Placement of Nodes in an Adaptive Distributed Multimedia Server

Balázs Goldschmidt[1], Tibor Szkaliczki[2,*], and László Böszörményi[3]

[1] Budapest University of Technology and Economics
balage@inf.bme.hu

[2] Computer and Automation Research Institute
of the Hungarian Academy of Sciences
sztibor@sztaki.hu

[3] University Klagenfurt, Department of Information Technology
laszlo@itec.uni-klu.ac.at

Abstract. Multimedia services typically need not only huge resources but also a fairly stable level of Quality of Services. This requires server architectures that enable continuous adaptation. The Adaptive Distributed Multimedia Server (ADMS) of the University Klagenfurt is able to dynamically add and remove nodes to the actual configuration, thus realizing the *offensive adaptation* approach.

This paper focuses on the optimal placement of nodes for hosting certain ADMS components (the so-called *data collectors*, collecting and streaming stripe units of a video) in the network. We propose four different algorithms for *host recommendation* and compare the results gained by running their implementations on different test networks. The *greedy* algorithm seems to be a clear looser. Among the three other algorithms (*particle swarm*, *linear programming* and *incremental*) there is no single winner of the comparison, they can be applied in a smart combination.

1 Introduction

Even highly sophisticated multimedia servers with a distributed architecture, such as the Darwin server of Apple [1] or the Helix architecture of RealNetworks Inc. [2] are *static* in the sense that actual configurations of the the distributed server must be defined manually. The Adaptive Distributed Multimedia Server (ADMS) of the University Klagenfurt [3] is able to dynamically add and remove nodes to the actual configuration. Thus, ADMS realizes the *offensive adaptation* approach [4]. In case of shortage of resources, instead of reducing the quality of the audio-visual streams by usual, defensive, stream-level adaptation, it tries to migrate and/or replicate functionality (i.e. code) and/or audio-visual data on demand.

[*] Partial support of the EC Centre of Excellence programme (No. ICA1-CT-2000-70025) and the Hungarian Scientific Research Fund (Grant No. OTKA 42559) is gratefully acknowledged.

It is crucial for the performance of the system, to find optimal placement for the server nodes. The *host recommender* component of the ADMS system determines the host computers in the network where the server components (also called applications) should be loaded. The optimal location depends on the actual load and capacity of the available nodes and links, and on the actual set of client requests.

The distributed multimedia server architecture [3] has different components that can be located on different hosts of the network. *Data managers* or *servers* store and retrieve the media data. *Data collectors* or *proxies* collect the data from the servers and stream them to the clients.

This paper deals with the configuration recommendation algorithms. Unfortunately, the related problems are NP-complete. However, there are many approximation algorithms that can result in nearly optimal solutions within short time. We examine, what kind of mathematical approaches can be applied to the adaptive host recommendation.

The formal and detailed description of the model and the algorithms can be found in a technical report[1] [5].

2 Related Work

Finding the optimal deployment of proxies in a network is a well known problem in the literature. Most of the works, however, deal only with (1) static configurations, (2) web-proxies, and (3) caching-problems [6–8]. Static configuration means, that proxies are deployed once, and their placement can not change later, or with high cost only. On the other hand, web proxies have to serve the clients with relatively short documents and images that have to arrive unmodified. Online multimedia data delivery serves huge data-streams, where some modifications are still acceptable, having the emphasis on the timing constraints. Finally, we are currently not interested in caching problems, because they are orthogonal to the offensive adaptation problem. This issue was discussed elsewhere [4]. Therefore, we cannot use the former results directly.

We looked at the mathematical background and found the facility location problem(FLP), which is an intensively studied problem in operations research. The problem is to select some facility candidates and to assign each client to one of the selected facilities while minimizing the cost of the solution. A detailed description of the problem can be found in [9]. Despite the similarity, some significant differences prohibit the direct application of the approximation algorithms for the FLP to the host recommendation. First, while the cost of a facility is usually a constant value in case of the FLP, in the current problem it depends on the maximum bandwidth required by the clients assigned to the proxy. Furthermore, the limited bandwidth of the subnets must be taken into account as well.

In [9] it is shown that FLP is NP-hard for general graphs. Many constant approximation algorithms have been published for the facility location problem

[1] http://143.205.180.128/Publications/pubfiles/pdffiles/2004-0005-BGAT.pdf

with polynomial running time that are usually combined with each other. The linear programming techniques play key role in many algorithms with constant approximation ratio [10, 11]. The best approximation ratio was achieved combining the primal-dual method with the greedy augmentation technique [12].

As a different approach, evolutionary algorithms (EA) also provide an efficient way of finding good solutions for NP-hard problems. In [13] an EA is proposed to solve the P-median problem, which problem is in close relation to FLP. A kind of evolutionary algorithms, particle swarm optimisation proved to be effective in a wide range of combinatorial optimisation problems too [14].

3 The Problem Model

In order to implement and compare different algorithms that solve the problem, we have defined the following model and metrics.

From the actual point of view the Adaptive Distributed Multimedia Server consists of Data Managers and Data Collectors. Multimedia data (videos) are stored on the Data Managers. The videos are sliced, and the resulting stripe units are distributed to the Data Managers. When needed, the video is recollected and streamed to the clients by the Data Collectors. This technique helps both network and node resource load balancing.

According to [15], Data Managers that contain stripe units of the same video should be kept as close to each other as possible, practically on the same subnet, because that configuration gives the best performance. Based on this result our model considers such a group of Data Managers a single *server*.

The media is collected from the Data Managers by Data Collectors (proxies). They can be loaded on any node that hosts a Vagabond2 Harbour [16]. The nodes that may play the role of a Data Collector are considered *candidates*.

The *client* is the third kind of component in the model. The clients connect to Data Collectors and get the desired media via streaming from them. The clients define their requests as lists that contain QoS requirements in decreasing order of preference. We assume that the demand list has a last, default element, that represents the situation when the client's request is rejected. In the current model we also assume that all clients want to see the same video at the same time.

In our current model only the proxies can be deployed dynamically. The locations of the servers and that of the clients are not to be modified.

The network model is basically a graph where the nodes are called *areas*. An area is either a subnet (including backbone links) or a router that connects subnets. We handle them similarly because from our point of view they have the same attributes and provide the same functionality. The edges of the graph are the connections between the routers and the subnets they are part of.

We assume that we know the subnet location of the clients and servers, the candidate nodes, and the attributes of the network, the latter presented as the attributes of each area: bandwidth, delay jitter, etc. The route between any two nodes is described as a list of neighbouring areas.

The solutions of a problem are described as possible *configurations* that provide the following information:

- for each client, the index of the QoS demand that has been chosen to be satisfied, and the candidate that hosts the proxy for the client
- for each candidate, the server, where the video should be collected from

In order to compare different solutions, we have defined a cost function that gets the initial specifications and a possible configuration as input parameters. Using this information the function calculates the network resource needs (total allocation and over-allocation), the number of rejected clients (those whose chosen demand is the last, default one), the sum of the chosen demand indexes of the clients (*linear badness*), and the so called *exponential badness*, that is defined as $\sum_{c \in C} 2^{i_c}$ where C is the set of clients, and i_c is the index of the chosen demand for client c. This last metric is useful if we want to prefer 'fair' to 'elitist' configurations.

Given two costs, that cost is less, that has (in decreasing order of preference):

- less over-allocation (we can't afford over-allocation),
- less rejection (we want to increase the number of accepted clients),
- less exponential badness (we prefer fairness),
- less total allocation (we want to minimize the network load).

Using this cost metric we were able to compare several algorithms.

4 Solution Algorithms

In this chapter we provide a short introduction to the algorithms we have implemented and tested. More details can be found in [5].

4.1 Greedy

The greedy algorithm we use here is almost identical to that published in [17]. The difference is that the cost function is changed to that described in the previous chapter, and that the clients' demand list is also taken into account.

4.2 Particle Swarm

The particle swarm algorithm is based on the algorithm of Kennedy and Eberhardt [14]. The original algorithm uses a set of particles, each of them representing a possible configuration. Every particle is connected to several other particles thus forming a topology.

The particles are initialized with random values. Each particle knows its last configuration ($\underline{x}(t)$), its best past configuration ($\underline{b}_p$), and the configuration of that neighbour (including itself) that has the least cost ($\underline{b}_n$). In each turn, if it has less cost than any of its neighbours, it counts a new configuration by creating a linear combination of $\underline{b}_p$ and $\underline{b}_n$, using probabilistic weights, then it adopts this new configuration. The whole process runs until some condition is met. The combination is defined as:

$$\underline{x}(t+1) = \underline{x}(t) + \varphi_1(\underline{b}_p - \underline{x}(t-1)) + \varphi_2(\underline{b}_n - \underline{x}(t-1))$$

where φ_i is a random number from $[0, 1)$.

This original algorithm, however, can solve those problems only, where the dimensions of the configurations represent binary or real values. This is the consequence of the linear combination technique. In our case the configurations' dimensions represent unordered elements of sets. Therefore, instead of the linear combination that can not be applied to them, we use the following. For each dimension we first take the value with a certain probability from either $\underline{x}(t)$, or $\underline{b}_n$ (crossover). Then, with another probability, we change it to a random value (e) from its value set (mutation). Finally, we assign the value to the given dimension.

Crossover and mutation are also applied by genetic algorithms[18]. The difference between genetic algorithms and our algorithm is that in our case the connections of the partners (neighbours) are static. Particles do crossovers only with their neighbours, and the crossovers transfer information from the better configuration to the worse only, thus the best results are always preserved.

The algorithm runs until every particle has the same cost. But this condition combined with the mutation leads from the initial fast evolution to a final fluctuation. We apply the simulated annealing of the mutation rate in order to avoid this phenomenon [19]. The control variable of the annealing is the number of particles that have their configuration changed. The less particle changed, the less the mutation rate is.

4.3 Linear Programming Rounding

LP Model. We chose an algorithm based on linear programming rounding for the solution of the configuration problem. For simplicity, we minimize only the number of refused clients, the exponential goodness and sum of the reserved bandwidth for each subnet. Weights are assigned to the different optimization criteria to express their priority.

Inequations express three types of constraints as follows. The reserved bandwidth of a subnet is less than or equal than the available. A proxy-server connection needs at least as much bandwidth as the maximum among the accepted requests served through it. Each client is either assigned to a client-proxy-server route or rejected.

We introduce variables $X_{i,j,k}$ to indicate whether the request of client i is served by server k through proxy j. Their possible values are 0 and 1. Since the time complexity to find the exact solution for an integer linear programming problem is large, we consider the LP-relaxation of the problem, where the possible values of the variables can be any real number.

Rounding. First we solve the linear program and obtain an optimal solution. If $X_{i,j,k} = 1$ then let the request of client i be served by server k through proxy j. Unfortunately, the possible fractional values of X-type variables do not represent legal solutions. We round the solution in a greedy manner. We take each X variables with fractional value one after the other. If the client is still not served, we try to select the current client-proxy-server route denoted by variable $X_{i,j,k}$, and check the load conditions in the network. The client i is served by server k through proxy j in the solution if and only if these conditions are fulfilled after the selection of the route.

4.4 Incremental Algorithm

In order to find solutions quickly, we implemented a very simple but efficient algorithm. It operates on the so-called *FLP graph* which is a bipartite graph, where one set of nodes denotes the clients, the *facility* nodes represent the proxy-server routes and the edges between them correspond to the client-proxy-server routes that are able to satisfy client requests. The main steps of the algorithm are as follows.

After generating the FLP graph, the facilities are sorted in decreasing order of the bandwidth of the represented proxy-server routes. We take the facilities one after the other. The algorithm selects facility f_i if it can serve new clients or there is at least one client c_j already assigned to a facility denoted by f_0 where the QoS parameters of edge (c_j, f_i) is better than that of edge (c_j, f_0). If facility f_i is selected, we take the clients adjacent to it one after the other and client c_j is assigned to it if it fulfills the above condition and its request can be satisfied through facility f_i without overloading the network. A facility is deselected if no clients are assigned to it.

5 Results

We implemented the algorithms and tested them on simulated network environments. Each test network consists of 50 subnets. Six test series were generated; each of them consisted of ten cases. The number of servers is always ten, while the number of clients and proxies varies in different series; there are 5, 10, 15, 20, 25, 30 clients and 10, 20, 30, 40, 40, 40 proxies in the different series. We examined the cost of the solutions and the running time as a function of the number of clients. Figure 1 shows the costs (linear and exponential badnesses, numbers of rejected clients) and the runtime. The figures compare the results of the algorithms described above, namely linear programming rounding, swarm algorithm, greedy algorithm and incremental algorithm. The linear programming does not produce legal solutions, without rounding, but can be used as a lower bound for the cost measures.

According to the figure, the swarm algorithm produces the best results, and the linear programming rounding achieves results with slightly higher cost and the greedy algorithm fails to find nearly optimal algorithms for a high number of clients.

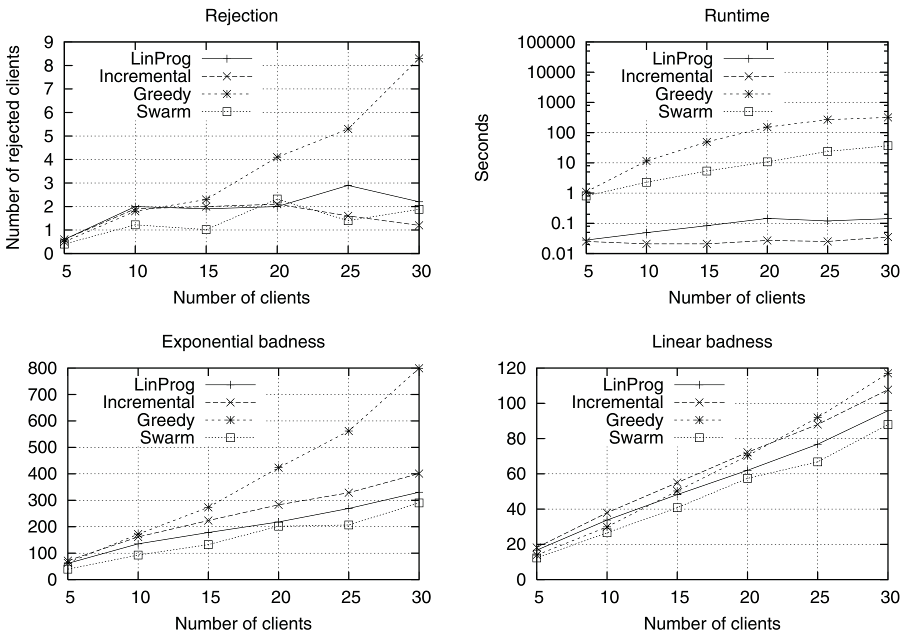

Fig. 1. The results of the measurements for linear programming rounding, incremental, greedy, and swarm algorithms.

On the other side, the running time of the incremental algorithm is clearly the best, the running time of linear programming with rounding is the second, while the swarm and the greedy algorithms run substantially slower.

6 Conclusions and Further Work

We introduced a number of algorithms for host recommendation in an Adaptive Distributed Multimedia Server. We did not find an algorithm that is the best in every aspect. A good idea is producing an initial solution quickly using the fast deterministic algorithms. Later, if time allows, a more sophisticated solution (admitting more clients) might be rendered, using the stochastic swarm algorithm.

In the future, we intend to improve the model of the network by incorporating node information also. Thus not only the predicted future values of the network parameters and client requests might be taken into account at the recommendation, but the performance of the host nodes also. Later the implementations of the algorithms will be integrated into the Adaptive Distributed Multimedia Server and tested also in a real network environment.

References

1. Apple Computer, Inc. QuickTime Streaming Server: Darwin Streaming Server: Administrator's Guide. (2002) http://developer.apple.com/darwin/projects/streaming/
2. Helix Community: Helix Universal Server Administration Guide. (2002) https://www.helixcommunity.org/2002/intro/platform
3. Tusch, R.: Towards an adaptive distributed multimedia streaming server architecture based on service-oriented components. In Böszörményi, L., Schojer, P., eds.: Modular Programming Languages, JMLC 2003. LNCS 2789, Springer (2003) 78–87
4. Tusch, R., Böszörményi, L., Goldschmidt, B., Hellwagner, H., Schojer, P.: Offensive and Defensive Adaptation in Distributed Multimedia Systems. Computer Science and Information Systems (ComSIS) **1** (2004) 49–77
5. Goldschmidt, B., Szkaliczki, T., Böszörményi, L.: Placement of Nodes in an Adaptive Distributed Multimedia Server. Technical Report TR/ITEC/04/2.06, Institute of Information Technology, Klagenfurt University, Klagenfurt, Austria (2004)
6. Steen, M., Homburg, P., Tannenbaum, A.S.: Globe: A wide-area distributed system. IEEE Concurrency (1999)
7. Li, B., Golin, M., Italiano, G., Deng, X., Sohraby, K.: On the optimal placement of web proxies in the internet. In: Proceedings of the Conference on Computer Communications (IEEE Infocom). (1999)
8. Qiu, L., Padmanabhan, V.N., Voelker, G.M.: On the placement of web server replicas. In: INFOCOM. (2001) 1587–1596
9. Cornuejols, G., Nemhauser, G.L., Wolsey, L.A.: The uncapacitated facility location problem. In Mirchandani, P., Francis, R., eds.: Discrete Location Theory. John Wiley and Sons, New York (1990) 119–171
10. Shmoys, D., Tardos, E., Aardal, K.: Approximation algorithms for facility location problems. In: Proceedings of the 29th ACM Symposium on Theory of Computing. (1997) 265–274
11. Charikar, M., Guha, S.: Improved combinatorial algorithms for the facility location and k-median problems. In: IEEE Symposium on Foundations of Computer Science. (1999) 378–388
12. Mahdian, M., Ye, Y., Zhang, J.: Improved approximation algorithms for metric facility location problems. In: Proceedings of 5th International Workshop on Approximation Algorithms for Combinatorial Optimization. (2002)
13. Dvorett, J.: Compatibility-based genetic algorithm: A new approach to the p-median problem. In: Informs Fall 1999 Meeting. (1999)
14. Kennedy, J., Eberhardt, R.C.: Swarm Intelligence. Morgan Kaufmann (2001)
15. Goldschmidt, B., Tusch, R., Böszörményi, L.: A corba-based middleware for an adaptive streaming server. Parallel and Distributed Computing Practices, Special issue on Dapsys 2002 (2003)
16. Goldschmidt, B., Tusch, R., Böszörményi, L.: A mobile agent-based infrastructure for an adaptive multimedia server. In: 4th DAPSYS (Austrian-Hungarian Workshop on Distributed and Parallel Systems), Kluwer Academic Publishers (2002) 141–148
17. Goldschmidt, B., László, Z.: A proxy placement algorithm for the adaptive multimedia server. In: 9th International Euro-Par Conference. (2003) 1199–1206
18. Davis, L., ed.: Handbook of Genetic Algorithms. Van Nostrand Reinhold (1991)
19. Kirkpatrick, S., Gelatt, C.D., Vecchi, M.P.: Optimization by simulated annealing. Science, Number 4598, 13 May 1983 **220, 4598** (1983) 671–680