# INCREMENTAL PLACEMENT OF NODES IN A LARGE-SCALE ADAPTIVE DISTRIBUTED MULTIMEDIA SERVER

Tibor Szkaliczki*
*Computer and Automation Research Institute of the Hungarian Academy of Sciences*
sztibor@sztaki.hu

Laszlo Boszormenyi
*University Klagenfurt, Department of Information Technology*
laszlo@itec.uni-klu.ac.at

**Abstract** An incremental algorithm is proposed to dynamically place the proxies of the Adaptive Distributed Multimedia Server (ADMS) developed at the University Klagenfurt. In order to enhance the performance of the server, the proposed algorithm examines the suitable network nodes for hosting proxies. The main benefit of the algorithm is the capability to process large problems within strict time constraints. The short running time of the algorithm enables the distributed server to adapt quickly to the changing network parameters and client demands.

**Keywords:** incremental algorithm, multimedia server, video streams, host recommendation, data collector

## 1. Introduction

It is a usual task to select nodes for hosting dynamic server applications in a network. The Adaptive Distributed Multimedia Server (ADMS) of the University Klagenfurt [Tusch, 2003] is able to add and remove its components to different nodes of the network. This novel feature of the multimedia server enables the dynamic placement of the server components according to the current requests and the QoS parameters of the network. The running time of the host recommendation algorithm

*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.

becomes crucial in this case since the delivery of the data-streams can start only after the placement of the server nodes.

In this paper, we propose an incremental algorithm that is especially suitable for large-scale distributed video servers delivering stream-data to large number of clients. In this case the time-consuming algorithms aiming at the "perfect" solution are not applicable. The proposed algorithm takes the possible nodes one after the other, and it places a proxy at the examined node if it improves the solution. The simplicity of the proposed algorithm enables to find an initial solution as fast as possible and than the algorithm incrementally improves it complying with the time constraints in order to approximate the optimal placement.

## 2. Related Work

Finding the optimal deployment of proxies in a network is a well known problem in the literature [Steen et al., 1999], [Qiu et al., 2001]. However, we cannot use the former results directly because of the significant differences between the ADMS proxies and the well-studied web-proxies. The placement of the web-proxies cannot be changed later or with high cost only. Moreover, the multimedia server provides huge data-streams instead of documents and images. The caching problems of the multimedia servers are not in the scope of the present paper, they are discussed in [Tusch et al., 2004].

In an earlier paper we dealt with the configuration recommend,ation algorithms for the offensive adaptation [Goldschmidt et al., 2004]. We proposed four different algorithms (greedy, particle swarm, linear programming rounding and first ideas on an incremental algorithm) and compared the results gained by running their implementations on different test networks. The particle swarm algorithm, a kind of evolutionary algorithms produced the best result while the incremental algorithm found the solution in the shortest time. The current paper explores the incremental algorithm in detail. We present the results after the definition of the problem model.

Incremental algorithms are applied to many problems in the area of the combinatorial optimisation, see as an example [Ramalingam and Reps, 1996, Zanden, 1996]. Their main step is updating the solution of a problem after a unit change is made in the input. The incremental algorithms result in significant decrease of computation time in case of many problems.

## 3. The problem model

The task is to find suitable locations for proxies of the distributed multimedia server while maximising the clients' satisfaction and minimising the network load. The *clients* receive the same video in parallel. The videos are stored at the *server* nodes. The proxies get the desired video from the servers and forward the received packets to the clients without storing them. According to the present model, the proxies can reduce the bandwidth of the video and can send the same video to different clients with different bandwidths. The proxies can be located only on nodes prepared for hosting dynamic server components. The nodes that are able to host proxies are called *possible proxies*.

Technical report [Goldschmidt et al., 2004] contains the detailed description of the problem model. 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. The edges of the graph are the connections between the routers and the subnets they are part of. We assume that we know the Quality of Service attributes of each area: bandwidth, delay jitter, etc.

There are three kinds of components that can be found on the nodes, namely servers, possible proxies and clients. In the problem specification, each area may contain several components with different types, such as clients, possible proxies and servers. The clients define their demands as lists that contain QoS requirements (e. g. bitrate and delay jitter) in decreasing order of preference. In the current model we assume that all clients want to see the same video immediately.

The solution of a problem is described as a possible *configuration* that provides the following information for each client: the index of the QoS demand that has been chosen to be satisfied, the possible target node that hosts the proxy for the client, the server, where the video should be collected from.

The following *cost* functions are defined to measure the quality of the solutions: the network resource needs (total allocation), the number of rejected clients, the sum of the chosen demand indices 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 parameter for client $c$.

Table 1 gives a short summary of the results published in [Goldschmidt et al., 2004] in order to compare different host recommendation algorithms. The incremental algorithm can find a solution in the shortest time. The number of rejections is also very low, but the exponential

badness is higher by more than 30 percents than in case of the swarm algorithm.

| *algorithm* | *exp. badness* | *lin. badness* | *rejections* | *time (sec)* |
|---|---|---|---|---|
| Greedy | 800 | 117 | 8.3 | 296.9 |
| Particle swarm | 299 | 90 | 1.9 | 172.7 |
| Linprog rounding | 320 | 95 | 2.1 | 0.11 |
| Incremental | 400 | 109 | 1.2 | 0.03 |

*Table 1.* The results of the measurements for different algorithms solving networks with 50 nodes, 10 servers, 40 possible proxies and 30 clients

We enhance our model published earlier for the case when huge number of clients exist in the network. We want to serve as many clients as possible with the server, which can be much more than the number of the areas of the networks. Fortunately, the variety of the client demands is limited in practical cases regarding the case where all clients demand the same video. This enables to handle many clients without significant increase of the running time. Now, let the clients denote client groups with the same demand lists in a subnet. The problem specification can be simply modified by adding the size of the client groups to the description of the client demands. Thus, the number of client groups can be bounded by the constant multiple of the number of nodes where the constant is typically smaller than ten. The client groups may be divided into smaller groups at the output, because the clients may receive the same video in different quality even if they belong to the same client group.

## 4. Incremental algorithm

According to the results presented on Table 1, the incremental algorithm proved to be promising to find a solution for large-scale networks.

A special graph is applied to store and retrieve the client-proxy-server routes that are able to satisfy the client requests. We call it FLP graph, because the idea comes from the facility location problem, a kind of optimisation problem in the area of operations research. One set of the nodes denotes the clients, the other one represents the proxy-server pairs referred to facilities. The edges between them correspond to the client-proxy-server routes. An edge is put into the graph only if the route is able to satisfy the request of the client. Using this graph, we can easily retrieve each routes for a proxy or a client.

The original algorithm is modified in order to accelerate the generation of the initial solution. The algorithm generated first the FLP graph.

However, this can be time-consuming for large-scale networks. For this reason, the generation of the FLP graph is partitioned into smaller steps processing individual proxies which are inserted into the incremental algorithm.

```
1 for each proxy do
2    Calculate the QoS parameters of the routes from the proxy
3    for each server do
4       Add a facility node together with edges to the FLP graph
5       Decide on selecting the current facility
6       if the facility is selected then
7          for each client connected to the current facility do
8             Decide on assigning the current client to the facility
9 Unselect facilities that are not connected to any client
```

The parameters of the routes between the proxy and each other nodes can be calculated in Step 2 by running the shortest path algorithm. The QoS parameters of the client-proxy-server routes can be easily determined from the results calculated in Step 2. If the QoS parameters of the route satisfy any requirements of the client, an edge is added to the FLP graph between the current client and facility. The algorithm stores the parameters of the route at the edge together with the index of the first requirement of the client that can be satisfied by the represented client-proxy-server route.

The algorithm selects facility $f_i$ in Step 5 if there is at least one client $c_j$ among the nodes adjacent to the facility that it is still not assigned to any facility or if client $c_j$ is assigned to facility $f_0$ then the parameters of the edge between $c_j$ and $f_i$ are better than that between $c_j$ and $f_0$, and the badness of the first satisfiable demand on the edge between $c_j$ and $f_i$ is not greater than that of the satisfied demand on the edge between $c_j$ and $f_0$. These criteria are complemented with one more in Step 8: the demand of client $c_j$ can be satisfied through facility $f_i$ without overloading the network.

In order to accelerate the algorithm, only a subset of the facilities is processed instead of each possible proxy-server pair. We tested some different kinds of subsets in order to determine how the number of processed facilities can be substantially reduced without increasing the cost values. According to our experiences, a facility may be omitted from processing if there is another one where each of the QoS parameters are better. The results published in this paper are produced applying this acceleration method.

The algorithm is slightly modified in order to deal with the client groups. New procedures are needed to calculate the bandwidth requirement of the demands and to select a client group. In order to minimise

the exponential badness, the algorithm tries to serve the clients belonging to the same group with equal quality, that is, the difference between the index of the satisfied demands is at most one in a client group. Each client in a group is served by the same proxy and server in the present version. This restriction can be eliminated later in order to improve the quality of the solution.

## 5. Results

The implemented incremental algorithm is tested on simulated network environments. The first test network has 50 nodes, 40 possible proxies, 30 clients and 10 servers and the numbers of the components are increased proportionally in the further networks in order to find out the size of the largest problem that still can be solved, see Table **??**. The test were running on a 1.2 GHz processor with 384MB memory. The algorithm successfully solves problems not greater than 500 nodes in less than 10 seconds.

| *nodes* | *servers* | *proxies* | *clients* | *time* |
|---|---|---|---|---|
| 50 | 10 | 40 | 30 | 0.03 |
| 100 | 20 | 80 | 60 | 0.16 |
| 200 | 40 | 160 | 120 | 0.821 |
| 300 | 60 | 240 | 180 | 2.333 |
| 400 | 80 | 320 | 240 | 5.137 |
| 500 | 100 | 400 | 300 | 9.663 |

*Table 2.* The dimensions of the test networks and the running times

Let us analyse how the cost of the solution is decreasing as the algorithm examines newer and newer facilities or proxy-server pairs. Figures 1 show the costs (exponential badnesses, numbers of rejected clients) as a function of the time elapsed from the start of the algorithm. In the second case, the algorithm was stopped before finishing because it would be running too long.

As we can see, first the cost starts falling quickly and later only slight improvements are achieved. If the quick response is a crucial point, this fact makes worth realising intermediate solutions while the algorithm is running. Thus we can instantly start to deliver media to several clients and then gradually increase both the number of the served clients and the quality of the media stream. In this way, we can give recommendation for problems even if the running time for the optimal solution would be extremely long.

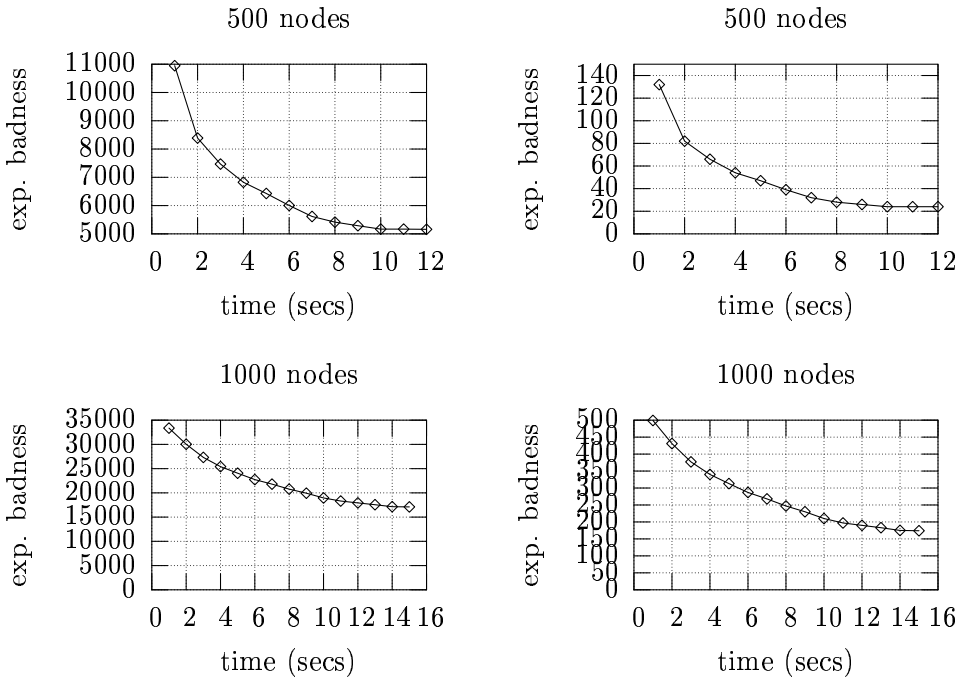


*Figure 1.* The badness and the number of rejections as a function of the elapsed time processing a network with 500 nodes, 100 servers, 400 possible proxies and 300 clients (above) and with 1000 nodes, 200 servers, 800 possible proxies and 600 clients (below)

We tested how the program is able to manage clients groups. The initial network consists of 50 nodes, 10 servers, 40 possible proxies and 300 clients. In this case, each client group contains only a single client. Further networks are created with 3000 and 30000 clients by increasing the size of the client groups to 10 and 100, respectively, while the size of the network and the number of server components do not change. There is no significant change in the running time as the size of the client groups increases, see Table 3.

At last, let us examine the quality of the solution. Using linear programming, we can find a lower bound for the cost. The algorithm accepts each of the requests in the network with 50 nodes. We find a lower bound of 7 for the number of rejections in the case of 100 nodes and the incremental algorithm generates a solution with 9 rejections. The exponential badnesses are 360 and 1152 in the two cases instead of the lower bound of 215 and 902, respectively.

| *nodes* | *servers* | *proxies* | *clients* | *exp. badn.* | *lin. badn.* | *reject.* | *time(sec)* |
|---|---|---|---|---|---|---|---|
| 50 | 10 | 40 | 300 | 5340 | 1141 | 29 | 0.27 |
| 50 | 10 | 40 | 3000 | 53408 | 11393 | 290 | 0.26 |
| 50 | 10 | 40 | 30000 | 1511544 | 160804 | 21964 | 0.3 |

*Table 3.* The results for client groups with increasing sizes

## 6. Conclusions and Further Work

We examined an incremental algorithm for the configuration recommendation in a large-scale adaptive distributed multimedia server with huge number of clients. The speed of the incremental algorithm, the continuous decrease of the solution cost and the introduction of client groups enabled us to solve large problems. Further development is needed to improve the quality of the solution in order to decrease the number of the rejections and to improve the quality of the delivered video.

## References

[Goldschmidt et al., 2004] Goldschmidt, B., Szkaliczki, T., and Böszörményi, L (2004). Placement of Nodes in an Adaptive Distributed Multimedia Server. Technical Report TR/ITEC/04/2.06, Institute of Information Technology, Klagenfurt University, Klagenfurt, Austria.

[Qiu et al., 2001] Qiu, L., V.N., Padmanabhan, and G.M., Voelker (2001). On the placement of web server replicas. In *INFOCOM*, pages 1587–1596.

[Ramalingam and Reps, 1996] Ramalingam, G. and Reps, Thomas W. (1996). An incremental algorithm for a generalization of the shortest-path problem. *J. Algorithms*, 21(2):267–305.

[Steen et al., 1999] Steen, M., Homburg, P., and Tannenbaum, A. S. (1999). Globe: A wide-area distributed system. *IEEE Concurrency*.

[Tusch, 2003] Tusch, R. (2003). Towards an adaptive distributed multimedia streaming server architecture based on service-oriented components. In Böszörményi, L. and Schojer, P., editors, *Modular Programming Languages, JMLC 2003*, LNCS 2789, pages 78–87. Springer.

[Tusch et al., 2004] Tusch, R., Böszörményi, L., Goldschmidt, B., Hellwagner, H., and Schojer, P. (2004). Offensive and Defensive Adaptation in Distributed Multimedia Systems. *Computer Science and Information Systems (ComSIS)*, 1(1):49–77.

[Zanden, 1996] Zanden, B. Vander (1996). An incremental algorithm for satisfying hierarchies of multiway dataflow constraints. *ACM Transactions on Programming Languages and Systems*, 18(1):30–72.