Client Behaviour Prediction in a Proactive Video Server

Péter Kárpáti, András Kocsor, László Böszörményi

University of Klagenfurt, Department of Information Technology
{kpeter,laszlo}@itec.uni-klu.ac.at
University of Szeged, Research Group on Artificial Intelligence
kocsor@inf.u-szeged.hu

Abstract. We present a possibility how to add proactive behaviour to Video-on-Demand systems. To do so we propose categorizing videos and using external information as well as observing the behaviour of our clients. We examined 23 predictor functions on artificial and real datasets using different similarity measures to compare them. Our model is quite simple; therefore some extensions are proposed at the end.

Keywords: Proactive, Video-on-Demand, ADMS, Client Behaviour Prediction, Ranking

1 Introduction

On-demand streaming can be regarded as a solved problem, at least inside a good local area networking environment, as often used by enterprises. The same techniques may, however, lead to unsatisfactory results if used in a general Internet setting. Therefore, it seems to be a good idea to create a video server that is able to make local copies of its code and the required parts of its videos at the place where these are needed. Thus, the clients have the impression of being served by a local enterprise server for the price of a few, more or less idle nodes which temporarily host the downloaded code and videos.

Such a video server implements an offensive adaptation strategy [1]. As videos are usually large in size, replication itself may take a considerable time. Therefore, the videos should be replicated in advance – in other words proactively. This was our basic motivation in creating a proactive and offensively adaptive video server.

Every proactive server needs to know something about the future behaviour of its users. It can obtain this information from outside the system or make predictions with the help of observations about the past. Our server uses a mixture of these two approaches. It can predict on its own which clients are interested in a given video topic and is able to use external hints about the popularity of videos. In this paper we examine different predictor functions (simple moving averages, autoregressive moving averages, a neural network approach) which help the server to choose which clients will be covered by proactive adaptation. We investigate the predictors in two different test scenarios: in an artificial one and in a real one. In order to compare the results of the predictors we have three different similarity measures at our disposal.

1.1 The Adaptive Distributed Multimedia Streaming Server (ADMS)

The client behaviour prediction is investigated in the context of our Adaptive Distributed Multimedia Streaming Server (ADMS) [2]. ADMS consists of four components: the Data Distributor (DD) distributes the videos, received from a Production Client, onto Data Managers (DM) which store stripe units of the videos. The Data Collector (DC) collects these units, assembles them and streams the requested video to a Retrieval Client. Finally the ADMS Controller (AC) organizes how the server works. When a component in the system is overloaded, the server is able to set up new components and also to migrate and replicate videos between DM groups, thus complying with the new requests.

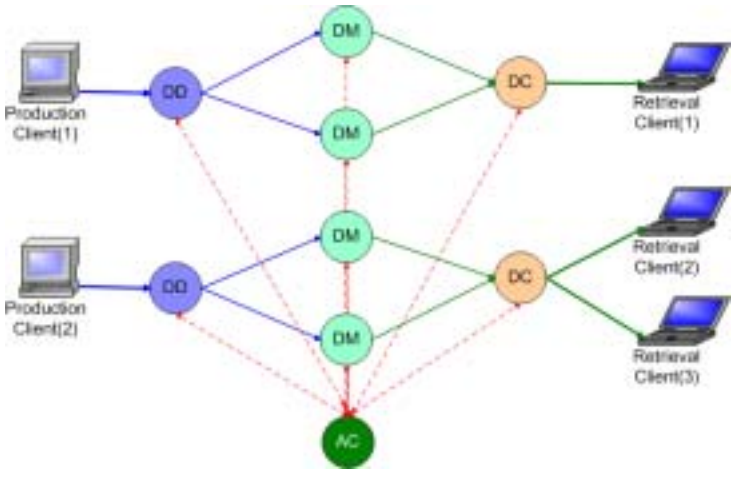


Fig 1. ADMS Architecture

To achieve this, the ADMS relies on an appropriate middleware called Vagabond2 [2] which makes it possible for the ADMS to load and evacuate its components to and from nodes belonging to the middleware (so-called Harbours). For this paper the following two components of Vagabond2 are of special importance: the

Resource Broker which provides information about the free capacities of different resources in the system and which can predict a future state of the resources with a given probability, and the Configuration Recommender (CR) which can suggest an optimal configuration of the server to fulfil a group of requests. A configuration consists of the placement of the components, the relationships <Video Instance – DMs>, <DMs – DCs> and <DCs – Clients> and defines which client gets which variation of a video and in which way. The CR can give recommendations for *light-weight actions* such as placing DC components [3] or for *heavy-weight actions* such as replicating or migrating video instances. (Under the term light-weight actions we consider those which can be accomplished quickly and do not need too many resources in contrast to the time-consuming and/or resource-intensive heavy-weight actions.) The CR provides an optimal spatial configuration at a certain point in time. In this paper we discuss how to add temporal optimization, based on predictions.

We can classify streaming servers as *static* or *self-organizing* servers [5]. The composition of a static streaming server can only be changed by a manual system reconfiguration while a self-organizing streaming server is capable of automatically distributing service code and/or content to another server machine. State-of-the-art streaming servers, like the Darwin Streaming Server [6] and the Helix Universal Server [7, 8], use static organization. Static organization can be good enough in an environment where the users and their behaviour are well known and stable. It can still have cost problems if a global optimum for the placement of the components and videos does not exist because of time-dependent changes in the behaviour of the clients. A well-made self-organizing server can always find an "almost best" configuration of its software and video components, according to the distribution of the current requests. To find such a configuration and to realize it are two different things. There are activities which can be accomplished quickly (e.g. to set up a DC in the case of ADMS) so the system can react to a current situation immediately, but these activities have their limits in efficiency. Other activities (such as the replication of a larger multimedia dataset from one end of the system to the other) are time consuming and therefore not adequate for the system to react quickly to new increasing demands. Thus self-organizing streaming servers must have more knowledge about the future load in the system in order to exploit their flexibility effectively and to maintain an almost optimal state. In other words, self-organizing streaming servers need to apply client behaviour predictions and/or to be able to use information from outside the system.

1.2 Videos and popularity hints

In our model the videos are divided into two groups: series and one-off programmes (or non-series). Series are broadcast regularly within a short period of time and the parts have something common. Examples are the news or "Friends". The clients can subscribe to series and define a periodic slot in which to watch the latest episode(s). It also means that proactive distribution of them is not a problem since the clients tell the server their future behaviour, so the videos can be placed in the system in an optimal fashion.

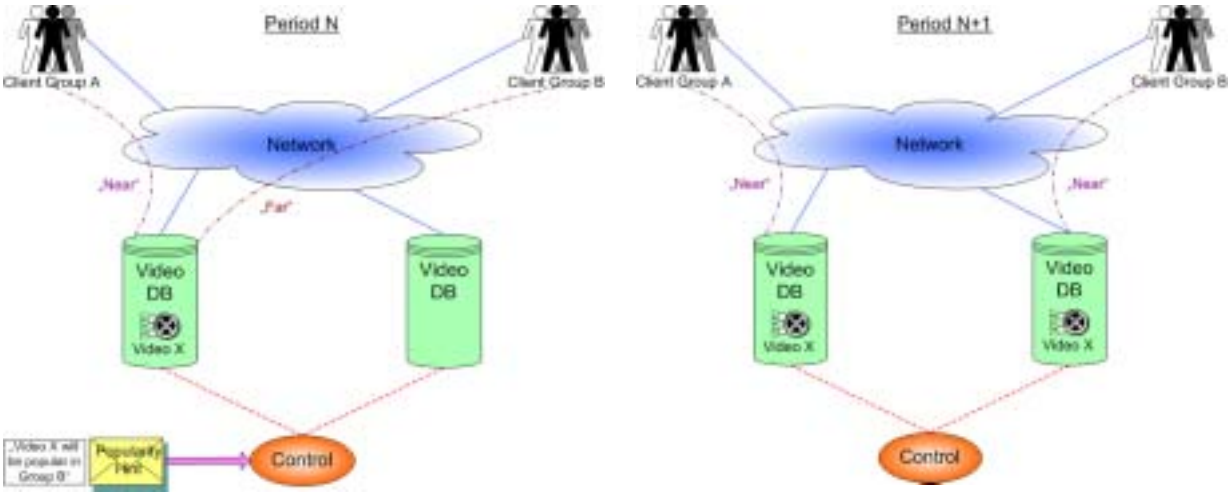


Fig 2. Proactive adaptation driven by a popularity hint

Examples for one-off programmes are movies, educational videos and so on. They usually have one dominant topic and they are what we are focusing on. In the model, the server wants to identify the clients who are most interested in the topic of a given video and to place this video near them if it is worth it (for example, depending on the business model, the server should not move a big video only for one user).

Popularity hints are injected from outside into the system and report useful information about the expected popularity of a given video. In our model a popularity hint about a video predicts what percentage of the clients who are interested in the topic of this video will be interested in the video itself. This could be estimated from the cost of advertising or from previous experiences (e.g. of another country). The control mechanism of the

server predicts the ranking of the users interested in the given topic and then chooses the best clients with the given percentage. It will try to distribute the video close to these clients. (By the prediction, the server uses its knowledge about the past requests of the clients.)

The activities of the server are divided into equal periods (measured e.g. in days, weeks or months). In the example above (Fig 2. left hand side) we are currently in the N-th period. The server has a control functionality (AC in ADMS) somewhere and two places (DM groups in ADMS) for videos. In the first video database there is Video X and it is close to Client Group A but far from Client Group B. (A client is "far" from a video server if a reasonably smooth streaming of the videos cannot be guaranteed.) The server gets a popularity hint which says that in the next period Video X will be popular in Client Group B. Let us assume that the server expects from client statistics that Client Group A will (still) like Video X as well, therefore it decides to replicate the video to the second video DB, closer to Client Group B for the beginning of period N+1 (Fig 2. right hand side).

1.3 Further organization of the article

In the next section we deal with related research, i.e. the proactive capability in video servers, prediction and recommendation in the Web. In the third section we briefly present the predictors and similarity measures used. In the next two sections, the two test scenarios and their results are outlined. At the end we conclude and show the directions in which this work will continue.

2 Related works

A related area to ours is Web prefetching. In some of these systems a threshold probability is selected and objects whose estimated probability of use exceeds that threshold before modification or eviction from the cache are fetched, in order to achieve balance between the benefits and costs. Prefetch-Nice [9], a non-interfering Web prefetching system, suggests a way to get rid of the threshold, which is difficult to set. The prediction algorithm can specify arbitrarily long lists of the most beneficial objects to prefetch, sorted by benefit. Requests for these objects are issued in sorted order and it is ensured that these requests do not interfere with demand requests or other system activities. The clients send their access histories to a hint server which computes a hint list consisting of the most probable documents that the client will request in the future. This approach is similar to ours but there are some differences as well. The type of information and the means used by the prediction in both systems are similar but the purpose and the way of usage are different. Their hint list contains Web objects and depends on the experienced popularity of the objects and the access pattern of the specific user, while ours contains clients sorted by their relative ranks inside a client group for a specific video topic. The effects of the actions in the systems are different as well because Web objects are usually smaller and less resource intensive than videos but more frequently used. Therefore we want to perform fewer but longer valid actions, supporting them with hints from outside the system (which is less usual in Web prefetching systems because of the short validity and the large number of objects).

E-Commerce sites (like Amazon.com or Reel.com) use recommender systems to increase their sales [10]. For example, the Amazon.com "Customers who bought"-type recommendation generates a list of books which contains books frequently purchased by customers who purchased the selected book. This type of recommendation is based on item-to-item correlation and could be used to support our predictions. We could recommend videos which a client might be interested in and which can be accessed in good quality by them in the given situation.

3 Client behaviour prediction

When a video is loaded into the system, meta-information about it is provided using MPEG-7 descriptors regarding the physical, qualitative (MediaFormat D) and content-dependent (Classification DS) features of the video. Our prediction is based on the topic (Classification DS, Subject field). Currently, we assume that each video belongs to precisely one topic.

In the case of one-off programmes, we want to find out which of our clients are most interested in the topic of a given programme. Therefore information about the users' past interests is collected and used to rank the clients regarding their probable interest in a given video. In addition, popularity hints can be attached to the videos, which indicate what percentage of the people who are interested in the topic of the video will most probably watch the video. The server chooses a certain percentage of the most interested clients (e.g. according to the percentage given in the popularity hint), asks the CR for recommendations for *these* clients and schedules the appropriate actions (replicating videos, setting up software components) to establish the supposed configuration. (If no popularity hint is given, the ranked list of users can be cut off at a predefined or dynamically computed value.)

It is important to emphasize that we *do not want to predict the future request of single clients, but the ranking of the clients according to their interest in a given video topic*. Therefore the result of a prediction is a *ranked list of clients*. The predictions relate to the very next period in the future.

We examine three types of predictor functions (simple moving averages, autoregressive moving averages, the neural network approach) and three similarity measures. We use 23 predictors in all with the different parameter settings. These are presented in the next subsections.

3.1 Simple moving average predictors

We present seven predictor functions (p) here, some of them with more parameter settings. The values of the predictors are computed periodically, the length of the periods is equal and set externally (e.g. one week). Within every period, the value of the requests made by a client for a specific topic is stored in an entry of a history vector (h). Each element of h represents one period. A client's *request* contains the title of the video, the required date of the streaming and the client identifier. Each video instance has a *value* depending on the applied business model. In the artificial experiment (section 4.1) each video has the value 1, while in the real scenario (section 4.2), the value was the size of the requested object in bytes. The predictors compute their new values at the beginning of a new period from the values of the last N periods. For some predictors the size of h is one, for others it can be set externally. In h the most recent element is stored at index N and the oldest at index *1*.

We used two simple predictor functions: the *Last Value* and the *Last Difference*. The Last Value predictor always returns the most recent value, while the Last Difference predictor adds the last difference to the last value. They are defined as:

1. Last value (LastVal): $p(h) = h[N], \;\; N = 1$

2. Last difference (LastDiff): $p(h) = h[N] + (h[N] - h[N-1]), \;\; N = 2$

We used five other moving average predictor functions. The *Sum* predictor adds up the last N values and does the ranking accordingly. This predictor does not take into account how far back the periods are. The *Binomial* predictor considers the last value with α weighting and the previous *N-1* values with *1-*α weighting. We chose 0.75, 0.5 and 0.25 for α. This predictor can emphasize the most recent period and treats the older ones as equal. The *Quadratic* predictor uses quadratic weight depending on the distance back in the time; the older a value is, the lower the weighting. This predictor prioritizes the number of requests in more recent periods. The last two predictors are the *approximations* for the Binomial one and for the Quadratic one. They use the last predicted (p_k) and the last measured value (*h[k+1]*) for their predictions.

3. Sum: $p(h) = \sum_{i=1}^{N} h[i]$

4. Binomial (Binom): $p(h) = \alpha * h[N] + (1-\alpha) * \frac{\sum_{i=1}^{N-1} h[i]}{N-1}, \;\; \alpha = 0.75, \;\; 0.5, \;\; 0.25$

5. Quadratic (Quad): $p(h) = \sum_{i=1}^{N} (i^2 * h[i])$

6. Binomial approximation (Bin.A.):

$$p_1 = h[1],$$

$$p_{k+1} = \alpha * h[k+1] + (1-\alpha) * p_k \;\; , \;\; \alpha = 0.75, \;\; 0.5, \;\; 0.25$$

7. Quadratic approximation (Quad.A.):

$$p_1 = \left(\sum_{i=1}^{N} i^2 \right) * h[1] \;,$$

$$p_{k+1} = N^2 * h[k+1] + p_k - N^2 * \frac{p_k}{\sum_{i=1}^{N} i^2}$$

3.2 Autoregressive moving average and neural network based predictors: LPC, ARMA and ANN

In this section we continue to discuss the predictors under investigation used for estimating the clients' requests. The moving average methods mentioned above and their weighted counterparts are subsets of Autoregressive Moving Average (ARMA) time series models [11,12]. The basic idea behind ARMA is that the predicted value for the variable to be forecast is a weighted average of several previous values and the errors of the forecast at several previous time points. Naturally, the predicted value will not be perfectly accurate, but applying the error values in future weighted averages may produce more precise forecasts. The ARMA(*p,q*) model can be represented by the formula:

$$y(t) + a_1 y(t-1) + \cdots + a_p y(t-p) = e(t) + c_1 e(t-1) + \cdots + c_q e(t-q),$$

where y(t) is the value of the time series at time t, e(t) is the prediction error (i.e. the difference between the predicted value and the actual one), while $a_1 \ldots a_r$ and $c_1 \ldots c_q$ are free model parameters. Once we have fixed the model parameters, the actual predicted value $\tilde{y}(t)$ of the time series can be computed by:

$$\tilde{y}(t) = y(t) - e(t) = c_1 e(t-1) + \cdots + c_q e(t-q) - a_1 y(t-1) - \cdots a_p y(t-p)$$

Since in our case it is necessary to estimate the model parameters on line when the input data has been received, we apply – throughout this study – the recursive parameter estimation scheme with a normalized gradient approach (see [13]) as an adaptation method for fitting the model parameters. Linear Prediction Coefficients or LPC was originally a signal processing tool with applications in speech coding and filter design. LPC is a special case of ARMA modelling when q=0. It employs the autocorrelation method of autoregressive (AR) modelling to find the coefficients $a_1 \ldots a_r$, which leads to the well-known Yule-Walker equations [14]. As these equations form a symmetric Toeplitz-type linear equation system one can readily find the solution using the Levinson-Durbin algorithm [13]. We took this approach before carrying out our experiments. For testing purposes we employ the ARMA model with the parameter set (1,1),(1,2),...(3,3).

Besides the standard ARMA modelling approach, we also try an artificial neural network (ANN) approach as it is a viable technique for time series prediction as well [14]. We conduct experiments with this ANN method for the purposes of comparison. We train a linear network on a step-by-step basis in order to predict a time series. The network has *p* inputs, *p* delayed values of the time series, and has only one output which returns the predicted value. We use the Widrow-Hoff learning algorithm based on an approximate steepest descent procedure [15], with a value of *0.1* or *0.2* as the learning rate for incremental training.

3.3 Similarity measures

Similarity measures are used to compare the examined predictors with the perfect predictor. The perfect predictor gets the values from the next period in the future. Their similarities are measured by comparing the resulting ranked lists. We have chosen the following three measures: Kendall's tau [16], Spearman's footrule [18] and Ulam's distance [19]. We introduce to each measure a *precision function* which tells us what percentage of the predictions is correct on average. In the formulas *n* stands for the number of clients in a certain list.

Kendall's tau operates with the concept of conflict between elements. There is *conflict* between two elements when they are in a different order in the two ranked lists. Let *C* be the number of conflicts and *N* the number of non-conflicts in two lists. Then this similarity measure is defined as follows:

1. Kendall's tau: $x = \dfrac{N - C}{N + C}$, [-1,1] ↔ [only conflicts, no conflicts]

   Precision: $\dfrac{x+1}{2}$

Spearman's footrule uses the difference in rank of an element in the two lists. $\pi_k(i)$ means the place of the i-th element in permutation π_k where the permutations are the lists.

2. Spearman's footrule: $x = \dfrac{1}{2}\sum_{i=1}^{n} |\pi_1(i) - \pi_2(i)|$, the greater the result, the greater the difference

   Precision: $\dfrac{w-x}{w}$, $w = n * \left\lfloor \dfrac{n}{2} \right\rfloor - \left\lfloor \dfrac{n}{2} \right\rfloor^2$ (more about *w* in Appendix A)

Ulam's distance uses the maximum number of items ranked in the same order by both of the lists to compute the difference.

3. Ulam's distance: $x = n$ – *(the max number of items ranked in the same order by the two lists)*
   {0..n} ↔ {no conflicts ... only conflicts}

Precision: $\frac{n-x}{n}$

4 Test scenarios

4.1 Artificial dataset

We created 11 hypothetical clients with different request characteristics (see Fig 3.). We had 200 periods, but sometimes we show fewer periods (20 or 60) for sake of better readability. If the characteristic does not change, not all the periods are shown.

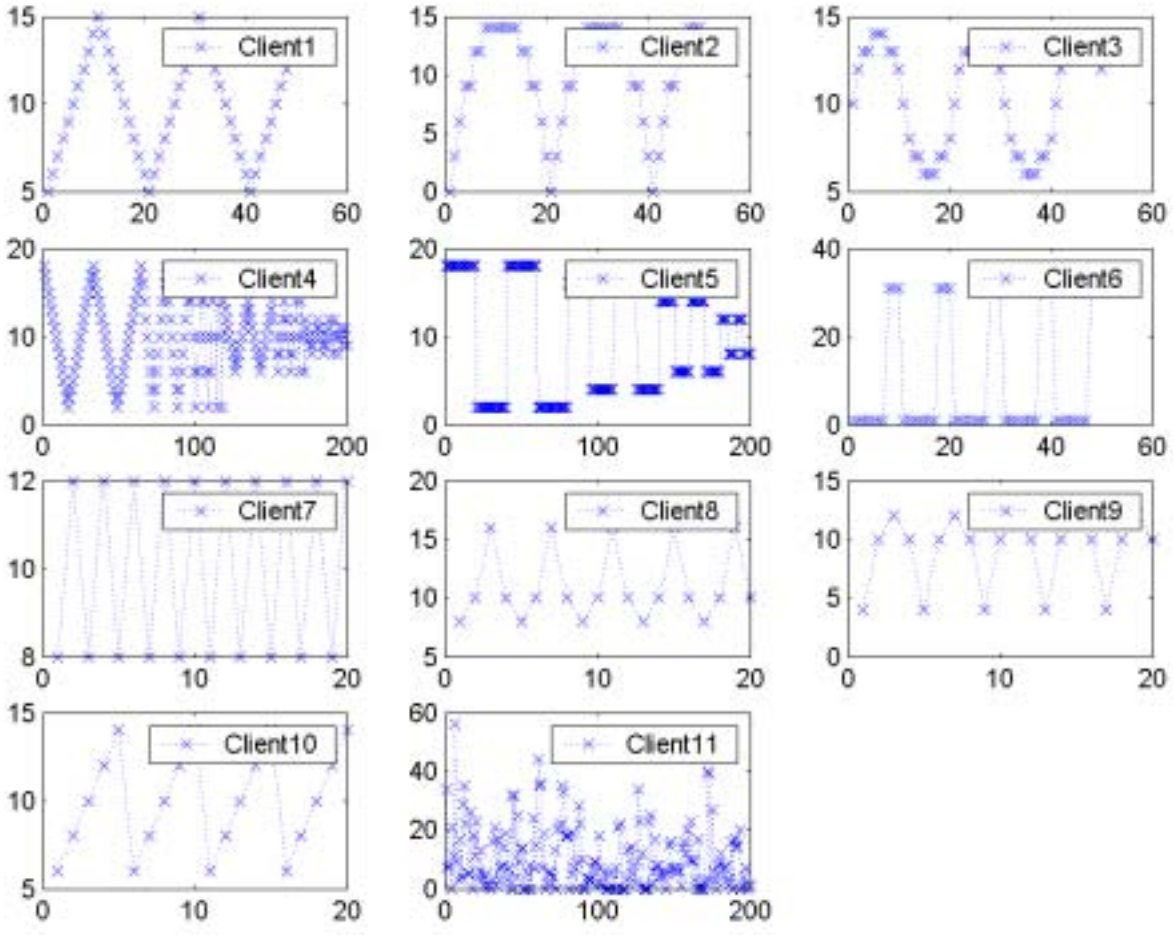


Fig 3. Artificial clients' characteristics (X: periods, Y: number of requests)

Client1's need grows and shrinks linearly – a rather abstract pattern of behaviour. Client2 is someone who needs to watch videos because of her job (e.g. a film reviewer or an advertisement developer). While running a project, she requests more videos of the given topic and between projects she only watches them on and off. We could say that two phases alternate in the lives of Client4 and 5; sometimes they have more time, sometimes less (e.g. somebody working only every second month). The tendency of their characteristics is similar to the transition from winter to summer when people are always outdoors more (and therefore spend less time in front of a screen): they watch fewer videos in their free time. The characteristic of Client6 shows us "weekend users", who only watch videos on their days off. The number of requests changes rapidly with Client7 depending on work patterns (e.g. a doctor or a fireman who is on duty for several days and then has a few days off.) In the case of Client10 we could think of students preparing for exams. There are universities where the students study one subject per period and at the end they have the exam. Let us assume that there are on-line videos about the lessons. As the exams come closer and closer, and more and more videos are available, the students' activity

watching the videos of the lessons increases. Client11 is a random user. Maybe it is harder to find an example for the clients who were not mentioned, but they can serve as a useful abstraction in order to get to know our predictors better.

We used regular, periodical characteristics here which rarely exist in this form in reality. In the case of a real user we expect some noise and less regular periodicity. We decided to use these less realistic patterns for the artificial dataset because they allow a more comprehensive reasoning.

4.2 World Cup 98 Web Log

We chose the access log of the World Cup 98 Website [20] to examine the predictor functions on a real dataset. This log file was recorded between 1st May and 23rd July 1998. There were more than 10^9 requests and almost 5000 GB of data were delivered. The Website had more than two and a half million clients. We chose the first 100 clients who requested the most content during the defined period, calculated the quantity of data requested daily in bytes, chose different period lengths given in days and made predictions for the next period using a limited number of previous periods as the basis. (All types of Web-data were involved in the choice of the 100 best clients because there was not enough video content.)

We predicted the ranked list of the clients regarding their activity in the next period. The characteristics of the clients' behaviour were chaotic, with big jumps regarding the amount of data requested through the periods, and therefore less predictable. (See figures in Appendix C.) No periodicity could be found which could be the basis for the prediction.

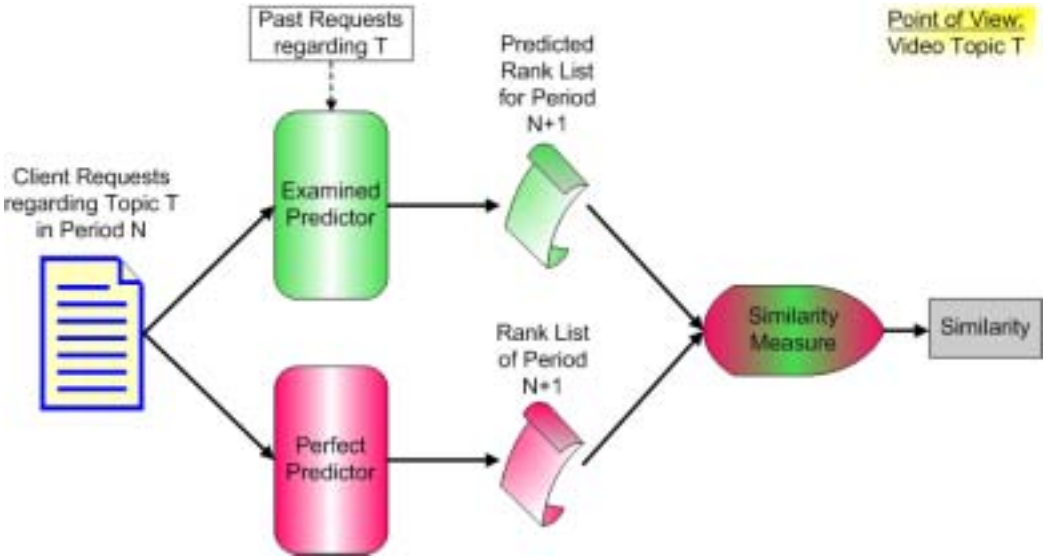
5 Results

5.1 Comparison method for the predictors

Here we present how the predictors are compared. We measure their precision on more datasets. Two test scenarios are chosen for comparison: one with artificial data and one with real data. The general comparison method works as follows:

Input: Spreadsheet of aggregated values of client requests for those who are interested in the given topic
Step 1: Rank the requests using the different predictors from period to period → ranked lists of clients
Step 2: Compare the ranked lists of a given period and predictor with the list of the perfect predictor in the same period using the 3 similarity measures → similarity values
Step 3: Compute the average similarity for each predictor per similarity measure through the periods → average similarities
Step 4: Compute the precision of the average similarities for each predictor per similarity measure → precision of the average similarities
Step 5: Compute the average precision for each predictor → average precisions
Step 6: Rank the predictors → predictor's ranks (the lowest is best)

(1.)



(2.)

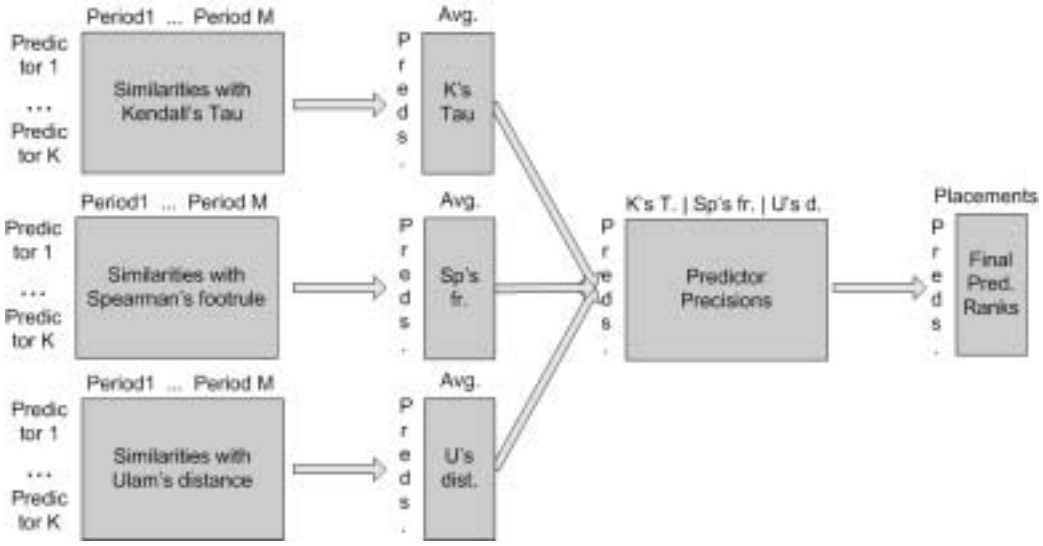


Fig 4. Comparing the predictors

In the case of M periods we used each predictor for the last M-1 periods.

5.2 Results from the artificial dataset

We created eleven client sets plus a combined client set to examine the predictor functions. First we generated five subtypes for each of our eleven kinds of client who always requested a constant number of videos, computed as follows (v is a vector and contains the number of requests of the original client for each period):

1. Maximum client: $\max(v)$
2. Above average client: $\text{avg}(v) + (\max(v) - \text{avg}(v)) / 2$
3. Average client: $\text{avg}(v)$
4. Below average client: $\text{avg}(v) - (\text{avg}(v) - \min(v)) / 2$
5. Minimum client: $\min(v)$

Our aim with these five constant clients was to study the precision and usability of the predictors regarding the different characteristics of requests. The twelfth client set consisted of the eleven original clients without the generated ones (called as "All clients"). As we already mentioned, we had 200 periods. The value of past periods taken into account by (some of) the predictors was 7.

We present the 9 most promising of the 23 predictors (based on the case with all clients). Their precision and *overall* placement (among the 23 predictors) is shown in Table 1. The first number is the average of the precision values given by the three similarity measures, followed by the placement in brackets. In the last column we show a summary (the average of the average precision values and the placements according to it).

	Client set 1 Avg. Prec.	Client set 2 Avg. Prec.	Client set 3 Avg. Prec.	Client set 4 Avg. Prec.	Client set 5 Avg. Prec.
LastVal	0.96278 (11)	0.9665 (2)	1 (1)	0.9893 (1)	0.98334 (1)
LastDiff	0.98195 (6)	0.95403 (4)	1 (1)	0.96752 (2)	0.97218 (6)
Binom 0.75	0.92481 (16)	0.94249 (6)	0.92835 (16)	0.94714 (12)	0.97478 (4)
Bin. A. 0.75	0.94417 (14)	0.95254 (5)	0.94584 (12)	0.93849 (16)	0.97962 (3)
LPC	0.98027 (7)	0.96985 (1)	0.99758 (2)	0.95775 (5)	0.98297 (2)
ANN 0.1	0.93272 (15)	0.69039 (23)	0.97125 (5)	0.94705 (13)	0.96101 (9)
ANN 0.2	0.95021 (13)	0.69114 (22)	0.98362 (3)	0.95552 (7)	0.96241 (8)
ARMA (1,1)	0.98195 (5)	0.93067 (9)	0.96706 (7)	0.95784 (4)	0.97301 (5)
ARMA (2,2)	0.98195 (1)	0.92332 (11)	0.94175 (14)	0.93691 (17)	0.93411 (17)

	Client set 6 Avg. Prec.	Client set 7 Avg. Prec.	Client set 8 Avg. Prec.	Client set 9 Avg. Prec.	Client set 10 Avg. Prec.
LastVal	0.94919 (1)	0.77908 (19)	0.80532 (18)	0.78699 (19)	0.94919 (2)
LastDiff	0.94919 (1)	0.70389 (22)	0.82794 (16)	0.83799 (14)	0.91653 (4)
Binom 0.75	0.94919 (1)	0.74102 (18)	0.80532 (17)	0.78801 (18)	0.86656 (16)
Bin. A. 0.75	0.93505 (2)	0.74102 (18)	0.79164 (21)	0.78234 (20)	0.83752 (22)
LPC	0.92965 (5)	0.99395 (1)	0.99507 (1)	0.99488 (1)	0.99488 (1)
ANN 0.1	0.92528 (6)	0.95096 (9)	0.878 (3)	0.91392 (3)	0.91541 (5)
ANN 0.2	0.93086 (3)	0.97264 (8)	0.92034 (2)	0.94854 (2)	0.94016 (3)
ARMA (1,1)	0.93058 (4)	0.77173 (17)	0.83008 (15)	0.83045 (15)	0.8887 (7)
ARMA (2,2)	0.87605 (13)	0.98344 (5)	0.83259 (14)	0.84599 (7)	0.85837 (18)

	Client set 11 Avg. Prec.	All clients Avg. Prec.	Summary Avg. Prec.
LastVal	0.89717 (2)	0.6654 (4)	0.89452167 (2)
LastDiff	0.86469 (19)	0.60944 (9)	0.8821125 (4)
Binom 0.75	0.89633 (4)	0.61553 (6)	0.86496083 (8)
Bin. A. 0.75	0.90266 (1)	0.6264 (5)	0.86477417 (9)
LPC	0.89056 (7)	0.78121 (1)	0.95571833 (1)
ANN 0.1	0.68435 (23)	0.67948 (3)	0.86519091 (7)
ANN 0.2	0.68611 (22)	0.70533 (2)	0.88724 (3)
ARMA (1,1)	0.87577 (14)	0.61545 (7)	0.87944083 (6)
ARMA (2,2)	0.85883 (20)	0.61141 (8)	0.88206 (5)

Table 1. Precision and placement of the most promising predictors in the artificial case (organized in three spreadsheets)

The first eleven sets with the generated clients are quite easy to predict and the predictors mostly work very precisely. The most precise predictor is the LPC with 95% average precision (computed from the average precision of all sets in the column Summary). It is 6% better than the second best which is, surprisingly, the simplest, the Last value predictor. The average fluctuation in the Summary for the placements 2 to 9 is less than 3%, thus we could think that those predictors are almost equal. But they show quite great differences in precision depending on the client set. (For example, Client sets 7, 8 and 9 are especially hard for the Last value predictor because they frequently contain big jumps; however, it is perfect for Client set 3.) Client 11 has random behaviour for which the two simplest predictors, the Binomial approximation with α=0.75 and the Last value, are the best. We expect a similar behaviour in real clients and therefore we consider these two predictors to be promising candidates for the future (alongside LPC). For the set of all original clients (All clients) the LPC is best again, and it seems to be the most precise predictor in general. If we also consider the resource need of the predictors (e.g. CPU, how many previous data they need to store per client), then we should consider the Last value and Binomial approximation predictors as well.

5.3 Results from the World Cup 98 dataset

The request values (how many bytes of data a client requested in a period) were quite large numbers in the case of the 100 best clients. We made 4 types of computation series: without any modification of the data, dividing the input by one million, using stepwise normalization for all predictors, and using the same only for the LPC, ARMA and the ANN predictors. We got the best precision with the last type of computation therefore we only discuss this one here.

In the stepwise normalization the average and the deviation of the already known requests were computed. In each period we computed them again with the new request value. The normalization is computed by subtracting the average of the known elements of the series from the new element and dividing its result by the actual deviation. Therefore we had to store four values for each client. (The Matlab code and explanation for this computation is attached in Appendix B.) The normalization was performed before the computation of the predicted value and after this, the inverse operation on the result was applied.

We carried out 6 test scenarios with the period lengths of 1, 4 and 7 days, the number of past periods taken into account being 4 and 7. We chose the 9 most promising of the 23 predictors. Their precision and *overall* placement (among the 23 predictors) is shown in Table 2, where ‘p’ means the period length in days and ‘pp’ means the number of past periods. The first number is the average of the precisions given by the three similarity measures, followed by the placement in brackets. In the last column we show a summary (the average of the average precisions and the placements according to it).

	p = 1 day, pp = 4 p Avg. Prec.	p = 1 day, pp = 7 p Avg. Prec.	p = 4 days, pp = 4 p Avg. Prec.	p = 4 days, pp = 7 p Avg. Prec.
LastV	0.58869 (1)	0.58869 (1)	0.58195 (5)	0.58195 (4)
Binom 0.75	0.58041 (2)	0.57893 (2)	0.56578 (10)	0.56404 (8)
Binom 0.5	0.58002 (3)	0.57852 (3)	0.56721 (8)	0.56144 (9)
Bin. A. 0.75	0.57734 (5)	0.57734 (5)	0.57432 (6)	0.57432 (6)
Quad.	0.57853 (4)	0.57746 (4)	0.58754 (4)	0.57941 (5)
LPC	0.56142 (13)	0.57012 (10)	0.58862 (2)	0.58954 (1)
ANN 0.1	0.56746 (11)	0.56653 (12)	0.58768 (3)	0.5837 (3)
ANN 0.2	0.57168 (10)	0.57147 (9)	0.58897 (1)	0.58465 (2)
ARMA (3,3)	0.49092 (21)	0.49092 (21)	0.52661 (18)	0.52661 (18)

	p = 7 days, pp = 4 p Avg. Prec.	p = 7 days, pp = 7 p Avg. Prec.	Summary Avg. Prec
LastV	0.56649 (5)	0.56649 (5)	0.57904333 (3)
Binom 0.75	0.55312 (12)	0.55203 (12)	0.56571833 (5)
Binom 0.5	0.54486 (17)	0.54837 (13)	0.56340333 (6)
Bin. A. 0.75	0.55378 (11)	0.55378 (11)	0.56848 (4)
Quad.	0.58051 (2)	0.57405 (2)	0.57958333 (2)
LPC	0.6232 (1)	0.68164 (1)	0.60242333 (1)
ANN 0.1	0.49539 (22)	0.493 (23)	0.54896 (8)
ANN 0.2	0.49459 (23)	0.49577 (22)	0.55118833 (7)
ARMA (3,3)	0.57132 (3)	0.57132 (3)	0.52961667 (9)

Table 2. Precision and placement of the most promising predictors in the real scenario (organized in two spreadsheets)

LPC proved to be the best predictor again, followed by the Quadratic and the Last value predictors slightly more than 2% below. LPC's precision improved when the length of the period and/or the number of the known past periods were increased while the precision of Last value decreased slightly when the period length was increased (the number of past periods does not affect this predictor). The Quadratic predictor's precision fluctuated at around 58% so it was quite stable. The Binomial approximation with α=0.75 was the 4th best and thus a suitable candidate for us.

6 Conclusions

It is a major problem that no real and fitting data sets are available and therefore we cannot draw general conclusions about the current behaviour practice of our potential clients. These test scenarios can only give us directions for future research. The elaborate LPC predictor proved to be the best among the chosen predictors with quite good average precision values (95% in the artificial case with all client sets, 78% with the set of all original clients and 60% in the real, quite chaotic scenario). We can see that the very simple Last value and the Binomial approximation predictors performed surprisingly well; therefore they are our "secret favourites". The Quadratic and ANN predictors proved to be effective sometimes and therefore we should use them in future experiments as well. Some of the ARMA predictors gave good results in the different cases but never the same ones. We are thinking about using this approach with dynamic parameter settings.

7 Problems and future work

There are some problems with our models about the topic of the videos and popularity hints. In reality, videos cannot be classified effectively using this simple approach (as they tend to fit into more than one category depending on sub plots). Second, the scope of the popularity hints is limited (e.g. we can't express such common sense issues such as "it is summer, people spend less time indoors so they watch fewer videos"). Third, we assume that the clients who saw the most videos belonging to a given topic will be most likely interested in another video of that given topic and this might not always be the case. In addition, this approach does not guarantee that the spatial-optimization module (CR) targets the right group of clients. In an extreme case it can happen that our best clients are standing alone (far from other users in the Internet) since the medium-ranked users comprise bigger groups (which means they are close to each other in the net) with a higher number of requests on average. In the background we assume an equal business model i.e. all of our users have the same importance.

We want to refine the model for the topics, for example a hierarchical model with subtopics, like the classification system in a library. Another alternative extension is if we let the videos belong to more than one topic, for example X% to topic A and Y% to topic B. Alongside topic we could also include the preferred quality in the prediction (e.g. low quality for mobile clients).

We have to examine how well our popularity hint model works. It would be interesting to use more information from external sources, like the expected popularity hint distribution of a video or knowledge about the "seasons" of video viewing throughout the year.

Another issue is the similarity measures. We used three of them and these computed quite large differences regarding precision. In our case it is an important difference if we predict the right order of our *best clients* (which will be taken into account by the spatial optimization) or of the *entire group* of clients. Therefore we want to find better fitting similarity measures.

It would be interesting to identify general patterns in the clients' behaviour to classify them (if such patterns exist). Each class could then have a best fitting predictor as the result in the artificial scenario showed. When clients change their behaviour pattern, the system should recognise that, choose the new class and change the predictor dynamically. We should identify a set of predictors which fit the main types of clients.

References

1. Roland Tusch, László Böszörményi, Balázs Goldschmidt, Hermann Hellwagner, Peter Schojer **Offensive and Defensive Adaptation in Distributed Multimedia Systems** *Computer Science and Information Systems (ComSIS), Vol. 1, No. 1, February 2004, pp. 49-77.*
2. Roland Tusch **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*
3. Balazs Goldschmidt, Roland Tusch, Laszlo Böszörmenyi **A Mobile Agent-based Infrastructure for an Adaptive Multimedia Server** *Technical Report No, TR/ITEC/03/2.05; Technical Report Version of the DAPSYS Journal Paper to appear in Parallel and Distributed Computing Practices, Special issue on DAPSYS 2002.*
4. Balazs Goldschmidt, Tibor Szkaliczki, Laszlo Böszörmenyi **Placement of Nodes in an Adaptive Distributed Multimedia Server** *Technical Reports of the Institute of Information Technology, University Klagenfurt, TR/ITEC/04/2.06, submitted to EuroPar 2004*
5. Roland Tusch **Design and Implementation of an Adaptive Distributed Multimedia Streaming Server**, *PhD Thesis, University Klagenfurt, 2004*
6. Apple Computer, Inc. **QuickTime Streaming Server, Darwin Streaming Server: Administrator's Guide**, *2002*. URL: http://developer.apple.com/darwin/projects/streaming/.
7. Helix Community **The Helix Platform***, 2002* URL: https://www.helixcommunity.org/2002/intro/platform.
8. Real Networks, Inc. **Helix Universal Server Administration Guide***, 2003*. URL: http://docs.real.com/docs/HelixServer9.pdf.
9. Ravi Kokku, Praveen Yalagandula, Arun Venkataramani, Mike Dahlin **A Non-interfering Deployable Web Prefetching System***, In 4th USENIX Symposium on Internet Technologies and Systems. Mar. 2003*
10. Schafer, J.B., Konstan, J.A., and Riedl, J. **Recommender Systems in E-Commerce**, *1999 in ACM Conference on Electronic Commerce (EC-99), pages 158-166.*
11. Box, G.E.P. and G.M. Jenkins (1976), **Time Series Analysis: Forecasting and Control**, *2nd ed., San Francisco: Holden Day*.
12. Gourieroux C. and A. Monfort (1997), **Time Series and Dynamic Models**. *Cambridge: Cambridge University Press*.
13. Ljung, L., **System Identification: Theory for the User***, Prentice-Hall, 1987, pp. 278-280.*
14. Bishop, C. M. (1995), **Neural Networks for Pattern Recognition**. *Oxford: Oxford University Press*.
15. S. Haykin, "**Neural Networks**", *Macmillan, 1994*
16. M. G. Kendall. **Rank Correlation Method**. *Hafner Publishing Company, New York, 1962. Third Edition*.
17. V. Ha, P. Haddawy **Similarity of Personal Preferences: Theoretical Foundations and Empirical Analysis**, *Artificial Intelligenc*e, 146(2):149-173, June 2003
18. C. Spearman. **Footrule for measuring correlation**. *British Journal of Psychology, 2:89-108, June 1906*.
19. S. M. Ulam. **Future applications of mathematics in the natural sciences**. *American Mathematical Heritages: Algebra and Applied Mathematics. Texas Tech. University, Mathematics Series., 13:101-114, 1981.*
20. URL: http://ita.ee.lbl.gov/html/contrib/WorldCup.html

Appendix A: Worst case scenario for Spearman's footrule

The worst case is if π_2 is the inverse of π_1. From this it follows:

$$\frac{1}{2}*\sum_{i=1}^{n}|\,\pi_1(i)-\pi_2(i)\,| = \frac{1}{2}*\sum_{i=1}^{n}|\,2*i-(n+1)\,| = \frac{1}{2}*\left(\sum_{i=1}^{\left\lfloor\frac{n}{2}\right\rfloor}(n+1)-2*i+\sum_{i=\left\lceil\frac{n}{2}\right\rceil+1}^{n}2*i-(n+1)\right) =$$

$$(*) = \frac{1}{2} * 2 * \sum_{i=1}^{\left\lfloor \frac{n}{2} \right\rfloor} (n+1) - 2 * i = \left\lfloor \frac{n}{2} \right\rfloor * (n+1) - 2 * \sum_{i=1}^{\left\lfloor \frac{n}{2} \right\rfloor} i = n * \left\lfloor \frac{n}{2} \right\rfloor + \left\lfloor \frac{n}{2} \right\rfloor - 2 * \frac{\left\lfloor \frac{n}{2} \right\rfloor * \left(\left\lfloor \frac{n}{2} \right\rfloor + 1 \right)}{2} =$$

$$= n * \left\lfloor \frac{n}{2} \right\rfloor - \left\lfloor \frac{n}{2} \right\rfloor^2 .$$

Remark:

At (*) if $\left\lfloor \frac{n}{2} \right\rfloor \neq \left\lceil \frac{n}{2} \right\rceil$ then

$$i = \left\lfloor \frac{n}{2} \right\rfloor \Rightarrow 2 * \left\lfloor \frac{n}{2} \right\rfloor - (n+1) = 0 \Rightarrow \sum_{i=1}^{\left\lfloor \frac{n}{2} \right\rfloor} (n+1) - 2 * i = \sum_{i=\left\lfloor \frac{n}{2} \right\rfloor + 1}^{n} 2 * i - (n+1) .$$

Appendix B: Stepwise Normalisation

Matlab code portion for computing *stepwise normalization*:

```
…
[c_period c_avg c_qsa c_dev] = stepwiseDeviation(c_period, c_avg, c_qsa, c_dev, clientReqs(i,j));
if c_dev ~= 0
      clientReqs(i,j) = (clientReqs(i,j) - c_avg) / c_dev;
else
      clientReqs(i,j) = 0;
end
…
```

Explanation – the normalization is computed by subtracting the average (c_avg) of the known elements of the series from the new element and dividing its result by the actual deviation (c_dev).

Matlab code for computing *stepwise deviation*:

```
function [period_new, avg_new, quad_sum_avg_new, dev_new] = stepwiseDeviation(period_old, avg_old,
        quad_sum_avg_old, dev_old, x_new)
period_new = period_old + 1;
avg_new = avg_old * (period_old / period_new) + x_new / period_new;
quad_sum_avg_new = quad_sum_avg_old * (period_old/period_new) + (x_new*x_new) / period_new;
dev_new = sqrt( quad_sum_avg_new - (avg_new * avg_new) );
```

Explanation – the code is based on the well known correspondence:

$$dev(x) = \sqrt{\frac{\sum_{i=1}^{n} x_i^2}{n} - \left(\frac{\sum_{i=1}^{n} x_i}{n} \right)^2}$$ where x is the request vector, n is the number of elements in x

and quad_sum_avg_old/new in the code means: $\frac{\sum_{i=1}^{n} x_i^2}{n}$.

Appendix C: World Cup 98 clients' characteristics

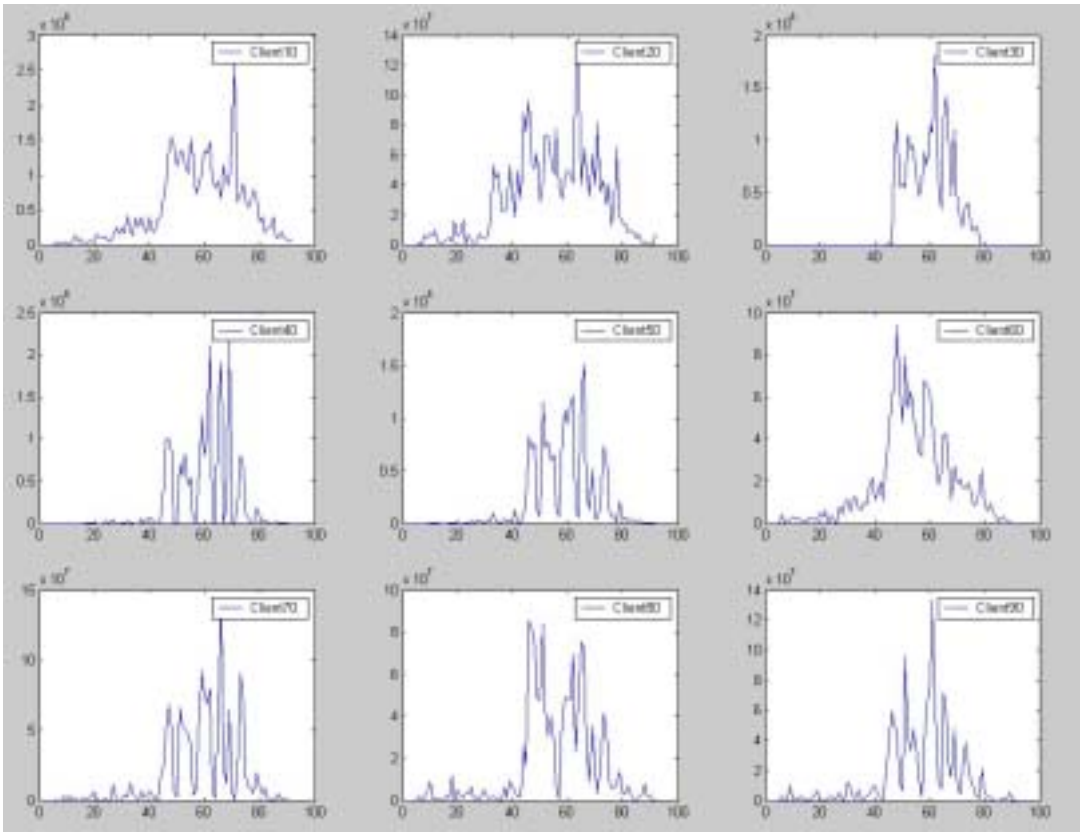


Fig 1. Requests from clients 10 – 90 (period = 1 day; X: periods, Y: requested bytes)

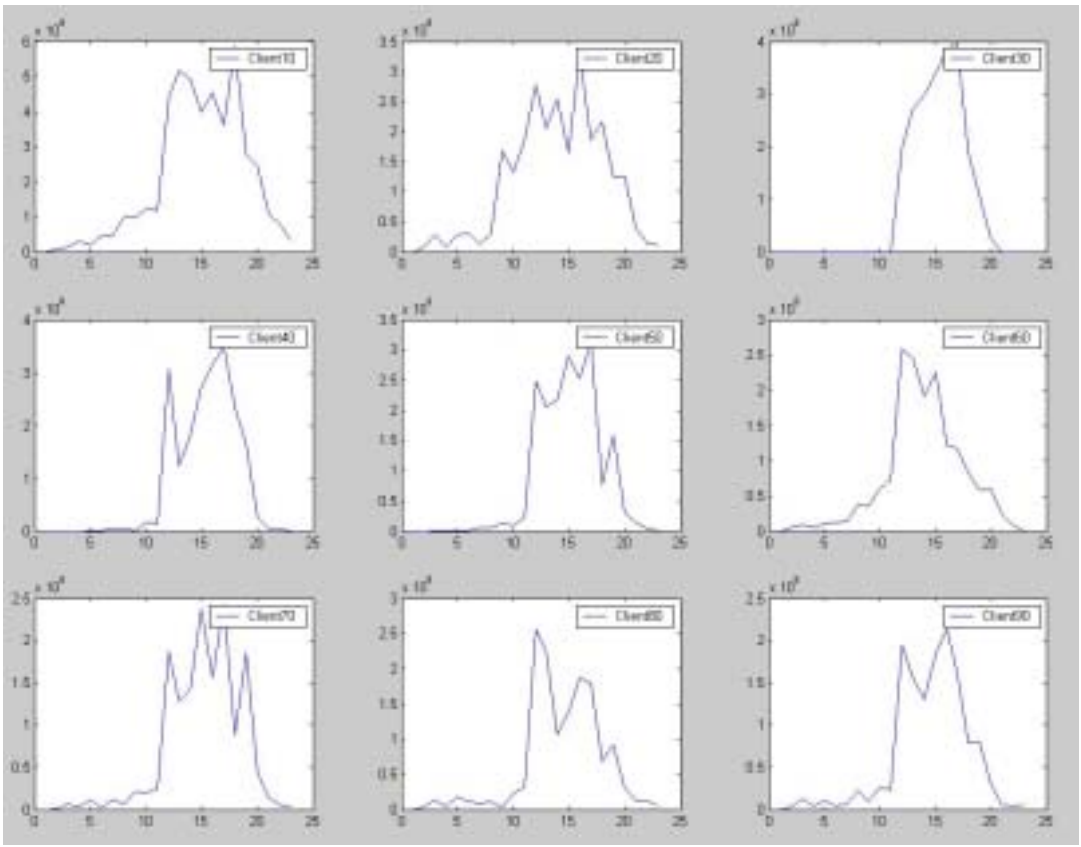


Fig 2. Requests from clients 10 – 90 (period = 4 days; X: periods, Y: requested bytes)

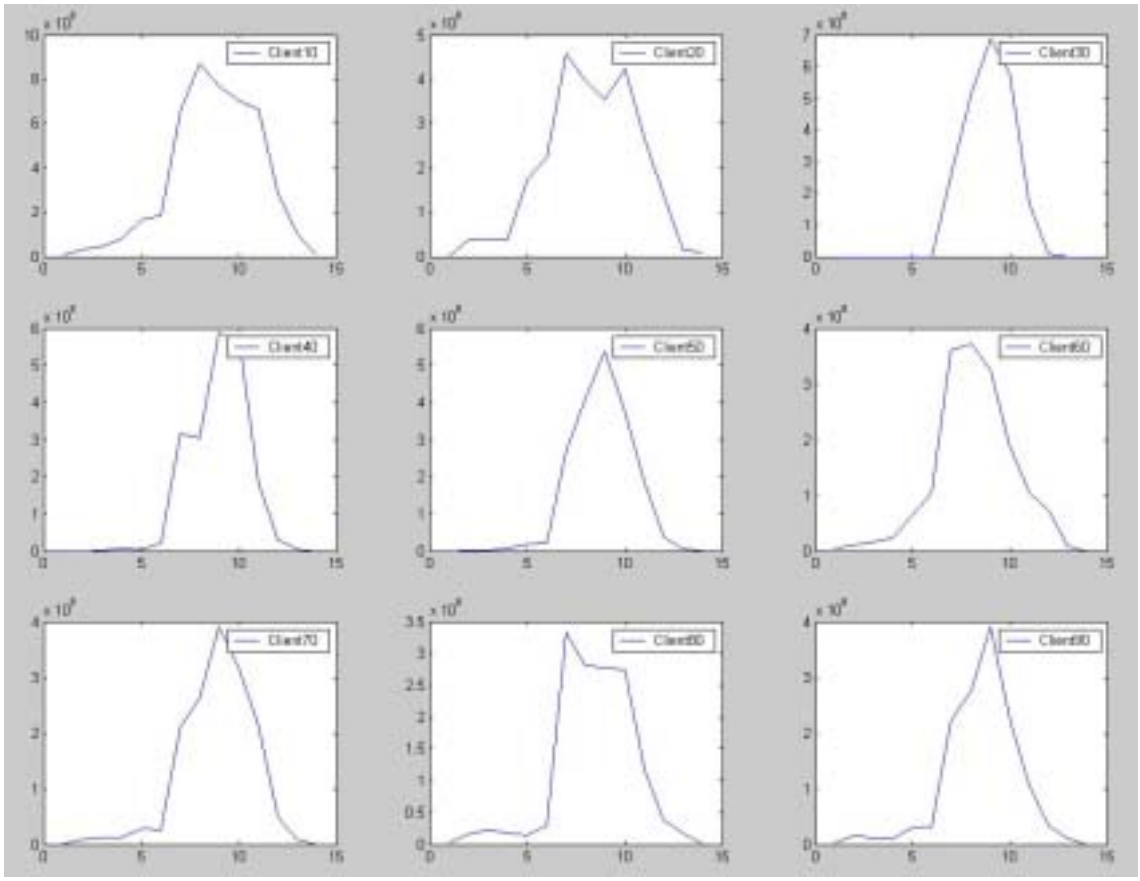


Fig 3. Requests from clients 10 – 90 (period = 7 days; X: periods, Y: requested bytes)

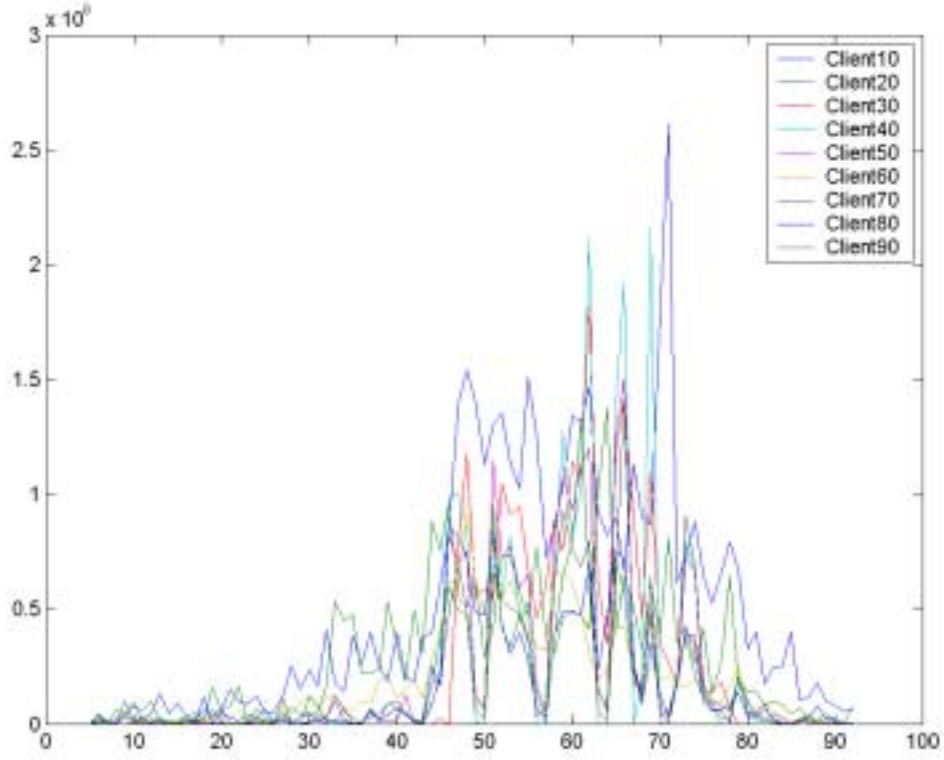


Fig 4. Frequently changing client ranks (1) (period = 1 day, past periods = 7; X: periods, Y: requested bytes)

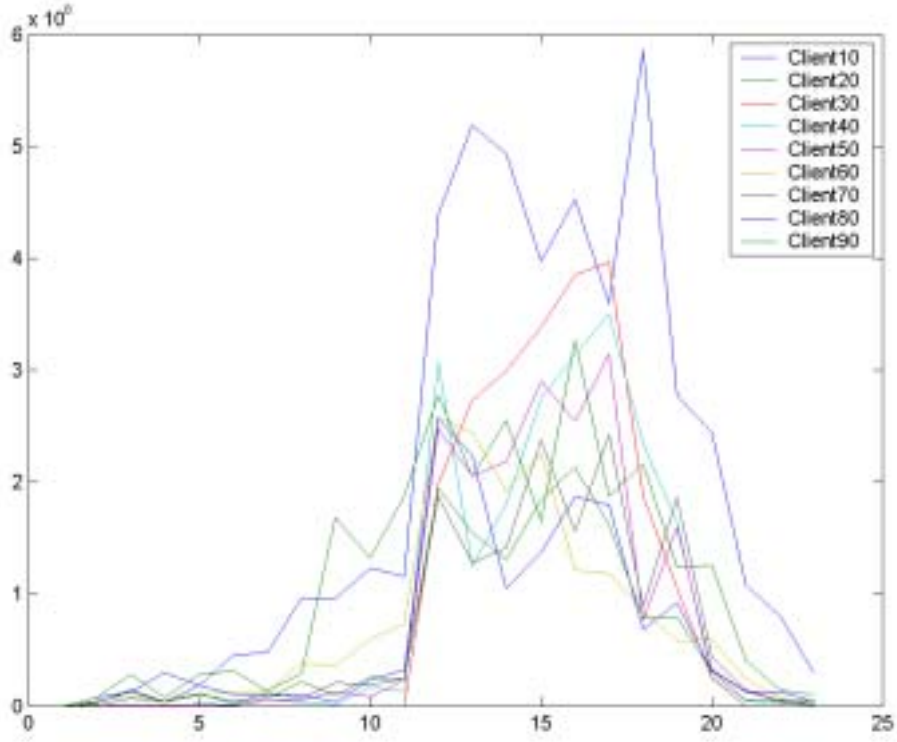


Fig 5. Frequently changing client ranks (2) (period = 4 days, past periods = 7; X: periods, Y: requested bytes)

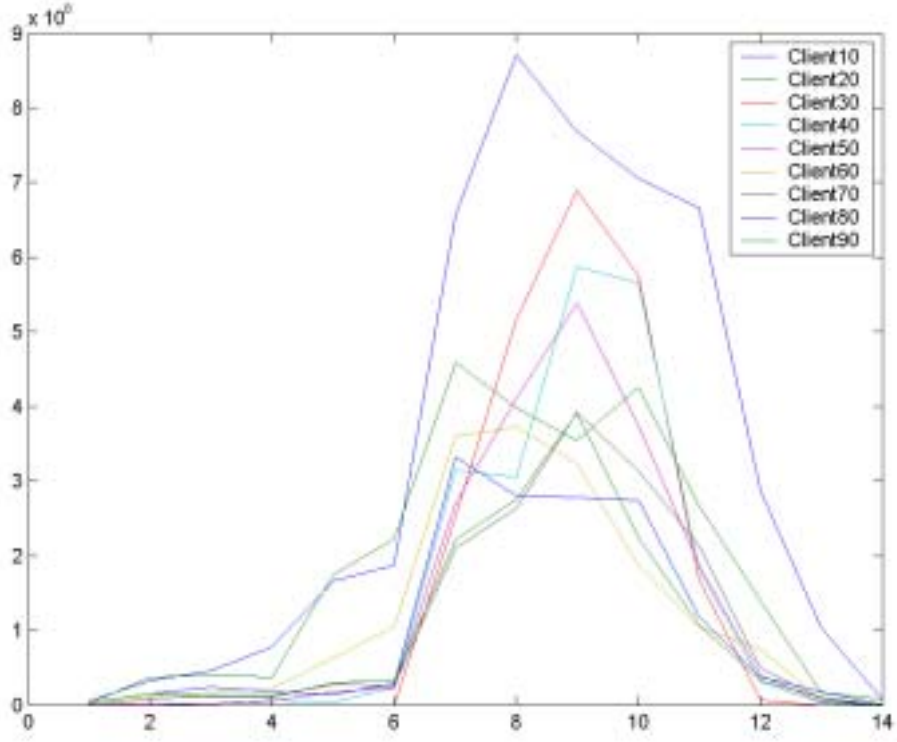


Fig 6. Frequently changing client ranks (3) (period = 7 days, past periods = 7; X: periods, Y: requested bytes)