

OPTIMIZING REQUESTS FOR THE SMART DATA SERVER*

ERNST-GEORG HAFFNER, UWE ROTH, THOMAS ENGEL, CHRISTOPH MEINEL

Institute of Telematics
Bahnhofsstr. 30-32
54292 Trier, Germany
+49 (0) 651-97551-0

Email: {haffner | roth | engel | meinel}@ti.fhg.de

ABSTRACT

The growing complexity of the Web information flow makes it necessary to improve Client/Server communication. In order to efficiently maintain heterogeneous and fast changing data types and different kinds of data sources, a Smart Data Server (SDS) can be used as a general framework for distributed functionality.

In this paper, we present three general methods of advanced Client/Server communication: *Prediction*, *Minimization* and *Request Optimization*, - implemented and in practical use as modules of the SDS.

Even though the overall aim of these features is the improvement of the information flow, the processing is quite different: *Prediction* aims at making pre-calculations or pre-fetching of very probable user requests in the future, *Minimization* leads to a reduction of transferred data while considering system requirements of the client (e.g. display resolution), and *Request Optimization* is a usage-dependent data request method. We will show that only a skilful combination of these methods can guarantee success in improving network communication.

Keywords

Network Performance, Prediction, Minimization, Request Optimization

1. INTRODUCTION

Due to the extremely fast growth of the Internet - especially the World Wide Web (WWW) - there is also an increasing amount of data and information flow between several servers and clients. Therefore, the need for central systems to collect and distribute information from heterogeneous data sources is also growing.

The Smart Data Server (SDS) as a general framework for distributed functionality has been constructed to fulfill the needs described above [1]. The SDS is able to connect different data sources and improve information exchange. The system serves as middle tier in a three tier architecture [2]. It can work together with several C/S

components and structures and is a pure Java implementation [3]. The overall idea is to put intelligence on the server side to increase the efficiency of the communication with the clients.

Similar approaches can be found in Satoshi Hirona's HORB, an object-oriented request-broker [4]. The Swift company works on a product using RMI of Java [5]. The idea of the "Common Request Broker Architecture" (CORBA) is also related to the presented work [6]. In this approach, several resources can be distributed over the network. DCOM [7] and Java's RMI [8] are also similar to the basic ideas of the SDS.

In this paper, we will show three general techniques to improve communication and data exchange between clients and servers and its usage as part of SDS modules:

- Prediction
- Minimization
- Request Optimization

The following three sections deal with a detailed discussion of these aspects. Section 5 sketches their implementation as modules of the SDS. We will see that only a clever combination of the various methods to improve communication will lead to increased efficiency. The paper finishes with the "conclusion" (section 6).

2. PREDICTION

There are several possibilities to overcome the problem of high user-perceived latencies after requesting a document. One idea is to predict future requests. Thus, time consuming calculations on the server side can be carried out before a request is made. If the server is very "sure" that certain documents will be requested in the near future, the according data can be send to the client (or pre-fetched by the client) even though the user does not know it.

* In Proc. "Applied Informatics", IASTED AI2000, Innsbruck, Austria, 2000, pp. 481-486

Many attempts were made to find the best and most efficient algorithms for fulfilling Prediction needs. Discrete Markov-Chain-models were the basis for the first Prediction algorithms for the Web ([9] and [10]). Later, these ideas were extended to a continuous chain approach [11] and path-profiling [12]. Corvella et. al. and Careres et. al. focus on a discussion of the negative effects of incorrect Prediction ([13], [14]). A very fundamental algorithm for Prediction is presented in [15].

Based on these concepts, we generated a test scenario for Prediction characteristics. It serves as the basis for an implementation of a SDS Prediction module [16]. For a general discussion of performance modeling see [17].

The critical point of Prediction is that the cost for the reduction of user-perceived latency is paid with an increasing amount of network load. Therefore, the Prediction algorithm must compare the probability that a future user request will be made with the costs of an incorrect pre-calculation or pre-fetching.

According to [16], every user session is modeled as a binary \mathbf{A} -vector of data set indices:

$\mathbf{S} = (s_i)$ with:

$$\begin{aligned} s_i &= 1 \text{ if the data set } \mathbf{d}_i \text{ was requested by the client;} \\ s_i &= 0 \text{ otherwise} \end{aligned}$$

Thus, \mathbf{A} represents the total number of possibly predictable data sets. A very important part of the Prediction algorithm is the symmetric, quadratic matrix φ , where the elements are frequencies of requests. The matrix φ is called the *Memory Matrix*.

$$\varphi = (\varphi_{ij}) \text{ with } 1 \leq i, j \leq A.$$

Thus, the Prediction algorithm is based on relative frequencies and the corresponding relative probabilities of user requests. φ is initialized as 0-matrix:

$$\varphi_0 = (\varphi_{ij}) = 0 \quad \forall 1 \leq i, j \leq A \quad (1)$$

Within every session \mathbf{s} , the matrix φ is calculated to φ' , so that the elements of φ represent the frequencies of data requests, depending on requests from other data sets.

$$\varphi' = (\varphi'_{ij}) = \begin{cases} \varphi_{ij} + 1 & : \text{ iff } s_i = 1 \wedge s_j = 1 \\ \varphi_{ij} & : \text{ otherwise} \end{cases} \quad (2)$$

The conditional probability p_i , that \mathbf{d}_i will be requested in the near future if \mathbf{d}_j has been requested during the same session \mathbf{s} can be calculated as:

$$p_i(\mathbf{d}_i | \mathbf{d}_j) = \frac{\varphi'_{ij}}{\varphi_{ij}} \quad (3)$$

With this simple approach we are already able to predict future user requests. To overcome the problem of time and cost consuming incorrect Predictions, we have to compare the probability of (2) with the costs of the Prediction itself. Only if the request probability exceeds

the expected cost reduction the Prediction does make sense. In accordance with [16], we use the condition as threshold:

$$p_i > \frac{1 + S_o(d_i)}{1 + S_f(d_i)} \quad (4)$$

where S_o describes the *Prediction-Overhead* that has to be small in comparison to S_f , the *System Flexibility*, for a low threshold. The Prediction-Overhead stands for the costs of the Prediction itself (bookkeeping-costs). Especially for a low degree of guesses, this factor can be very high. The System-Flexibility is a quotient of the avoidable costs and the resource costs for data requests that are not avoidable. Undoubtedly, one can only expect the Prediction algorithm to result in improved efficiency if the System-Flexibility is very high.

As a rule of thumb, one can say that a low threshold leads to numerous Prediction tries while a high threshold guaranties accurate request guesses (details can be found in [16]). The critical points of an implementation of the described Prediction algorithm as module of the SDS are highlighted in 6.1

Prediction is not always a good idea. The advantages of a predictive approach depend on the following conditions:

- Actual user-perceived latency
- Volume of the requested data
- Complexity of the requested data
- System costs for pre-calculations
- Network load
- Randomness of user behavior

The idea of Prediction is based on the fact that it is unacceptable for a user to wait more than an instant for the server response. If several other things can be done, though, while the server side is providing the response, the user-perceived latency may be rather insignificant.

It depends on the data - mostly on the server side - whether Prediction algorithms can be used. Data that is rather "fugacious" can not be pre-calculated or pre-fetched since it may be obsolete just when it is needed. Perhaps an accurate Prediction could be carried out but the data is too simple to be pre-calculated or too voluminous to be pre-fetched¹.

Furthermore, the behavior of the user plays an important role: Prediction can only be applied where the requests are somewhat constant. The higher the degree of randomness in the behavior of the users, the smaller the

¹ Pre-fetched data must be "distributed" during the time interval between two requests. For large data volumes this time span may be too short, especially for high network loads.

effect of the Prediction module. We will see in section 3 that - depending on user behavior - Prediction or Minimization is the proper idea for improving the Client/Server communication. Prediction alone can only help to improve C/S communication in a selected area of cases. Nevertheless, due to the concept of a general framework for distributed functionality the SDS is a very good platform for Prediction algorithms: the SDS can verify the performance of the Prediction approach.

3. MINIMIZATION

To overcome the disadvantages of incorrectly predicted user requests we constructed a scheme to reduce the transferred data by investigating superfluous or redundant data.

This idea runs counter to a general trend of Client/Server communication. While normally *Two- and Three-Tier-Architectures* [18] in contrast to central computing often transfer a maximum of data from the server to the client for future manipulating and working on it, it can be sufficient – especially for the Internet – to do the opposite. Sending only exactly the data that the client can use should improve C/S communication and reduce network load and also – as Prediction – user-perceived latency. Furthermore, this method is based on an idea that contrasts with Prediction: instead of sending more data than requested, it could be useful - in some cases - to send less!

For instance, the display resolution of the client hardware is a strong limitation for the amount of data that the server should return. If there is no further need of the information it would be senseless to send more data than the client can use. Of course, these two ways are equivalent: the client tells the server either how much data can be evaluated or what resolution can be found on its side. In both cases, the “smart” server has to work on the data to return a minimal amount of “high-class“ values.

A typical example for such a Minimization of transferred data can be found in finance, especially in portfolio-management applications. If the client application requests stock values of a few years, it could be quite pointless to sent a greater amount of values than the client hardware can resolve. It would be especially superfluous to be sending more than one stock value per day. Thus, the data volume exchanged between client and server can be minimized depending on client hardware/software characteristics and the data structure on the server side.

The following calculation shows the potential of such Minimization (example 1):

10years·300days·100values·10Bytes ≈ 3 MB of data
RESPONSE: (with Minimization 800·600resolution)
 800 average values·10Bytes ≈ 8 KB of data

Example 1: Request Minimization

There are two main problems concerning Minimization:

- What are the costs of Minimization on the server side?
- How can one be sure that the user does not need more data than requested?

Obviously, example 1 shows that the price to be paid on the server side consists of more calculations to retrieve 800 average values from 30.000 stock data prices per day. It would not be correct to simply ignore 36 numbers and return only every 37th value (30000/800≈37). What about average prices, what about the minimum and maximum stock prices during this period? To overcome this problem, we will see (in section 5.2) that it becomes necessary to send some boundary values (meta-data) together with the average data. The problem described here is not only a critical point for Minimization techniques. Even if all existing values would have been sent to the client, the problem would still remain: which data should the client software show if the display resolution was too small to present all of it?

The answer to the second question is astonishingly simple. The same Prediction algorithm of section 2 can also be used to calculate the probability of subsequent requests. The Minimization technique is used only when the request probability falls below the calculated Prediction threshold. The main difference to the method proposed in section 2 consists in the location of the Prediction method. To predict Minimization requests, the client side has to foresee potential subsequent data requests of “minimized data”. But what happens when the Prediction was incorrect? Mostly, it does not make sense that the “smart” server side remembers which data has been requested because average values has been sent. So, as before (2), incorrect Prediction leads to an increased network load.

4. REQUEST OPTIMIZATION

The third idea is strongly related to the second one. Not only the number of requested data is of high importance for the improvement of the C/S-communication, but also the type. Former two-tier-architectures followed the paradigm that clients are responsible for working with the data and that the server should provide the “raw stuff”. In comparison with this a “smart” data server prepares the data to optimize the client

REQUEST:
 “IBM” stock data values from 1990 till now.
RESPONSE: (without Minimization)

request. Thus, either the quality of the data on the client side or the network load can be improved [2].

As an instructive example for Request Optimization let us have a look at the following general problem of portfolio management systems and chart applications.

To display the 200 days moving average line of stock values, one needs 200 additional values (usually 2KB) of every share before the first day of interest to calculate the arithmetic average on the client side. This is an enormous overhead especially when the time span is rather short. The server can possibly pre-calculate typical requests and thus gain time and system load. In most cases, the client ignores the problem of additional data needed. A typical result of this kind of client-side calculation of moving average indicators is shown in figure 1:

The deficits on the left side of the figure result from



Certainly, other arguments must also be taken into consideration. In the examples described so far, a generation on the server side is very efficient only for “typical” moving average indicators. 38, 100 or 200 days moving average calculations are very common and the server can generate these values as soon as the stock data values are known. If the client requests a 142 days moving average, though, the server must calculate the data while the client is waiting for a response. Prediction techniques could help here, but the Prediction overhead is usually too high for this usage.

As a summary of the *Request Optimization* methods to improve C/S-communication, we can say that applications should offer means for alternative request methods so that the user can decide whether the “intelligence” - the location of data calculation - should reside on the client side or on the server side. A faster calculation of server



Figure 1: Difference between client side (left) and server side calculation of moving average indicators (right)

client side calculation restrictions. If the server only sends stock data values from a certain time span, the client is not able to generate all according moving average values. The right side shows a calculation of the moving average indicators on the server side. There is enough data to generate all indicators. Of course, the client could have requested stock data values before the first date of interest. Then the program would have been able to calculate the indicators itself.

The example described here presents a general problem: which part of a Client/Server application should contain the “intelligence”- the client or the server side? There are a number of advantages and disadvantages for both concepts. A principle idea in cases like the ones described in the example is to let the user decide. Request Optimization is solved best - other than Prediction or Minimization - when considering human bias. The problem can already be considered solved if the user has the possibility to choose whether deficits as shown in figure 1 bother him/her or not. There should be an *optional* way to retrieve indicators from the server side. The standard procedure may be the fast and efficient generation of indicators on the client side.

responds is often paid with a higher degree of data “inconvenience” (e.g. deficits like those in figure 1).

5. IMPLEMENTATION FOR THE SDS

In sections 2-4, we described and explained general features to improve C/S-communication. In this section, we will highlight some of the most interesting points in implementing modules for the SDS to fulfill the requirements described so far.

5.1 Prediction

A critical point of the Prediction algorithm and formula 1 is the huge demand for storage. We will discuss an alternative implementation of the quadratic, symmetric Memory Matrix ϕ . Undoubtedly, it would be superfluous to store identical entries in a symmetric matrix twice. Only the main diagonal and the upper triangle of ϕ are stored. Thus, we do not need to store $A \cdot A$ values, but only $1+2+3+...+A = (A \cdot A + A) / 2$.

The conversion from the 2-tupel indices to a linear array **MM** can be achieved by using the following function f :

$$f(i, j) = (\min(i, j) - 1) \cdot A + \max(i, j) \quad (5)$$

Thus, every formula of section 2 has to consider function (5) to retrieve the correct entry of φ .

$$\varphi_{ij} = \mathbf{MM}[f(i, j)] \quad (6)$$

With this modeling, the storage-consuming Memory Matrix φ can be implemented as a more efficient, linear array **MM**.

5.2 Minimization

As we explained in section 3, it is not sufficient to send a minimum of data that can be resolved by the client. Sometimes, meta-data is also needed, for instance the high- and low-values of stock data during a certain time span. This information must be send from the server to the client even if the corresponding values are not "resolvable". The SDS solves this problem by providing additional meta-information that can be requested by the client.

The following example 2 shows a possible request of a client to a Smart Data Server (further details of the communication between SDS and its clients can be found in [1]). The keyword "GRANULARITY" results from the resolution preferred by the clients.

REQUEST:

```
<FUNCTION RATE>
  <PARAM SYMBOL> SAP
  <PARAM DATEFROM> 1990-01-01
  <PARAM DATETO> 1999-12-31
  <PARAM GRANULARITY> 800
  <RESULT TABLE RATEDATA>
    <RESULT VALUE DATE>
    <RESULT VALUE CASHPRICE>
  </RESULT TABLE>
  <RESULT VALUE PRICEMIN>
  <RESULT VALUE PRICEMAX>
  <RESULT VALUE PRICEMINDATE>
  <RESULT VALUE PRICEMAXDATE>
</FUNCTION>
```

Example 2: Client request with meta-data and granularity

The idea is that the client codes its preferred resolution (here: granularity) into the request. Aside from the main information (stock data values for "SAP"), some meta-data is also requested (the highest and lowest values during the time span). According to the granularity, only 800 values are returned.

As explained in section 3, it is not always appropriate to request granulated data, because the user might want to "zoom" into the returned time span. In this case, additional requests would be necessary, even though a

complete, non-granulated request would have satisfied further needs before.

This problem must be solved on the client side. A Prediction algorithm similar to [16] can help to determine whether the user will make further efforts on the requested data. In case of doubt, the granularity feature should not be used.

5.3 Request Optimization

Request Optimization is a method to improve C/S-communication that depends highly on the type and structure of the transferred data. The implementation as module of the SDS is therefore explained here only as part of a stock-data services application.

To be able to fulfill very different client requirements, the SDS provides several functions with optional features. For instance, Moving Average trend indicators, a Relative Strength Index (RSI), a Moving Average Convergence/Divergence system (MACD) and Bollinger Bands can all be calculated on the server side or on the client side respectively. The SDS has to provide all these functions if the client is not able to generate them itself.

Unfortunately, the functions mentioned above can have parameters that differ from standard settings. A typical MACD request works on former stock data values of the last 12/26/9 days. Sometimes, though, other settings also make sense. The SDS can not pre-calculate trend indicators with non-standard parameters. Even though it might be possible - with means of Prediction - to foresee some very special user requests, the current implementation of the corresponding SDS module pre-calculates all trend indicators only for standard parameters. Experience shows that these kinds of user requirements are rather rare and a potential predictive threshold would be very high. In future extensions of SDS-modules, the parameters of trend indicators are possibly also predicted.

6. CONCLUSION

The presentation of the optimization features in this paper leads directly to a hybrid approach. The advantages of *Prediction* should help to overcome the disadvantages of *Minimization* and vice versa. Additionally, *Request Optimization* can improve the efficiency of overall C/S-communication.

The idea of Prediction is to guess future user-wishes on the basis of former requests. User-perceived latency can be reduced by predicting future requests and pre-calculating or even pre-fetching data. Problems occurring for Prediction are book-keeping costs and increasing network or system load for wrongly predicted requests. Furthermore, the efficiency of Prediction depends not only on the data itself but also on human bias: how constant/predictable are the requests?

Especially for cases where prediction fails, Minimization can help to improve C/S-communication. The principle of Minimization is to reduce the volume of data to be sent from the server to the client on the basis of hardware or software restrictions (e.g. display resolution). Even though this is a very simple principle, care should be taken while requesting data: meta-data (e.g. boundary values) must be requested explicitly and if a user wants to work on the retrieved data (e.g. “zooming”), additional requests might become necessary. Prediction on the client side can help to foresee whether Minimization for a concrete data request makes sense or not.

Request Optimization is a very subtle method to increase the efficiency, performance and quality of C/S-applications. An approach based on strictly defined data types, depending on the functional requirements of the application, allows alternative ways to generate data. In some cases, it might be wiser to generate data on the client side, based on minimal information from the server side; in other cases, the resulting inconveniences are not acceptable so that high-quality data should be provided by the “smart” server side. On the whole, this question should be decided directly by the user.

Our current work focuses on improving the described approaches. The Prediction algorithm, for instance, is designed to model time and document aging more adequately. The Minimization process is supposed to “recognize” if important boundary data is not part of the response granularity set and should, thus, try to improve the server response. The Request Optimization idea has to be extended to other functions, e.g. trend indicators or stock-data signals .

7. REFERENCES

- [1] U. Roth, E.-G. Haffner, T. Engel, Ch. Meinel, “An Approach to Distributed Functionality: The Smart Data Server (SDS)”, in: *Proceedings of the WebNet International Conference 1999*
- [2] U. Roth, E.-G. Haffner, T. Engel, Ch. Meinel, “The Smart Data Server - A New Kind of Middle Tier”, in: *Proceedings of the IASTED International Conference Internet and Multimedia Systems and Applications (IMSA '99)*, 1999
- [3] <http://www.javasoft.com>
- [4] <http://www.horb.org>
- [5] <http://www.swift.com>
- [6] <http://www.omg.org> / <http://www.corba.org>
- [7] Brown, Nat (1997). *Distributed Component Object Model Protocol -- DCOM/1.0*, Microsoft Corporation <http://msdn.microsoft.com/library/specs/>
- [8] Sun Microsystems (1998). *Java Remote Method Invocation Specification*, Revision 1.50, JDK 1.2, Sun Microsystems <ftp://ftp.javasoft.com/docs/jdk1.2/rmi-spec-JDK1.2.pdf>
- [9] Azer Bestavros, Using Speculation to Reduce Server Load and Service Time on the WWW, *Proceedings of CIKM95*, November 1995
- [10] Achim Kraiss, Gerhard Weikum, Vertical Data Migration in Large Near-Line Document Archives Based on Markov-Chain Predictions, *Proceedings of the 23rd VLDB Conference*, 1997
- [11] Achim Kraiss, Gerhard Weikum, Integrated document caching and prefetching in storage hierarchies based on Markov-chain predictions, *The VLDB Journal*, Springer-Verlag, 1998
- [12] Schechter, Krishnan, Smith, Using path profiles to predict HTTP requests, *Proceedings of the World Wide Web Conference*, 1998
- [13] Mark Crovella, Paul Barford, „The Network Effects of Prefetching”, *IEEE Infocom* 1998
- [14] Ramón Cáceres, Fred Douglass, Anja Feldmann, Gideon Glass, Michael Rabinovich, Web Proxy Caching: The devil is in the details, *Workshop on Internet Server Performance*, Madison, WI, June 1998
- [15] Zhimei Jiang, Leonard Kleinrock, An Adaptive Network Prefetch Scheme, *IEEE J Sel Areas Commun*, 1998
- [16] E.-G. Haffner, U. Roth, T. Engel, Ch. Meinel, “A Semi-Random Prediction Scenario for User Requests”, *Proceedings of the Asia-Pacific International Web-Conference 1999*, <http://www2.comp.polyu.edu.hk/~apweb99/cgi-bin/publish.cgi>
- [17] Stephen Erickson, Danying Yi, Modelling the performance of a Large Multi-Tiered Application, *American Management Systems*, AMS Center For Advanced Technology, 1999
- [18] Dickmann, A. „Two-Tier Versus Three-Tier“, *Apps. Informationweek* 533, 13/95, 1995, 74-80