A PEER-TO-PEER SEARCH IN DATA GRIDS BASED ON ANT COLONY OPTIMIZATION

Uroš Jovanovič
XLAB Research
Teslova 30, SI-1000 Ljubljana, Slovenia

Boštjan Slivnik
University of Ljubljana, Faculty of Computer and Information Science
Tržaška 25, SI-1000 Ljubljana, Slovenia

Keywords: Distributed search, P2P, data grids, ant colony optimization.

Abstract: A method for (1) an efficient discovery of data in large distributed raw datasets and (2) collection of thus procured data is considered. It is a pure peer-to-peer method without any centralized control and is therefore primarily intended for a large-scale, dynamic (data)grid environments. It provides a simple but highly efficient mechanism for keeping the load it causes under control and proves especially usefull if data discovery and collection is to be performed simultaneoulsy with dataset generation. The method supports a user-specified extraction of structured metadata from raw datasets, and automatically performs aggregation of extracted metadata. It is based on the principle of ant colony optimization (ACO). The paper is focused on effective data aggregation and includes the detailed description of the modifications of the basic ACO algorithm that are needed for effective aggregation of the extracted data. Using a simulator, the method was vigorously tested on the wide set of different network topologies for different rates of data extraction and aggregation. Results of the most significant tests are included.

1 INTRODUCTION

Nowadays, vast datasets too large to be stored on a single computer are being generated and used all the time. In the scientific environment, they are often produced during experiments in a number of different fields, ranging from physics to genetics. In business, companies are storing all kinds of data from which customer behaviour can be analysed and predicted. In computing and telecommunications, logs are generated day and night.

Some datasets are generated by a single source but must be distributed immediately because of their size. For example, data grids are being developed to manage all data produced by particle accelerators. Other datasets like computer logs, for example, are generated by multiple sources but are most often analysed together.

Data grids represent a promising way to handle large and distributed datasets (Chervenak et al., 1999; Berman et al., 2003). To enhance the overall system performance, data replication is used to minimize the transfer of different chunks of a distributed dataset to the computer running the application (Čibej et al., 2005).

However, if the amount of data needed by the application is small compared to the entire dataset, the data should be extracted at the remote servers, collected together, and finally transfered back to the application. Thus, an application must be capable of sending custom requests to the dataset servers. In a distributed environment based on an unstructured network with no fixed topology and no metadata prepared in advance, this problem is hard to solve efficiently.

Grid infrastructure like Globus or gLite (Globus, 2006; gLite, 2006) is used to handle all low level details of sending the requests and obtaining the results, and to provide read access to raw data in order to extract metadata. To avoid flooding the network with requests and thus keeping the load limited the implemented grid service uses a method of ant colony optimization for extracting and collecting data from chunks of a distributed dataset (Dorigo and Stützle, 2004).

It has been demonstrated that ACO can be an effective method for clustering (Handl et al., 2006), but so far only static datasets have been considered. In this paper the ACO algorithm, adapted and extended in order to handle extraction and collection from dynamic raw datasets, is described.

Jovanovič U. and Slivnik B. (2006).
A PEER-TO-PEER SEARCH IN DATA GRIDS BASED ON ANT COLONY OPTIMIZATION.
In *Proceedings of the First International Conference on Software and Data Technologies*, pages 363-366
DOI: 10.5220/0001319703630366

2 USER'S PERSPECTIVE

Following the approach introduced in (Jovanovič et al., 2006), a grid service must enable a user to do the following two tasks:

1. *Starting a new search:* As a user is given full (read) access to the raw dataset, a user should be able to specify a program that searches the local chunk of a distributed dataset and extract the relevant data.
2. *Checking the list of performed searches:* A user must be able to check if any other user has already performed a search equal or similar to the one he or she is about to start.

As the user's program extracts data from the local chunk only, it is the responsibility of the grid service to transfer the user's program to each dataset server, to run the program, and to collect the extracted data from all servers.

The first reason for keeping a list of performed searches is to keep the overall system load low (a) by not repeating the same search all over again and (b) by using results from a similar (possibly more general) search instead of starting a new one. The second reason is a fact that the list of performed searches acts as a virtual blackboard about what data or relations among data in the dataset seem interesting to other users. This might be especially valuable in a collaborative scientific research. It follows that the results of the searches performed in the past should be stored in one form or the other somewhere in a data grid.

Hence, the typical method for performing a search is best described by an (informal) Algorithm 1. Note (1) that starting a new search in line 6 does not block the execution as the new search is performed on remote servers; (2) that the search and its results will eventually appear on the list of performed searches (even if it yields no results) and thus the loop in lines 7–13 does terminate.

Algorithm 1 Performing a search.

```
 1: check the list of performed searches
 2: if the (similar) search has been found then
 3:     return its results and stop
 4: end if
 5: provide the extraction program
 6: start a new search
 7: loop
 8:     check the list of performed searches
 9:     if the search has been found then
10:         return its results and stop
11:     end if
12:     sleep for a specified amount of time
13: end loop
```

This method allows different implementations of a distributed index of performed searches and different implementations of searching. At the time being, the research is focused on the efficient peer-to-peer implementation of a single search (line 6). The task of maintaining a list of performed searches in a distributed index and especially a method for comparing descriptions of different searches are left for the future work.

3 ANT COLONY OPTIMIZATION FOR SEARCHING RAW DATASETS

Ant colony optimization is a biologically-inspired optimization method (Dorigo and Stützle, 2004). The basic idea is to use a large number of simple agents called ants: each ant performs a relatively simple task but combined together they are able to produce sophisticated and effective optimization. Further improvements of ACO are based usually on a combination of the ACO algorithm with other local optimization techniques (Dorigo and Stützle, 2004). Ant based clustering and sorting have already been studied in the past (Deneubourg et al., 1991; Handl et al., 2006).

There are two main differences between ant based clustering and peer-to-peer searching as described in this paper. First, a node in the system can possess a pile of data, not just one datum. Second, other ant based clustering algorithms are suited for mesh or similar regular planar topologies, and thus the distance function is modified in order to be efficient for an arbitrary topology of a distributed environment.

Extraction Ants. Using user-specified programs, extraction ants extract data from local chunks of raw datasets. Each extraction program is uniquely identified. Extraction ants mark their paths with pheromones associated with the id of the program they are carrying. In order for extraction ants to discover as much data as possible they avoid the trails and prefer the clean paths.

Aggregation Ants. The data extracted by extraction ants is collected into piles by aggregation ants. Aggregation ant is based on Algorithm 2. According to the pick-up function $1/(1+x/n)^k$, *small* piles of data are picked up with a very high probability while *large* piles are most unlikely to be picked up. The distinction between small and large pile is predefined and based on the characteristics of the environment, such as the average connection bandwidth, and expected amount of data. Note that the distinction between a small and large piles can be simply regulated by changing n (a measure for the size of a pile) and k

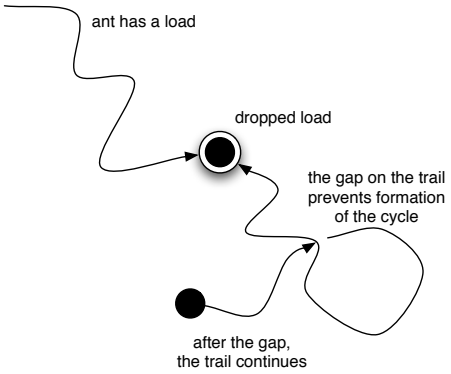

Figure 1: Result of the acyclic path marking. (The trail on the right is a "backward" trail produced when the ants is moving away from the location where a data has been dropped.).

(a measure of strictness) in the pick-up function. Second, the dropping probability function is simplified: the ant decides to drop the load whenever data of the same type is present on the node. Furthermore, in order to limit the network load, the number of hops that a loaded ant can take is limited.

The probability of finding some data is based on the number of pheromone trails that lead to the pile containing the data. To increase this probability, after dropping the load, an ant marks the first few connections with pheromones directed to the location of the dropped data. The outcome of such pheromone marking is a tree of pheromone trails where every branch of the tree is directed towards the root of the tree that contains a pile of data. The result of such process in shown in Figure 1.

There are positive as well as negative consequences of these so called pheromone fields. On the positive side, there is a high probability that a smaller pile is correctly merged with larger pile. On the negative side, once an ant enters the pheromone field, it gets trapped. The ant that picks up a pile gets trapped in the pile's own pheromone trails.

Pile Ants. In order to avoid the pheromone fields because of their negative effects, *pile* ants are used. They are simple random walkers that are produced by piles and ignore any pheromone information. At every new location, a pile ant checks if another pile of data of the same type as the one in the pile that emitted it, exists. The smaller pile is picked up and merged with the bigger pile.

One-Time Aggregating Ants. One-time aggregation ants are special type of aggregation ants. Whenever an extracting ant extract some data, a one-time aggregation ant is also created at the same location. This ant picks up the new extracted data and tries to drop it at an appropriate place. When the data is dropped, the ant *dies*, i.e., is removed from the system. One-time aggregation ants annul the time needed to discover new extracted data and increase the probability that they are dropped into near piles.

Algorithm 2 A pheromone-based aggregation ant.

```
1:  loop
2:    while the ant is empty do
3:      while the node is empty do
4:        select a random direction
5:        make a step in the selected direction
6:      end while
7:      if load is selected then
8:        pick up the load; h ← 0
9:      end if
10:   end while
11:   repeat
12:     select a direction
        using the existing pheromones
13:     mark the selected direction
14:     make a step in the selected direction
15:     h ← h + 1
16:     if h = h_max ∨ the node load = the ant load
        then
17:       drop the load; h ← 0
18:     end if
19:   until the ant is empty
20:   while h < h_max do
21:     select a random direction
22:     mark the selected direction
        using the cycle-prevention method
23:     make a step in the selected direction
24:     h ← h + 1
25:   end while
26: end loop
```

Query Ants. The role of the query ants is to find data collected in piles by following the pheromone paths created by aggregation ants.

Another method for data discovery excludes the need for query ants. It is based on maintaining of distributed indexing service. When the pile is *static* and *big enough*, it registers its location into this distributed indexing service. Here, static property of a pile means that the pile has not changed its position in some predefined amount of time. The second property, being big enough, means that the pile contains at minimum some predefined amount of aggregated metadata.

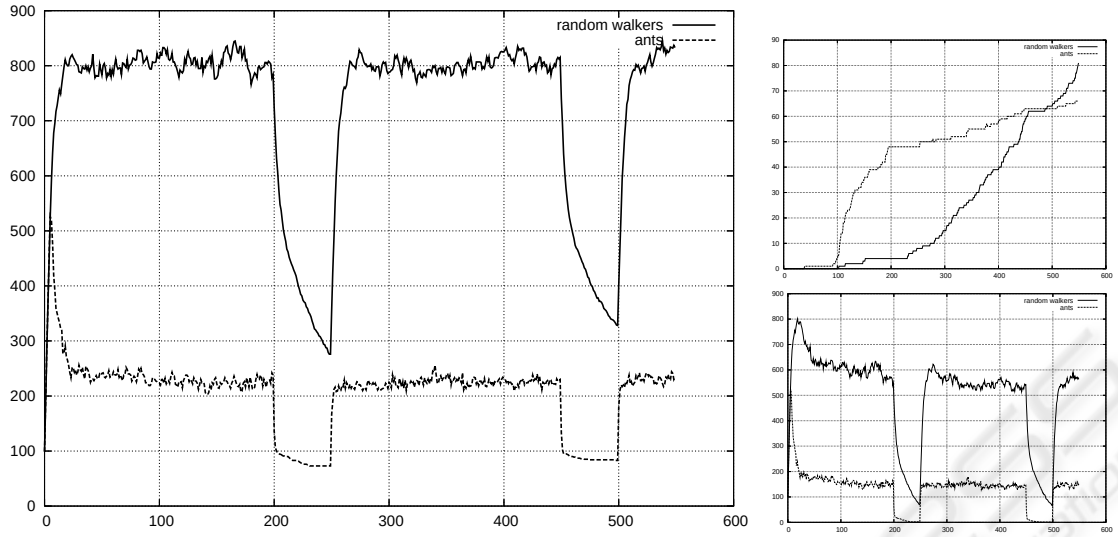

Figure 2: The results of testing with high rate of extraction (100 random extractions per iteration; 300 extraction ants): the total number of piles aggregated by random walkers and aggregation ants (left), number of large (static) and small piles aggregated by random walkers and ants (top right and top bottom, respectively).

4 EXPERIMENTAL RESULTS

The experimental results were obtained by simulating the extraction and collection of data using a two-layered network consisting of 2550 nodes. There are 50 fully-connected nodes, each of them being a node of a subnetwork consisting of another set of 51 fully-connected nodes. We have also performed the tests on different layouts and of different scale, but different topologies yield very similar results to those presented here.

For the chosen topology and the number of nodes, we tested our algorithm against the random walk heuristic. Note that during the extraction the aggregation is also being performed. We have also tested the scenario when the extraction stops and only aggregation is being performed. These cycles are referred to as pure aggregation cycles.

Figure 2 shows the results obtained when data is being extracted on 100 random locations in each iteration, and 300 aggregation ants were always present in the system besides one-time aggregation ants.

5 CONCLUSION

In this paper we have presented modifications of basic ACO algorithms that are well suited to the limitations of the distributed environments. The described modification of ACO enables very simple yet precise runtime control over the load caused by extraction and aggregation simply by regulating the number of ants.

REFERENCES

Berman, F., Fox, G. C., and Hey, A. J. G. (2003). *Grid Computing: Making the Global Infrastructure a Reality*. John Wiley and Sons, Ltd., Chichester, England.

Chervenak, A., Foster, I., Kesselman, C., Salisbury, C., and Tuecke, S. (1999). The data grid: Towards an architecture for the distributed management and analysis of large scientific datasets. *Journal of Network and Computer Applications*, 23:187–200.

Čibej, U., Slivnik, B., and Robič, B. (2005). The complexity of static data replication in data grids. *Parallel Computing*, 31:900–912.

Deneubourg, J. L., Goss, S., Franks, N., Sendova-Franks, A., Detrain, C., and Chretien, L. (1991). The dynamics of collective sorting: Robot-line ants and ant-like robots. In *From Animals to Animats 1 (Proceeding of the First International Conference on Simulation of Adaptive Behavior)*, pages 356–365. The MIT Press, Boston, USA.

Dorigo, M. and Stützle, T. (2004). *Ant Colony Optimization*. The MIT Press, Boston, USA.

gLite (2006). EGEE > gLite — Lightweight Middleware for Grid Computing. Retrieved April 28th, 2006, from http://glite.web.cern.ch/glite/.

Globus (2006). The globus alliance. Retrieved April 28th, 2006, from http://www.globus.org/.

Handl, J., Knowles, J., and Dorigo, M. (2006). Ant-based clustering and topographic mapping. *Artificial Life*, 12(1):35–62.

Jovanovič, U., Močnik, J., Novak, M., Pipan, G., and Slivnik, B. (2006). Using ant colony optimization for collaborative (re)search in data grids. In *Proceedings of the Cracow Grid Workshop '05, Cracow, Poland*, pages 205–207.