Searching Raw Datasets in Data Grids Using Ant Colony Optimization

UROŠ JOVANOVIČ
XLAB Research
Teslova 30, SI-1000 Ljubljana
SLOVENIA

BOŠTJAN SLIVNIK
Faculty of Comp. and Inf. Science
University of Ljubljana
Tržaška 25, SI-1000 Ljubljana
SLOVENIA

Abstract: - A pure peer-to-peer method for (1) an efficient discovery of data in large distributed raw datasets and (2) collection of thus procured data is considered. It is based on ant colony optimization (ACO) and supports a user-specified extraction of structured metadata from raw datasets, and automatically performs aggregation of extracted metadata. 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.

Key-Words: - Distributed search, P2P, data grids, ant colony optimization.

1 Introduction

Nowadays, vast datasets too large to be stored on a single computer are being generated and used all the time. 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 these large and distributed datasets as well as a platform for running data intensive applications [2, 1, 3]. 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 first, then collected together, and finally transfered back to the application. To achieve this, 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.

A few important issues should be noted at this point. First, the infrastructure should handle all the low level details of sending the requests and obtaining the results. Second, when searching for the relevant data, the application should not be limited to a set of metadata prepared in advance. Instead, at least full read access to the raw data must be given to the application. Third, as multiple applications can run simultaneously, the load caused by servicing the requests of one application should be limited. Keeping load under control is especially important if the dataset is generated at the same time as it is being used and thus the search is an ongoing process instead of an one-time event. Thus a decentralized, possibly a peer-to-peer solution is preferred. Fourth, if possible, the data extracted by one application should be available to all other applications.

The first two issues can be solved by using existing grid infrastructure like Globus or gLite [8, 7]. The third and the forth issue have already been addressed by proposing a grid service within Globus Toolkit 4 [10]. 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 [6]. Ant colony optimization has been used as it has been demonstrated to be an effective method for clustering (and sorting) of data within a distributed environment [9].

However, only static datasets has been considered so far when ant colony optimization has been used

either for clustering or for extracting and collecting data. In this paper, an ant colony optimization based method for extracting and collecting data from dynamic as well as static raw datasets stored within a data grid is desribed and analysed.

The paper is organized as follows. In Section 2, a user's perspective on a grid service based on this method is described as the motivation. Sections 3 contains a description of ant colony optimization in relation to data clustering. The details of all modifications for data aggregation in a dynamic environment are described in Section 4 together with the algorithm used by extraction ants. Before listing the conclusions and plans of the future work in Section 6, results obtained at testing the method using extensive simulations are given in Section 5.

2 User's Perspective

Following the approach introduced in [10], 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, if 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 [5]. 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 [6].

Ant based clustering and sorting was first introduced in [4]. Further analysis and extensions of the basic algorithm used for clustering is described in [9]. These algorithms were applied to problem domains that are first mapped into two dimensional cartesian space which is stored in computers main memory. Moving from this domain to real physically

distributed space presents new constraints to the environment in which the algorithm is used.

There are three main differences between ant based clustering and peer-to-peer searching as described in this paper. First, unlike in [4] and [9], a node in the system can possess a pile of data, not just one datum. The size of a pile is bounded only by the available storage space on the node.

Second, the distance function proposed in original algorithm and also in changed ATTA algorithm [6] uses window of 3×3 or 5×5 neighbor nodes. In 2D cartesian space, there are only 9 or 25 nodes to be considered. Unlike in original algorithms, distributed environments can be based not only on mesh topology but on arbitrary topologies and close neighborhoods can include hundreds of nodes (this is especially true in the grid environment). Communicating with all neighbors and gathering information about their status can present massive load on the network and must therefore be avoided. In order to limit the network load and prohibit the distribution of large amounts of data, new probability function is proposed (see below).

And third, instead of using only one type of ants as in [6], there are three different types of ants in our scheme: extraction, aggregation and query ants.

4 Ant Specialization

4.1 Extraction Ants

Using user-specified programs, extraction ants extracting data from local chunks of raw datasets stored in the system, i.e., in a data grid. Each user-specified extraction program is identified by its unique identification code, or id for short. Extraction ants are marking their paths with pheromones associated with the id of the program they are carrying. Hence, extraction ants carrying different extraction programs use different pheromones.

In order for extraction ants to discover as much data as possible we have modified the behavior of the basic ant algorithm. Unlike the normal ants that prefer following the pheromone trails, the extraction ants avoid the trails and prefer the clean paths.

4.2 Aggregation Ants

The data extracted by extraction ants is collected into piles by aggregation ants.

Due to the environment restrictions described in Section 3, the probability function for dropping and picking-up data has to be changed from the one reported in [6].

Let us first describe the pick-up function $1/(1 + x/n)^k$. According to the chosen function, *small* piles of data are picked up with a very high probability. On the other hand, *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 (a measure of strictness) in the pick-up function.

Second, the dropping probability function is simplified. Since the inspection of ant's neighbor nodes is expensive (regarding net traffic), the only information available to the ant is the content of its current node. The ant therefore decides to drop the load whenever data of the same type is present on the node.

Another distinction from the original algorithm is a limited number of hops that a loaded ant can make. This limitation is also used in order to avoid the network load.

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, we added another change in the behavior of ants. When the load is dropped, an ant takes off in search for another data to be picked up. During its first few steps, the ant marks the path with pheromones directed to the location of the dropped data as shown on Figure 1.

Large piles of data are surrounded by many pheromone trails that are directed to pile's location. These trails act as a pile's gravitational field. Whenever an ant comes close to them, there is a high probability that it will be drawn towards the center of directed pheromone trails. This is desired when the ant is carrying some data, but it is undesired when the ant drops the load and tries to find new data. In order to avoid this pheromone fields and escape its *pulling* effect, the ant that has just dropped the load makes first few steps regardless of the pheromone information, that is, the first few moves are completely random based.

Random movement during the marking of pile's location can produce cycles. These cycles can lead to

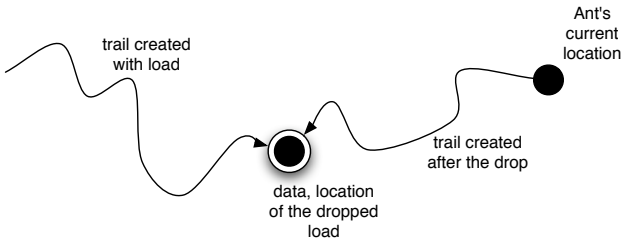

Figure 1: Ant makes a trail leading to the location of dropped metadata pile. (The trail on the right is a "backward" trail produced when the ants is moving away from the location where a data has been dropped.)

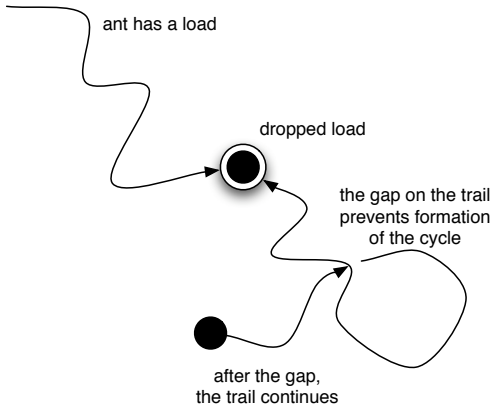

Figure 2: Result of the acyclic path marking. (The trail on the right is a "backward" trail produced when the ants is moving away from the location where a data has been dropped.)

inefficient behavior of ants that follow the pheromone trails – by following the pheromone trail, they move in circles. It is therefore desired to avoid the creation of cycles.

In order to achieve acyclic pheromone trails, we need to modify the process of marking used paths with pheromones. To do so, we randomly select ant's new destination first. Then, we check if there exists a pheromone trail directed from this new destination. If there is no such trail, we mark a path from new destination to current location with a pheromone. Otherwise, this path is not marked. The result of such process in shown in Figure 2.

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, which constrains a pile of data.

Hence, an aggregation ant is based on Algorithm 2.

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
```

4.2.1 Dynamic Environment and One-Time Aggregating Ants

Most of the analysis of ACO algorithms described in literature is based on the idea that the data is scattered around the environment first, and that the number of data does not increase over time. But such assumptions are not realistic enough for a real distributed environment where we wish to deploy our ACO algorithm. To address the specifics data grids (as described above), new type of ant is being introduced, *one-time aggregation ant*.

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. If no appropriate pile of already organized data is found in predefined maximal allowed number of

hops, the data is dropped and the ant *dies*, i.e., is removed from the system. On the other hand, if a pile is found, the data is dropped on the pile and the ant dies.

One-time aggregation ants annul the time needed to discover new extracted data and increase the probability that they are dropped into near piles. Furthermore, because they are removed from the system as soon as the data is dropped, they do not present significant increase of network load.

4.3 Query 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.

5 Experimental results

In order to eliminate the possibility of error, two independent implementations were developed, from which one is written in Java and the other is written in pure functional language Haskell. All results presented here were obtained by using a two-layered network which describes GRID topology consisting of connected clusters. Regarding the definition of a layered topology, we have 50 fully-connected nodes, each of them being a node of a subnetwork consisting of another set of 51 fully-connected nodes. Therefore, our system is composed of 2550 nodes. We have also performed the tests on different layouts and of different scale (classical 4- and 8-mesh topologies, for instance). Different topologies yield very similar results to those presented here.

The new ACO-based method was tested against random walkers which are mostly used in the state-of-the-art techniques used in unstructured peer-to-peer systems. In the first scenario, the extraction is very sparse, no more than 10 extractions are made per iteration (only 0.4% of nodes). In the second scenario, the extraction is denser, 100 nodes participate in the process of extraction in each iteration.

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.

The most important property of the generated solutions is the number of piles of data in the system - the network load depends on this property, since all the piles have to register to the indexing system and therefore, the lower the number the lesser the load of the network. Also, when the search for data is executed, the indexing system informs the user (query ants) about the locations of piles and all must be accessed. Again, smaller number of piles presents small load to the network. So it is desirable to have as organized (low number of piles) system as possible. Figure 3 shows the results obtained by simulating both scenarios; temporary decrease of number of piles corresponds to the pure aggregation phase.

The graph shows that the ants are able to organize data in lesser number of piles. Even if we give random walkers a time to organize (pure aggregation cycle), the solution that is produced is worse than the solution of ants without the cycle. We must note though that by increasing the size of the pure aggregation cycle, the random walkers do catch up with the ants without this cycle. When both types are given the same amount of time for either aggregation and extraction or pure aggregation, the ants always outperform the random walkers.

6 Conclusion

Distributed environments such as grids and peer-to-peer systems present physical constraints that are not present if the optimization is done on a single computer. Communication between nodes can represent network load that makes the system useless. In this paper we have presented modifications of basic ACO algorithms that are well suited to the limitations of the distributed environments. We have compared the performance of our algorithm with the algorithm based on random walkers and shown that the ACO algorithm always outperforms the one based on random walkers.

The described modification of ACO enables very simple yet precise run-time control over the load caused by extraction and aggregation simply by regulating the number of ants.

In the future, the method described herein is going to be implemented as a fundamental part of a grid service instead of the method described in [10].

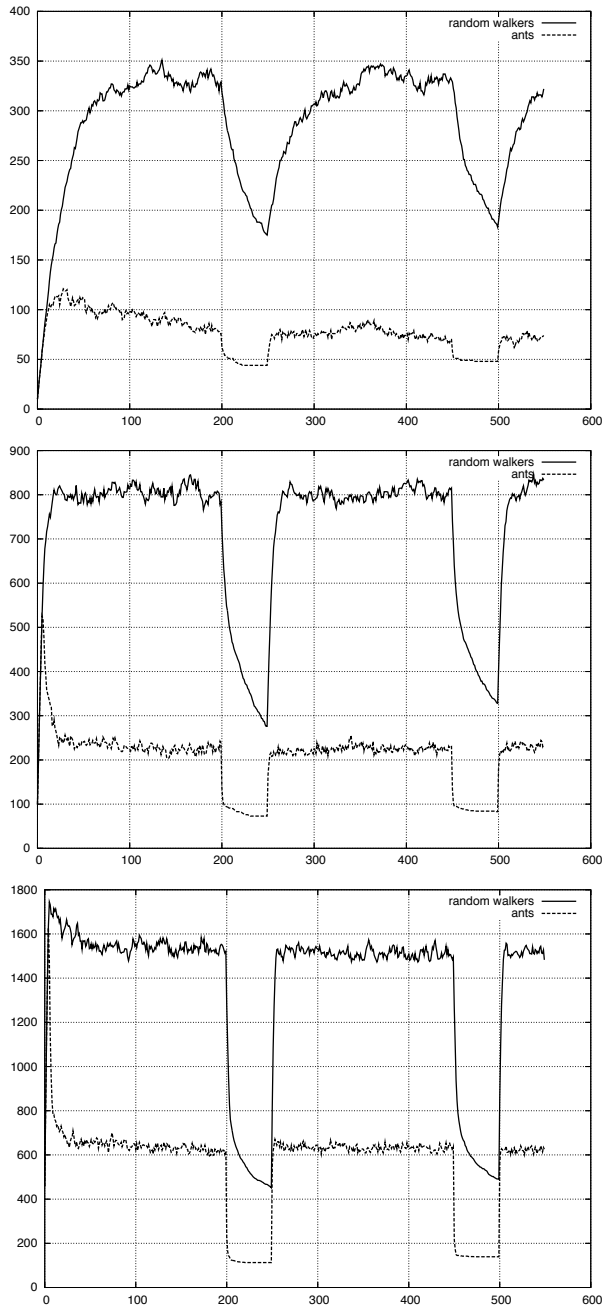

Figure 3: The results of testing with low rate of extraction (top — 10 random extractions per iteration and 300 extraction ants), medium rate of extraction (below — 100 random extractions per iteration and 300 extraction ants), and high rate of extraction (below — 500 random extractions per iteration and 300 extraction ants): the total number of piles aggregated by random walkers and aggregation ants.

References

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

[2] A. Chervenak, I. Foster, C. Kesselman, C. Salisbury, and S. Tuecke. 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, 1999.

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

[4] J. L. Deneubourg, S. Goss, N. Franks, A. Sendova-Franks, C. Detrain, and L. Chretien. 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, 1991.

[5] M. Dorigo, V. Maniezzo, and A. Colorni. Positive feedback as a search strategy. Technical report, Politecnico di Milano, Italy, 1991.

[6] M. Dorigo and T. Stützle. *Ant Colony Optimization*. The MIT Press, Boston, USA, 2004.

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

[8] Globus. The globus alliance. Retrieved April 28th, 2006, from http://www.globus.org/, 2006.

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

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