uses these algorithms:

### 6.1 Algorithm to Select the Best Server

The best server is obtained by using Algorithm 1. An example of an information file is shown in Table 1. We created the command 'iping' as an example of a 'ping'-like application that checks the responses from the servers. In this example, the units for the iping values of Time and TransferRate are msec and MB/sec, respectively. The set $\mathcal{S}$ has all of the servers that are listed in the information file, *infoText*. The *infoText* file also has the historical pheromone element values ($p_i$) for each server that were previously defined in the equations (Eq. (1), Eq. (2)). Reading $P_i$ means to read the required $p_i$ from *infoText* and calculate $P_i$ as defined in the equations (Eq. (3), Eq. (4)). The $n$ corresponds to the number of downloads in progress at that time.

The iping Boundary Time ($ipBT$) is a fixed reference value for the iping results and is set at the hypothetically best response time. This helps to filter out servers in the *srvList* that have unacceptable response times (either because they are busy and cannot respond within an acceptable time or because they are offline).

### 6.2 Algorithm to Update the Information File

While executing a download, the given server becomes the *bestServer* that is selected by Algorithm 1. The *infoText* file is then updated immediately after each download is completed. The *infoText* file is updated using Algorithm 2. The *stdOutput* is the standard output for the download commands. $TransferRate_{new}$ is the TransferRate of the current download from the *bestServer*.

### 6.3 Comparison with Traditional Method

One of the traditional methods is just using the best transfer rate from the previous session. The algorithm using this method can be implemented by using the best transfer rate with the same algorithms (Algorithm 1, 2) instead of using $P_i$ and $p_i$. This method seems to be simple, but there is no difference in the algorithms. In addtion, with this method we cannot correct the historical information, as when using $h$ in our approach. The results of the traditional method are shown in Sect. 7.3.1.

**Algorithm 1** Select the best server with pheromone

```
1: open file infoText
2: create the set $\mathcal{S}$
3: close file infoText
4: for each serverName $s_i$ in $\mathcal{S}$ do
5:     execute iping to $s_i$
6:     $tp_i \leftarrow$ response time of $s_i$ iping
7:     add $tp_i$ to ipingList
8: end for
9: $tp_{min} \leftarrow \min(tp \in ipingList)$
10: for each serverName $s_i$ in $\mathcal{S}$ do
11:     if $tp_i \leq (tp_{min} + ipBT)$ then
12:         add $s_i$ to srvList
13:     end if
14: end for
15: open file infoText
16: if $n > 1$ then
17:     for each selectedServer $ss_i$ in srvList do
18:         seek the location of $ss_i$ information
19:         read $P_i$ from infoText
20:         add $P_i$ to PList
21:     end for
22: else
23:     for each selectedServer $ss_i$ in srvList do
24:         seek the location of $ss_i$ information
25:         read $TransferRate_i$
26:         add $TransferRate_i$ to trList
27:     end for
28:     for each selectedServer $ss_i$ in srvList do
29:         calculate $p_i(1)$ with $TransferRate_i$ and trList
30:         add $p_i(1)$ to PList as $P_i(1)$
31:     end for
32: end if
33: close file infoText
34: $P_{max} \leftarrow \max(P \in PList)$
35: for each selectedServer $ss_i$ in srvList do
36:     if $P_i$ is equal to $P_{max}$ then
37:         bestServer $\leftarrow ss_i$
38:         break;
39:     end if
40: end for
41: return bestServer
```

## 7. Simulation

We created a simulator to study the behavior of ACO-based data transfers. The simulator provided a controlled environment within which it was possible to study different types of scenarios.

**Algorithm 2** Update the information file

```
 1: open stream $stdOutput$
 2: execute download from $bestServer$
 3: $TransferRate_{new} \leftarrow TransferRate$
 4: close stream $stdOutput$
 5: open file $infoText$
 6: create the set $\mathcal{S}$
 7: for each $serverName\ s_i$ in $\mathcal{S}$ do
 8:     seek the location of $s_i$ information
 9:     read $TransferRate_i$
10:     add $TransferRate_i$ to $trList$
11: end for
12: close file $infoText$
13: calculate $p_{new}$ with $TransferRate_{new}$ and $trList$
14: open file $infoText$
15: seek the location of $bestServer$ information
16: update $TransferRate_{bestServer} \leftarrow TransferRate_{new}$
17: add $p_{bestServer} \leftarrow p_{new}$ to $infoText$
18: remove $p_{oldest}$ from $infoText$
19: close file $infoText$
```

## 7.1 Model

For the simulation we modeled two typical scenarios for downloading data in a distributed environment. The model assumed the data set spanned five different servers.

- *Phased Degradation*. In this case the performance of each server degrades over time as shown in Fig. 3. After the first file has been transferred the first server's performance degrades. The other servers' performance also degrades as they complete transfers. After seven transfers the performance improves for all of the servers, so they return to their optimal performance status after 12 files have been transferred. This situation is fairly common in distributed environments when clients start working in lock-step among the servers. Such a situation may appear when a group of clients start to use one system until its performance becomes unacceptable and they search for a new server for their downloads.
- *Random Degradation*. In this case the performance of each server degrades randomly over time as shown in Fig. 5. The performance of the first server degrades after 10 transfers and then improves and degrades again after 17 transfers. The second server degrades after seven transfers, returns to optimal performance after 13 transfers and then degrades after 20 transfers. The other servers follow similar patterns. This situation models a more random access pattern where there is no coupling among the performance of the servers.

## 7.2 Procedure

The simulation first required the preparation of input data for the ACO-based data transfer application. The models were used to generate several information files (Fig. 2). The

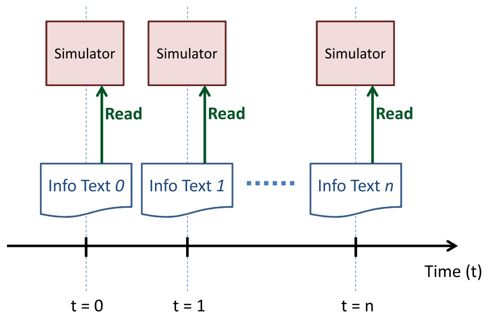

**Fig. 2** Simulator uses several information files.

server conditions for the two scenarios were defined in advance in each information file. The simulator runs by reading each information file. These information files contain rows of numbers according to the following schema (the example information file is already shown in the Table 1):

- server name
- 'ping' time
- download transfer rate
- upload transfer rate
- $p_i(k)$

Each row corresponded to the download information for a given file for a given server. For these simulations the download information was based on the iRODS *iget* download application. The simulation program read in a data set and ranked the servers according to the 'ping'-like information. The 'ping' information was based on the results of the *iping* command for iRODS. This determined the initial selections for which agents would use the hosts.

Each agent in the simulation program then used the best host on the list and started to read the simulation data (which included simulated download rates for the servers). The pheromone was then computed with Eq. (1) using the information from all of the available hosts that had completed their first download. Each agent ready to perform a download selected the available host with the best pheromone and updated the pheromone value after the download using Eq. (2). This procedure continued until the simulated data was exhausted.

## 7.3 Results

The results of the simulation are shown in Fig. 3 for the *Phased Degradation* model and in Fig. 5 for the *Random Degradation* model. Both models are using four pheromone histories ($h = 4$).

### 7.3.1 Phased Degradation

Figure 3 shows the first simulation results corresponding to the *Phased Degradation* model. The upper figure shows the transfer rate for each server without the ACO-based download. The *selected* curve corresponds to the ACO-based download. The lower figure shows the pheromone value for

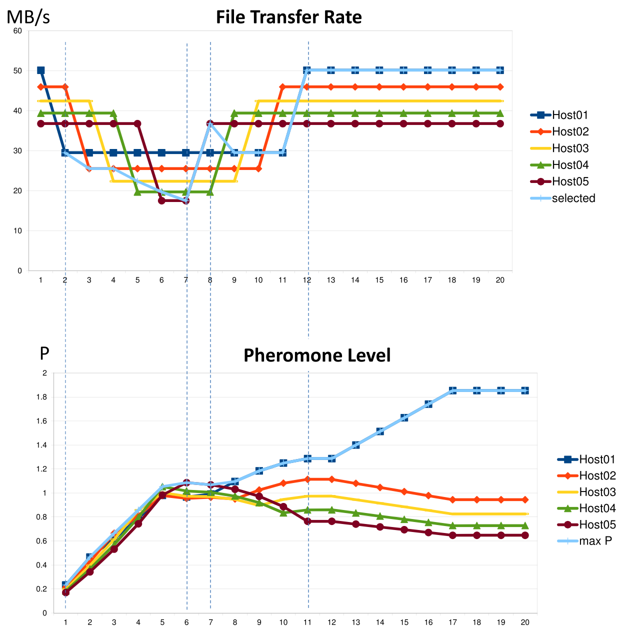

**Fig. 3** Transfer rate and pheromone for the phased degradation (the x-axis corresponds to the downloaded file number).

each host with the *max P* curve corresponding to the ACO-based download case. In this case of rapid degradation of the servers the ACO-based approach performed well. The pheromone is based on history information and there is always a delay between the response of the ACO-based download and the performance of the server.

In the initial stages the best server rapidly becomes the worst server resulting in the ACO-based approach tracking the degradation of the servers. However, as the servers improve in performance the ACO-based approach gradually improves. Clearly, this situation is a troublesome case, but realistic, situation and it is encouraging that the trough in the performance is steeper than that for each server, indicating that the algorithm is doing well in a bad situation. The lower graph shows consistently that the value of *max P* corresponding to the best path consists of those hosts with the best pheromone value at that time.

We also simulated this situation with the same data in the traditional method described in Sect. 6.3. Figure 4 shows the results from using slightly lower-performance hosts compared with our approach.

### 7.3.2 Random Degradation

Figure 5 shows the simulation results for the *random degradation* model. This case clearly shows that the results from the ACO-based approach shown in the *selected* curve outperform those based on any individual server. The visible dips at the beginning and end of the transfer period are an artifact of the pheromone having to rely on historical information. The pheromone for the best hosts shown in the *max P* curve in the lower figure consistently corresponds to the best host at that time.

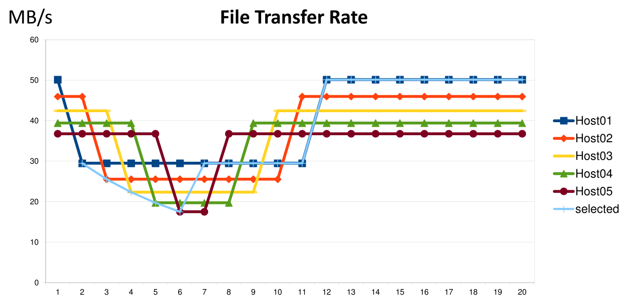

**Fig. 4** Transfer rate in the traditional way.

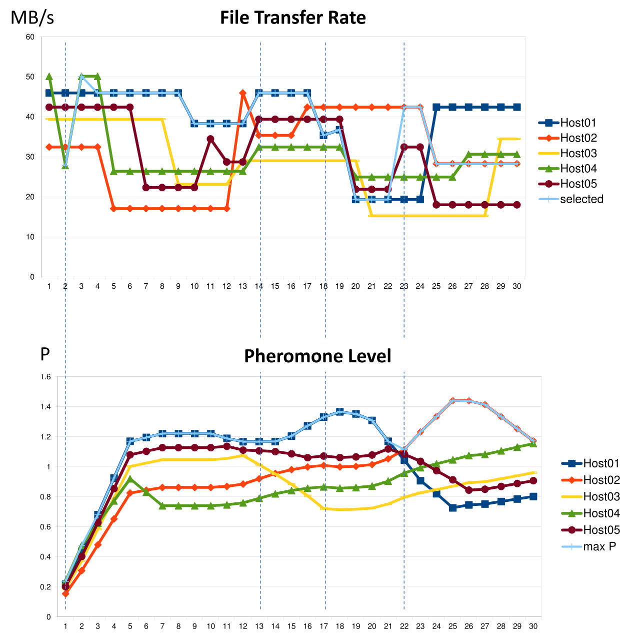

**Fig. 5** Transfer rate and pheromone in the random degradation model (the x-axis corresponds to the downloaded file number).

## 8. Test Implementation

We took the ACO-based application and used it for a real test-setup consisting of three distributed servers: one in the UK, one in the USA and one in Japan. We used the *iget* application of iRODS to download files from each system. The implementation required the development of scripts to provide the functions needed to implement the ACO-based downloads. These consisted of:

- iping (new i-command for iRODS). This was needed to perform the light-weight queries of the servers r to determine the server rankings.
- iping.py (script to drive the iping command). This script wrapped the iRODS icommand with iget.py.
- iget.py (script using the original iget command). This script implemented the ACO agent (as the ACO-based download algorithms) and called the iRODS *iget* command.

### 8.1 iping/iping.py

Currently iRODS does not have a command like *ping* that can be used to check the server availability. We created the *iping* application that calls the iRODS server and gets the echo outputs from the server.

The *iping.py* can specify the iRODS host with the option "-H" and iping the server. That is because the *iping* command can execute only on the server that is specified in the client's iRODS configuration file (.irodsEnv). All i-commands should follow the information in the .irodsEnv file so we avoid including the option specifying a server in the *iping* command, instead, the *iping.py* script takes charge of the options. The *iping* application also includes "*ping_to_all()*" function that can execute *iping* to all servers specified in a configuration file. This function is useful for checking all server availability just before executing *iget* commands. After executing the *iping* command invoked by the *ping_to_all()* function, the *iping.py* updates the ranking of the servers.

### 8.2 iget.py

The scripts for downloading (*iget.py*) a file execute the following steps:

1) execute *iping* for all of the servers
2) read the configuration file
3) select the best server
4) execute *iget* for a file
5) get the current *iget* transfer rate
6) calculate $p_i(k)$
7) update the transfer rate and $p_i(k)$

The steps except for 4) executing *iget* are our ACO agent tasks. The best server is selected in exactly the same manner as in the simulation. First, the servers are ranked according to their ping responses, and then the server with the best $P$ is chosen.

### 8.3 Results

The test-setup was highly distributed and consisted of three iRODS servers: one located at Queen Mary University of London (QMUL), UK, one at Louisiana State University (LSU), USA and one at KEK in Japan.

The tests were performed at KEK which is regarded as the local host and so the performance would be much better within unloaded servers. To address this we artificially adjusted the ranking results from the *iping* to give KEK the lowest ranking. The results are shown in Fig. 6. In this example, the same pheromone values are given as for the initial pheromone and the pheromone history ($h$) is 4. The first file was downloaded from LSU and the next from QMUL. In both cases the pheromone value was low (with the initial value set to one third of the corresponding pheromone for each server). The third file was downloaded from KEK resulting in a much higher pheromone value. Subsequent agents quickly settled on this host for downloading the data reinforcing the pheromone value for the KEK server. This demonstrated that the ACO-based approach was able to quickly find the optimum performance in a real environment.

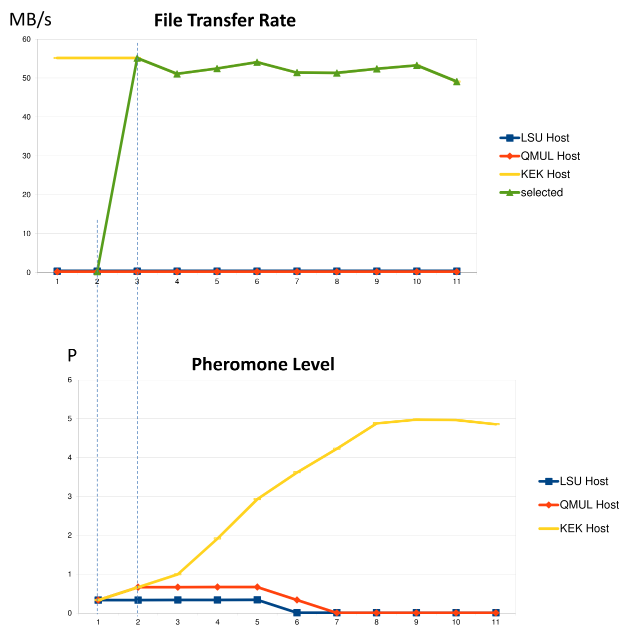

**Fig. 6** Transfer rate and pheromone in the actual case (the x-axis corresponds to the download file number).

We ran this demonstration with one client and three servers. This approach can scale easily since each client independently checks the servers. Therefore, we can use this approach in real environments (as mentioned in the introduction).

## 9. Summary and Future Work

In this paper we have described an approach inspired by swarm intelligence. We created a simulator for our ACO-based approach and obtained results showing that our approach works well. This approach can provide a fault tolerant and efficient means of transferring data in a dynamic environment. Also, we implemented an iping command with several scripts and demonstrated them in the iRODS file system. The demonstration showed that our ACO-based approach can quickly find the optimum performance in a real environment. In addition the self-optimizing nature of the ACO indicates the system is tolerant to servers going offline. In such cases the pheromone for that server will diminish from the next iteration encouraging the agents to select an active server.

For future work we want to apply this approach to different kinds of Data Grids. We also envisage a need to 'publish' the transfer information so that other clients in similar locations will be able to take advantage of the recent history to more quickly select an optimal set of servers to download from. We are also considering how to include constraint information to allow some clients to have preferential downloads over others.

## Acknowledgment

The authors would like to thank Dr. Francesca Di Lodovico for the iRODS setup between KEK and QMUL. Also, the support of Dr. Shantenu Jha and his team for use of the LSU server resource is gratefully acknowledged.

### References

[1] B. Gerardo and W. Jing, "Swarm intelligence in cellular robotic systems," Proc. NATO Advanced Workshop on Robots and Biological Systems, Tuscany, Italy, June 1989.
[2] Y. Itow, T. Kajita, K. Kaneyuki, M. Shiozawa, Y. Totsuka, Y. Hayato, T. Ishida, T. Ishii, T. Kobayashi, T. Maruyama, K. Nakamura, Y. Obayashi, Y. Oyama, M. Sakuda, M. Yoshida, S. Aoki, T. Hara, A. Suzuki, A. Ichikawa, T. Nakaya, K. Nishikawa, T. Hasegawa, K. Ishihara, A. Suzuki, and A. Konaka, "The JHF-Kamioka neutrino project," KEK Report, vol.4, p.29, 2001.
[3] "T2K-ND280 collaboration," Online, http://www.nd280.org/
[4] "iRODS – the integrated rule-oriented data system," Online, http://www.irods.org
[5] A. Rajasekar, M. Wan, R. Moore, and W. Schroeder, "A prototype rule-based distributed data management system," Proc. HPDC workshop on Next Generation Distributed Data Management, Paris, France, May 2006.
[6] "Data Intensive Cyber environments (DICE) Center at the University of North Carolina at Chapel Hill," Online, http://dice.unc.edu/
[7] C. Blum, "Ant colony optimization: Introduction and recent trends," Physics of Life Reviews, vol.2, pp.353–373, Oct. 2005.
[8] C. Jiang, C. Wang, X. Liu, and Y. Zhao, "A survey of job scheduling in grids," Lect. Notes Comput. Sci., vol.4505/2007, pp.419–427, 2007.
[9] G. Subashini and M. Bhuvaneswari, "Non dominated particle swarm optimization For scheduling independent tasks On heterogeneous distributed environments," Int. J. Advance. Soft Comput. Appl., vol.3, no.1, March 2011.
[10] A. Abraham, H. Liu, W. Zhang, and T. Chang, "Scheduling jobs on computational grids using fuzzy particle swarm algorithm," pp.500–507, Springer-Verlag Berlin Heidelberg, 2006.
[11] H. Izakian, B.T. Ladani, K. Zamanifar, and A. Abraham, "A novel particle swarm optimization approach for grid Job scheduling," Information Systems, Technology and Management, Communications in Computer and Information Science, vol.31, Part 5, pp.100–109, 2009.
[12] A. Abraham, S. Das, and S. Roy, "Swarm intelligence algorithms for data clustering," In Soft computing for knowledge discovery and data mining, vol.Part IV, pp.279–313, 2007.
[13] A.N. Sinha, N. Das, and G. Sahoo, "Ant colony based hybrid optimization for data clustering," Kybernetes, vol.36, no.2, pp.175–191, 2007.
[14] R. Peterson and E.G. Sirer, "Antfarm: efficient content distribution with managed swarms," NSDI '09: USENIX Symposium on Networked Systems Design and Implementation, pp.107–122, 2009.
[15] Y. Yang, Y. Zhao, and F. Hou, "Ant colony optimization algorithm based P2P system replica optimal location strategy," Service Operations and Logistics, and Informatics, pp.494–497, Oct. 2008.
[16] "Akamai technologies, globally distributed content delivery," http://www.akamai.com/dl/technical_publications/GloballyDistributedContentDelivery.pdf

**Yutaka Kawai** has been a researcher in Computing Research Center at High Energy Accelerator Research Organization (KEK) since 2009. His research interest is the interoperability for Grid/Cloud computing. He is currently a Ph.D. student at the Graduate University for Advanced Studies (SOKENDAI). He is a member of IPSJ and IEEE.

**Adil Hasan** is a research fellow at the University of Liverpool and a honorary research fellow at Kings College London. He has considerable experience in data management and distribution in both scientific and non-scientific data. Recently he was part of the technical coordination team for the SHAMAN EU-funded digital preservation project.

**Go Iwai** is currently Assistant Professor in the Computing Research Center of Applied Research Laboratory at High Energy Accelerator Research Organization (KEK). His research focus is developing the universal interface to different middleware components in Grids and Clouds. He is a member of Physical Society of Japan.

**Takashi Sasaki** is Professor of High Energy Accelerator Research Orgernization (KEK) and the Graduate University for Advanced Studies (SOKENDAI) since 2007.

**Yoshiyuki Watase** joined High Energy Physics Laboratory (KEK), Tsukuba, Japan in 1974 to study elementary particle physics by high energy particle accelerators. His current interests are in the field of distributed computing to cope with a huge amount of experimental data in the world wide collaboration. He is a member of Physical Society of Japan, IPSJ, and IEEE CS.