

# Modeling “Just-in-Time” Communication in Distributed Real-Time Multimedia Applications

R. Yang<sup>\*†</sup>, R.D. van der Mei<sup>\*†</sup>, D. Roubos<sup>\*</sup>, F.J. Seinstra<sup>\*</sup>, G.M. Koole<sup>\*</sup>, and H.E. Bal<sup>\*</sup>

<sup>\*</sup>Vrije Universiteit Amsterdam, Faculty of Sciences  
De Boelelaan 1081a, 1081 HV, Amsterdam, The Netherlands  
Email: ryang@few.vu.nl

<sup>†</sup>Centre for Mathematics and Computer Science  
Kruislaan 413, 1098 SJ, Amsterdam, The Netherlands

**Abstract**—The research area of **Multimedia Content Analysis (MMCA)** considers all aspects of the automated extraction of new knowledge from large multimedia data streams and archives. In recent years, there has been a tremendous growth (in data and computational demands) in the MMCA domain, and this growth is likely to continue in the near future. Multimedia applications operating in real-time environments must run under very strict time constraints, e.g., to analyze video frames at the same rate as a camera produces them. To adhere to such constraints, large-scale multimedia applications typically are being executed on Grid systems consisting of large collections of compute clusters.

In services-based scenarios, where video content analysis is being performed by a set of remote multimedia servers, results on a particular video frame are obtained quickest if a server is unoccupied (i.e., not working on previously submitted frames). Keeping a server unoccupied, however, is a waste of available compute resources. Therefore, it is important to tune the transmission of newly generated video frames to the occupation of remote servers. However, due to variations in transmission latencies, it is difficult to accurately tune the sending of video frames such that resource utilization is optimized. In this paper we refer to this issue as the problem of “just-in-time” communication.

In this paper we address this issue by introducing an adaptive control method that reacts to the continuously changing circumstances in Grid systems so as to obtain the highest service utilization possible, and to minimize service response time for individual video frames. Extensive experimental validation on a real distributed system, in combination with a trace-driven simulation, show that our control method indeed is highly effective.

## I. INTRODUCTION

Today, multimedia data is rapidly gaining importance along with recent deployment of publicly accessible digital television archives, surveillance cameras in public locations, and automatic comparison of forensic video evidence [11]. In a few years, computerized access to the content of multimedia data will be a problem of phenomenal proportions, as digital video may produce high data rates, and multimedia archives steadily run into petabytes ( $10^{15}$ ) of storage space. As individual compute clusters can not satisfy the increasing computational demands, distributed supercomputing on large collections of clusters (Grids) is rapidly becoming indispensable.

Applications in Multimedia Content Analysis (MMCA) often must run under strict time constraints. Consider, for example, the automatic detection of suspect behavior in video

data obtained from surveillance cameras. In a services-based distributed execution scenario, a client program (typically a local desktop computer) connects to one or more remote *multimedia servers*, each running on a (different) compute cluster. At application run-time, the client application sends video frames captured by a camera to the server, which performs the analysis in a data parallel manner.

Analysis results for a video frame are obtained in the fastest possible way if the employed multimedia server is unoccupied (i.e., not working on previously submitted video data). Keeping a server unoccupied, however, is a waste of available compute resources. To optimize resource utilization, it is essential to tune the transmission of video frames to the occupation of remote multimedia servers. However, due to variations in transmission latencies and other variabilities in the computing environment (e.g., CPU power, memory, I/O), it is difficult to accurately tune the sending of video frames to the response time of a multimedia server. In this paper we refer to this issue as the problem of “just-in-time” communication.

The “just-in-time” communication problem requires prediction methods that react to the continuously changing circumstances in Grid systems. Specifically, given a fixed amount of computing capacity, it is essential to tune the sending of video frames to a multimedia server such that resource utilization is optimized, and server response time is minimized. An immediate consequence of a “just-in-time” communication approach is that a multimedia server always analyzes most recently generated (“up-to-date”) video frames; no server response delays are introduced due to frame buffering at either the client side or at the server. Clearly, this is an important, even critical, requirement in real-time execution.

In this paper we argue that existing prediction methods are not capable of adhering to the specific requirements of just-in-time communication. One problem of existing methods is that *random peaks* in obtained performance cause accumulative errors in the predictions, resulting in significant deviations from the optimal rhythm in the transmission of frames. Another problem is that existing methods can not deal with *periodic peaks* very well either. In this paper, we propose two policies to amend these particular problems. The first, referred to as the *one-before-last-measurement* (BLM) policy, is to restore the

rhythm of transmission by removing the extra delay observed at an earlier moment. The second, referred to as the *peak-prediction* (PP) policy, is to find the periodic characteristics of the peaks in processing times and then to predict occurrence of subsequent peaks. Our proposed prediction method, including the BLM and PP policies, provides a good solution for our just-in-time communication problem. Specifically, we observe that, in comparison to traditional methods, our method improves server utilization from 85% to 98%, and reduces the average waiting time per frame by 250%.

The remainder of this paper is organized as follows. In Section II we present related work, and indicate required improvements. Section III presents the experimental setup, and describes one example application. In Section IV our new prediction method is formulated. Section V discussed our experimental results. Finally, in Section VI we present our conclusions and address topics for further research.

## II. OBTAINING JUST-IN-TIME COMMUNICATION

In this section we first explore potential approaches and existing prediction methods that may be applied to solve the just-in-time communication problem. We discuss the drawbacks of each approach, and indicate a set of required policies that need to be put in place to obtain a truly effective solution.

A simple execution approach, which we refer to as the back-to-back method (BBM), is to perform the sending of a newly generated video frame exactly after a result has been received from the same server (see Figure 1). Using the BBM method, any video frame processed by a multimedia server is guaranteed to be most up-to-date. A drawback of BBM, however, is that the server is idle when it has processed a frame and is waiting for the next one. In a bottleneck situation, the video frame transmission time from the client to the server ( $T_{c1}$ ) and the time to send a result back ( $T_{c2}$ ) may be long. For simplicity, we assume  $T_{c1} = T_{c2} = T_c$ . Then, the service utilization ( $SU$ ) using BBM is given by

$$SU = \frac{T_s}{T_s + 2 \cdot T_c},$$

where  $T_s$  is denoted as the service processing time of a video frame. Obviously, if the communication time increases, service utilization decreases.

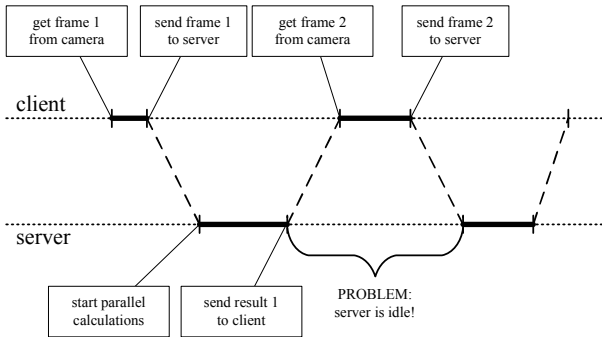


Figure 1. BBM approach to video frame transmission.

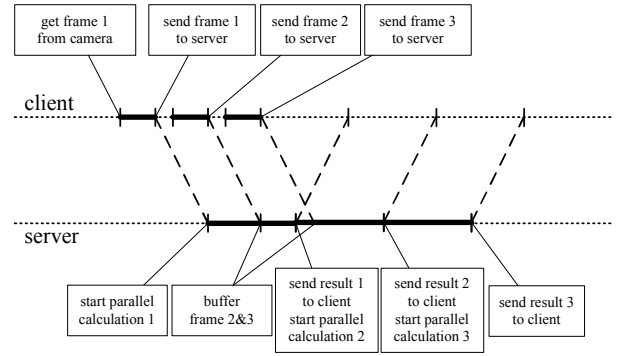


Figure 2. BSM approach to video frame transmission.

An alternative approach, referred to as the buffer storage method (BSM), is to establish a buffer at the server side. As long as the buffer is not full, the client is allowed to keep sending frames to the server. When the server is busy, the frames will be stored in the buffer before being processed (see Figure 2). Using BSM, service utilization can reach 100%. However, the drawback is that the data in the buffer may have become outdated *before* the actual video content analysis even takes place, due to the long waiting time. A solution would be to simply remove outdated frames at the server side. This, however, leads to (a lot of) unnecessary traffic between client and server, which should be avoided as resources are scarce.

Given the previous two methods, the optimal strategy would be to send each  $(i+1)$ -th frame with a delay after sending the  $i$ -th frame. The delay is exactly the processing time of the  $i$ -th frame. For instance, if the service processing time of the current frame equals  $T_{s_i}$ , sending the next frame after a period of  $T_{s_i}$  will give an optimal solution. With this strategy, the server gets the most up-to-date frame and the service utilization is unity (see Figure 3). Unfortunately,  $T_{s_i}$  is unknown before the result of the current frame is returned back to the client side. It is therefore essential to have an accurate prediction of the processing time of video frame data.

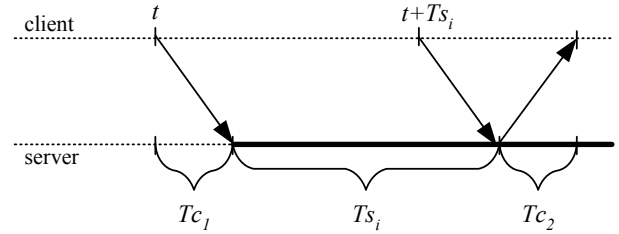


Figure 3. An optimal solution for video frame transmission.

We have observed that existing predictive methods (i.e., the adapted mean-based method [13], the adapted median-based method [13], exponential smoothing [1], [2], [6], [12], and the Robbins-Monro Stochastic Approximation method [7]), are all capable of generating an accurate trend line based on the processing time of previous frames. However, for our just-in-time communication problem, these methods are not sufficiently optimized for particular cases. The first problem

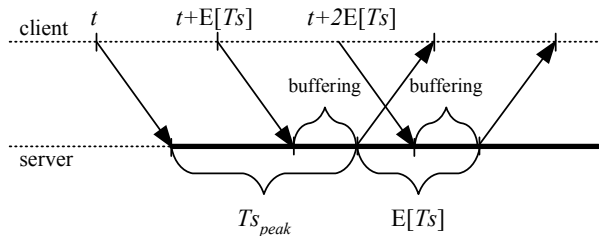


Figure 4. All frames are affected continuously by sudden long process times.

appears, when the processing time of certain frames suddenly become much longer (e.g., a peak) than the expected  $T_s$  obtained from a trend line. The sudden change breaks the rhythm of frame transmission and causes accumulative waiting times for all subsequent frames, even when the processing time returns back to the expected  $T_s$  (see Figure 4).

Apart from random peaks, a second problem is that one can observe processing times to have periodic peaks. If the service processing time of frame  $i$  is predicted as a peak, then the sending of frame  $(i + 1)$  should be delayed to prevent a long buffering time. None of the prediction models mentioned above can deal with random peaks very well, nor do these pay attention to periodic characteristics.

In the remainder we propose two policies to amend these particular problems. The first, referred to as the *one-before-last-measurement* (BLM) policy, is to restore the rhythm of transmission by removing the extra delay observed at an earlier moment. The second, referred to as the *peak-prediction* (PP) policy, is to find the periodic characteristics of the peaks in processing times and then to predict occurrence of subsequent peaks. In addition, we will show that our proposed prediction method, including the BLM and PP policies, provides a good solution for the just-in-time communication problem.

### III. EXPERIMENTAL SETUP

In a Grid environment, resources have different capacities and many fluctuations exist in load and performance of geographically distributed nodes [3]. As the availability of resources and their load continuously vary with time, the repeatability of the experimental results is hard to guarantee under different scenarios in a real Grid environment. Also, the experimental results are very hard to collect and to observe. Hence, it is wise to perform experiments on a testbed that contains the key characteristics of a Grid environment on the one hand, and that can be managed easily on the other hand. To meet these requirements, we perform all of our experiments on the recently installed DAS-3 (the Distributed ASCI Supercomputer 3) Grid test bed [8].

DAS-3, see Figure 5, is a five-cluster wide-area distributed system, with individual clusters located at four different universities in The Netherlands: Vrije Universiteit Amsterdam (VU), Leiden University (LU), University of Amsterdam (UvA), and Delft University of Technology (TUD). The MultimediaN Consortium (UvA-MN) also participates with one cluster.

#### A. Example application

In our experiments, we use the DAS-3 system to run a real-time multimedia application, referred to as ‘‘Aibo’’. The ‘‘Aibo’’ application demonstrates real-time object recognition performed by a Sony Aibo robot dog [10]. Irrespective of the application of a robot, the general problem of object recognition is to determine which, if any, of a given repository of objects, appears in an image or video stream. It is a computationally demanding problem that involves a non-trivial trade-off between specificity of recognition (e.g., discrimination between different faces) and invariance (e.g., to shadows, or to differently colored light sources). Due to the rapid increase in the size of multimedia repositories of ‘known’ objects [4], state-of-the-art sequential computers no longer can live up to the computational demands, making high-performance computing (potentially at a world-wide scale, see also [10]) indispensable.

The application has been implemented using the Parallel-Horus software architecture, that allows programmers to write parallel and distributed multimedia applications in a fully sequential manner [10]. The automatic parallelization and distribution of the application results in services-based execution: a client program (typically a local desktop machine) connects to one or more *multimedia servers*, each running on a (different) compute cluster. Each multimedia server is executing in a fully data parallel manner, thus resulting in (transparent) *task parallel execution of data parallel services*.

More specifically, in the application, before any processing takes place, a connection is established between the client application and a multimedia server. As long as the connection is available, the client can send video frames to this server.

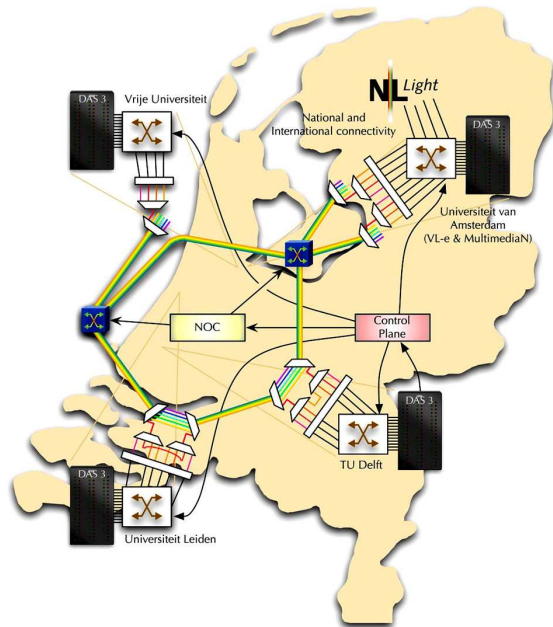


Figure 5. The Distributed ASCI Supercomputer 3.

Every received video frame is scattered by this server into many pieces over the available compute nodes. Normally, each compute node receives one partial video frame for processing. The computations at all compute nodes take place in parallel. When the computations are completed, the partial results are gathered by the communication server again and the final result is returned to the client. In this paper, the time to process a single video frame in this manner is defined as the *service processing time*  $Ts_i$ . The individual values of  $Ts_i$  are collected as data source for a trace-driven simulation. In our simulation, the service utilization and total waiting times are calculated by using different prediction models in combination with our BLM and PP policies.

#### IV. METHOD FORMULATION

This section describes our newly proposed modeling approach in detail. The approach is based on the results of extensive experimentations performed on DAS-3 (see Section III). Our real-time multimedia ‘‘Aibo’’ application is run to generate data that are used in our trace-driven simulation for validating the final model. The notations used in this paper are defined as follows.

- $Ts_i$ : the processing time of the  $i$ -th frame.
- $Tc_i$ : the communication time of sending the  $i$ -th frame from the client to the server.
- $t_i$ : the time point when the client sends the  $i$ -th frame to the server.
- $r_i$ : the time point when the client receives  $i$ -th frame from the server.

##### A. Preliminary

1) *Trend line*: As shown in Figure 3, if we can predict the service processing time of the current frame accurately, then sending the next frame after the predicted time unit should provide an optimal solution. Therefore we investigated several conventional prediction methods (i.e., adapted mean-based methods, adapted median-based methods, exponential smoothing methods, and Robbins-Monro Stochastic Approximation methods) for predicting the service processing time. We found that, based on the earlier service processing times, and by using any of these prediction models, an accurate trend line can be generated. Figure 6 gives an illustration of the predicted service processing time versus the measured value of running the Aibo application using one compute node, using a single CPU only.

2) *Periodicity of the peaks*: Another important observation from our experimental results is the occurrence of periodic peaks using large numbers of compute nodes. Because our multimedia application is partially implemented in Java, the *Java garbage collector* [9] has an influence on the service processing time. In case of large service processing times, the effect of garbage collection generally is insignificant and can be ignored. This is the situation as depicted in Figure 6. In contrast, when the service processing time is small compared to the garbage collection time, the periodic peaks are significant.

We ran the Aibo application using 64 compute nodes (using one CPU per node) on three different moments in time. From these data sets, we notice that there is a deterministic period of the occurrences of certain specific peaks (see Figure 7).

##### B. Method

Based on the experimental results, we conclude that an effective prediction method for our application must have the following characteristics: (1) it must be able to generate an accurate trend line of the service processing time, (2) it should be able to deal with outliers in the observed processing time as soon as possible, and (3) it must be able to predict when the next peak occurs. In this section, we discuss the applied prediction models and our BLM and PP policies in detail.

1) *Prediction models*: Among existing predictive methods there is a huge difference in the way previously obtained data are handled. In some cases one wants to adapt very quickly to observed changes in the data, while there are also cases in which this behavior is not desired. The adapted mean-based method [13] uses arithmetic averages over some portion of the measurement history to predict the next measurement. In particular, the extent of the history taken into account depends on a parameter  $K$ , specifying the number of previous measurements for the arithmetic average. The parameter  $K$  is changed by  $-1$ ,  $0$ , or  $+1$  over time based on the prediction error. In our experiments, the initial value of  $K$  is set to 20.

Adapted median-based methods [13] use a portion of the measurement history defined by the parameter  $K$  to calculate the median which is used for the prediction. The parameter  $K$  is adapted in the same way as in the mean-based method above. Note that the prediction of this method is not influenced much by asymmetric outliers (e.g., a peak in the processing time), since this does not affect the median greatly.

In exponential smoothing [1], [2], [6], [12] earlier measurements are not weighted equally as in the case of a mean-based method, but with exponentially decreasing weights as the measurements get older. More specifically, denote by  $w(i)$  the weight for the  $i$ -th previous measurement. Then,  $w$  is the following function

$$w(i) = \alpha(1 - \alpha)^i,$$

with  $\alpha$  a parameter determining the rate of decay of the function. In our experiments, we set  $\alpha = 0.5$ . As in the previous methods, the parameter  $K$  determines the number of earlier measurements that we intend to use. In case  $K > \{\# \text{ available previous measurements}\}$  and in case  $K < \infty$  we made sure, by scaling of the weights, that the sum of the weights used sum up to 1.

The Robbins-Monro approximation method [7] is a stochastic approximation method. If we denote by  $\hat{T}s_i$  the estimation of the  $i$ -th processing time, then the estimation is updated according to the following relation

$$\hat{T}s_{i+1} = \hat{T}s_i + \varepsilon_i(Ts_i - \hat{T}s_i),$$

where  $\varepsilon_i$  is a parameter possibly depending on  $i$ . The intuition behind the update rule is the following. In case the observed

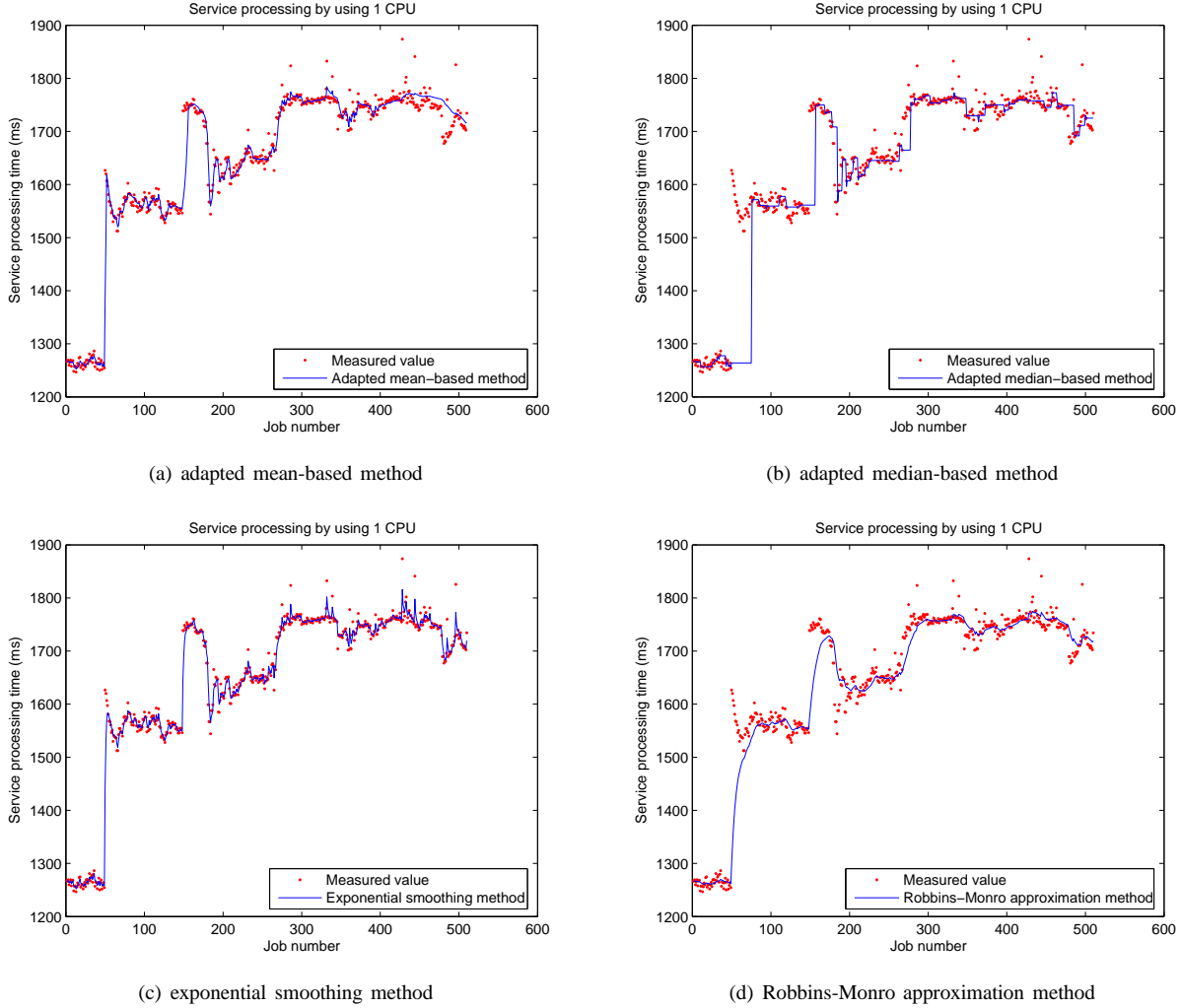


Figure 6. Trend line generated by different prediction models.

processing time is higher than estimated, the prediction for the next processing time is increased by a small amount of the difference, and vice versa. When  $\varepsilon_i = 1$  for all  $i$ , then the prediction for the next processing time is equal to the last observation. We set  $\varepsilon_i = 0.5$  for our experiments.

2) *BLM Policy*: Our first policy to deal with peaks is called “one-before-last-measurement” (BLM) policy. This policy follows the following steps.

(a) The  $i$ -th job will not be sent until the result of the  $(i-k)$ -th job becomes available to the client. Because we must take care that the server has enough jobs to process, we can not use the last measurement data as a predictor (also indicated by Harchol-Balter and Downey [5]). Therefore  $k$  must be larger or equal to 2. Throughout this paper, we focus on the case that  $E[T_c] \leq \frac{E[T_s]}{2}$ . In this case, we set  $k = 2$ . This implies that at most one job is waiting in the buffer at the server side. As a result, the occurrence of cumulative waiting times can be prevented. In the case that  $T_c > \frac{E[T_s]}{2}$ , we only need to enlarge the value of  $k$ . Hence, for  $k = 2$ , we have the following

equation,

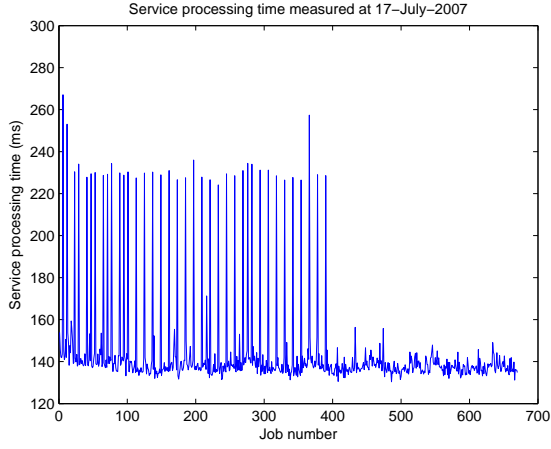
$$t_i \geq r_{i-2}. \quad (1)$$

This equation implies that the  $i$ -th video frame is sent after the result of the  $(i-2)$ -th frame is received by the client. Figure 8 gives an illustration.

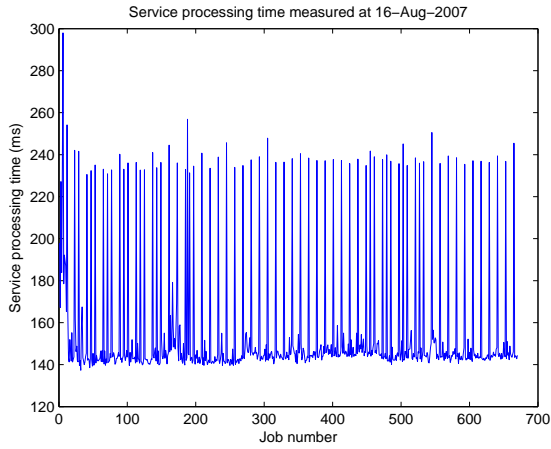
(b) Obviously, if the result of the  $(i-1)$ -th frame is received, the  $i$ -th frame must be sent immediately. Therefore, we have

$$t_i \leq r_{i-1}. \quad (2)$$

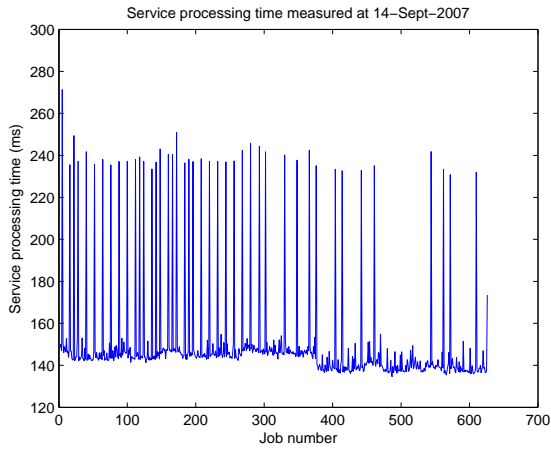
(c) Consider the difference between the send time of the  $(i-1)$ -th frame and the  $(i-2)$ -th frame. Denote the expected service processing time and the communication time as  $E[T_s]$  and  $E[T_c]$ , respectively. If  $T_{s_{i-2}} > E[T_s]$ , then it is optimal to send the  $i$ -th frame at  $r_{i-2} + E[T_s] - 2 \cdot E[T_c]$ . Figure 8(a) gives an example. In case  $T_{s_{i-2}} \leq E[T_s]$ , the optimal sending moment is at  $t_{i-1} + E[T_s]$ . See Figure 8(b). Hence we get



(a) 17-July-2007



(b) 16-August-2007

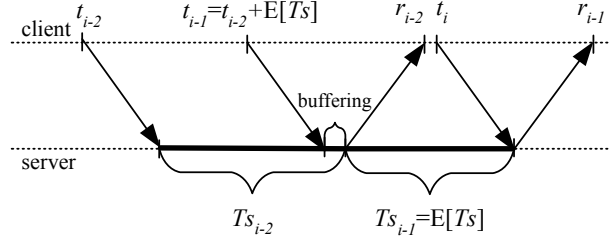


(c) 14-September-2007

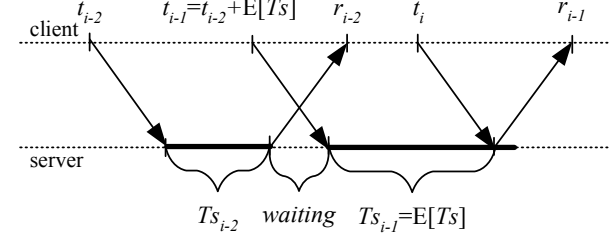
Figure 7. Service processing time taken at different times.

the following equation,

$$t_i = \begin{cases} r_{i-2} + E[Ts] - 2 \cdot E[Tc] & \text{if } t_{i-1} - t_{i-2} < Ts_{i-2}, \\ t_{i-1} + E[Ts] & \text{otherwise.} \end{cases} \quad (3)$$



(a) The optimal sending time in case of  $Ts_{i-2} > E[Ts]$



(b) The optimal sending time in case of  $Ts_{i-2} \leq E[Ts]$

Figure 8. BLM Policy.

Note that using the receiving time of the  $(i-2)$ -th frame to determine the sending time of  $i$ -th frame indirectly takes into account the variation of the communication time between the client and the server. Therefore, the assumption  $Tc_1 = Tc_2$  is not necessary any longer. Combining Equations 1, 2, and 3, the optimal sending time of  $i$ -th frame is given by

$$t_i = \min(r_{i-1}, \max(r_{i-2}, t_{i-1} + E[Ts], r_{i-2} + E[Ts] - 2E[Tc])). \quad (4)$$

3) *PP Policy*: Our second method, called peak-policy, tries to predict the next outlier based on historical observations. We define an outlier (i.e., a peak) as significantly different from the average processing time if the observation is much larger than the average (say 1.2 times larger). Based on the occurrences of peaks in the previous observations, we try to predict when the next peak will occur. Motivated by experiments, we observe that there is a deterministic period of the occurrences of peaks. See Figures 7(a), 7(b), and 7(c) for the experimental results. Denote  $P = \{i | Ts_i \text{ is peak}\}$  as the set of peaks and denote by  $p_j$  the  $j$ -th element of  $P$ . Let  $k$  be an integer number. If  $p_j - p_{j-1} = \dots = p_{j-(k+1)} - p_{j-k}$  then we say that there is a deterministic period of length  $d = p_j - p_{j-1}$ , and we expect the next peak to occur at job number  $j + d$ . Note that  $k$  defines the number of previous peaks that should have occurred equidistantly with length  $d$  such that we consider the peaks as periodical events. The optimal  $k$  is not known beforehand. Therefore, we will start with an arbitrary value and adjust it as time evolves. Suppose that  $k = 3$ , and we observe three peaks each having distance  $d$ , then the method predicts that the next peak occurs after processing of  $d$  frames. If it turns out that the prediction is wrong, then we increase  $k$  by 1, since probably  $k = 3$  was too low. In case the prediction is correct, then we decrease  $k$  by 1, such as to try a smaller

number. To prevent meaningless values for  $k$ , we restrict  $k$  to be in  $[3, \infty)$ .

By combining the BLM and PP policies with one of the prediction methods to predict service processing times, we obtain our final method to deal with the just-in-time communication problem in real-time applications.

## V. NUMERICAL RESULTS

In this section we present the results of our experiments performed on the DAS-3 system. The results are also used as the input for a trace-driven simulation in order to validate our final method for determining the exact transmission moments of video frames. In our experiments, the object recognition application is ran on 64 compute nodes using 1 CPU per node.

First, we apply the BBM method (Figure 1) to our Aibo application. In our experiment, we found that the average service processing time ( $E[T_s]$ ) and the average communication time ( $E[T_c]$ ) between client and server amount to 143.629 ms and 11.694 ms, respectively. In this case, the server utilization is about 85%, and the average waiting time per frame is 0. Consider that the service utilization using the BBM method is given by  $E[T_s]/(E[T_s] + 2 \cdot E[T_c])$ . This implies that when  $T_c$  is negligible, the BBM method approaches the optimal strategy. However, in a bottleneck situation where  $E[T_c]$  is long relative to  $E[T_s]$ , the BBM method performs badly.

Server utilization can be increased by sending frames with smaller intervals. However, if a sudden change (a peak) in service processing time takes place, all incoming frames are affected. A particularly difficult situation is when a series of long service times occurs, such that the waiting time of frames increases rapidly due to the accumulation of perceived gaps. In our experiments, we used simulation to evaluate the impact of changing the time interval between sending subsequent frames. The time interval is reduced in 5 steps according to Table I.  $E[T_s]$  and  $E[T_c]$  in Table I are adjusted by one of the prediction models. Since Figure 6 shows that all prediction models are capable of generating accurate trend lines, in this paper, we only choose one of these (i.e. the exponential smoothing method) as a representative prediction model. In Figure 9, it is shown that the average waiting time increases significantly as the service utilization approaches 100%. Hence, the prediction models are not sufficient for our just-in-time communication problem.

Table I  
TIME INTERVAL BETWEEN SENDING TWO SEQUENTIAL FRAMES.

Simulation index	Time interval
1	$T_{sBBM}$
2	$2E[T_c] + E[T_s]$
3	$1.5E[T_c] + E[T_s]$
4	$E[T_c] + E[T_s]$
5	$0.5E[T_c] + E[T_s]$
6	$0.375E[T_c] + E[T_s]$
7	$0.25E[T_c] + E[T_s]$
8	$E[T_s]$

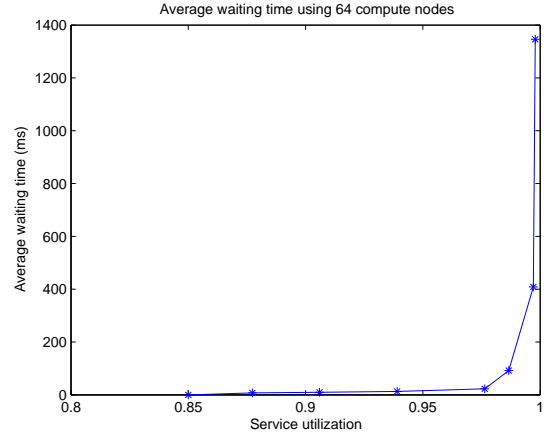


Figure 9. Average waiting time using 64 compute nodes.

In our final model, in which one of the prediction models is combined with the BLM and PP policies, we can achieve high service utilization while keeping the average waiting time low. By using the exponential smoothing method with our policies, we obtain service utilization of about 98%, and an average waiting time per frame of around 7 ms. If we define the waiting time percentage (WP) as

$$WP = \frac{\text{total waiting time}}{\text{total waiting time} + \text{total service processing time}},$$

then we obtain a WP of around 3.5%. Because of the lower value of WP, we can compare the performance of our final method to the BBM method by looking at the service utilization. Define the gain in service utilization  $Gain(SU)$  as follows,

$$Gain(SU) = \frac{\text{service utilization using final model}}{\text{service utilization using BBM model}}. \quad (5)$$

Figure 10 shows the gain of our final method related to the BBM method for different values of  $\frac{T_c}{T_s}$ . In this figure, we notice that the gain in utilization is almost linear in  $\frac{T_c}{T_s}$ . This

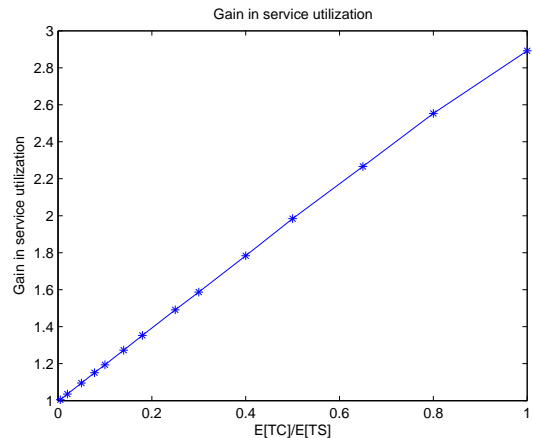


Figure 10. Gain in the service utilization.

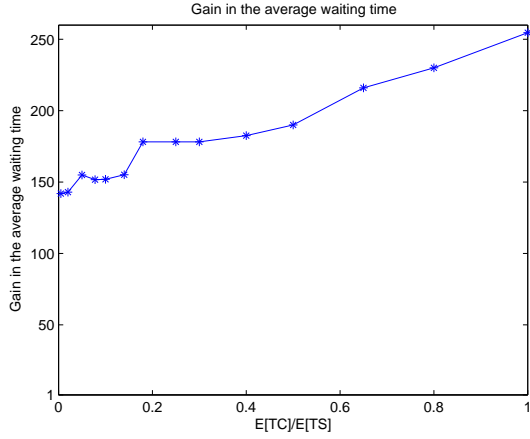


Figure 11. Gain in the average waiting time.

can be explained by the fact that the service utilization in the final model is very close to 1 and the service utilization belonging to the simple strategy can be approximated by  $E[TS]/(E[TS] + 2 \cdot E[TC])$ . Hence, based on Equation 5, we have

$$\text{Gain(SU)} \approx \frac{1}{Ts/(Ts + 2 \cdot Tc)} = 1 + 2 \frac{Tc}{Ts}.$$

Therefore, the gain in the service utilization is nearly increasing linearly with  $Tc/Ts$ .

The last comparison is done to evaluate the benefit brought by our policies. For the prediction method of exponential smoothing, we compare the performance of our final method to the prediction method by looking at the average waiting time. Define the gain in the average waiting time  $\text{Gain}(w)$  as follows,

$$\text{Gain}(w) = \frac{\text{average waiting time using final method}}{\text{average waiting time using the prediction model}}.$$

The results of this comparison are shown in Figure 11. The reason why the final model can gain so much, can be explained by the following example. Assume that during processing, only one peak takes place and that, after that peak, there are still 100 frames to be processed. In this situation the use of prediction models causes all following 100 frames to be delayed by the peak. But using our final model, there is only 1 following frame affected by the peak. Thereafter, the sending times of the next 99 frames are corrected. Thus no error accumulation occurs. Therefore, we conclude that our final model, incorporating BLM and PP, are indispensable and effective for just-in-time communication.

## VI. CONCLUSIONS AND FURTHER RESEARCH

In this paper, we have explored the just-in-time communication problem that requires high service utilization on the one hand, and minimized service response time on the other hand. Using a BBM method, the waiting time is zero. However, service utilization decreases when the communication time between client and server increases. By applying existing

prediction models to this problem, service utilization can be increased. However, at the same time, the average waiting time of video frames increases even faster. This can be explained by the fact that existing prediction models do not pay attention to peaks in the service processing time. For this reason, we have developed two innovative policies, BLM and PP. Using the first policy, cumulative waiting times are avoided by postponing transmission of a new job when a peak is detected. The second policy is used to predict possible peaks. If we can predict the moment when a peak occurs, then we can send new jobs at the right time. Combining these two policies with any of the existing prediction models described in this paper, we achieve our final method to solve the just-in-time communication problem.

Our final method is validated in our experiments. Moreover, we have extensively investigated the gain of our final model related to the BBM model, as well as the prediction methods without incorporating our newly developed policies. From our experimental results we conclude that our final method significantly outperforms the other methods. Specifically, we have observed that, in comparison to other methods, our final method improves server utilization from 85% to 98%, and reduces the average waiting time per frame by 250%.

## REFERENCES

- [1] R. Brown, *Statistical forecasting for inventory control*. McGraw-Hill New York, 1959.
- [2] R. Brown, *Smoothing, Forecasting and Prediction of Discrete Time Series*. Prentice-Hall, 1963.
- [3] M. Dobber, G. Koole, and R. van der Mei, "Dynamic Load Balancing for a Grid Application," in *Proc. International Conference on High Performance Computing (HiPC)*, vol. 1, pp. 342–352, 2004.
- [4] J.M. Geusebroek, G.J. Burghouts, and A.W.M. Smeulders, "The Amsterdam Library of Object Images," *International Journal of Computer Vision*, vol. 61, no. 1, pp. 103–112, 2005.
- [5] M. Harchol-Balter and A. Downey, "Exploiting process lifetime distributions for dynamic load balancing," *ACM Trans. Computer Systems*, vol. 15, no. 3, pp. 253–285, 1997.
- [6] C. Holt, "Forecasting Trends and Seasonals by Exponentially Weighted Moving Averages," *ONR Memorandum*, vol. 52, 1957.
- [7] H. Kushner and G. Yin, *Stochastic Approximation and Recursive Algorithms and Applications*. Springer-Verlag, 2003.
- [8] Online, "http://www.cs.vu.nl/das3/," 2007.
- [9] Online, "http://www.artima.com/underthehood/gc.html," 2007.
- [10] F.J. Seinstra, J.M. Geusebroek, D. Koelma, C. Snoek, M. Worring, and A. Smeulders, "High-Performance Distributed Image and Video Content Analysis with Parallel-Horus," *IEEE Multimedia*, vol. 14, no. 4, pp. 64–75, 2007.
- [11] C.G.M. Snoek, M. Worring, J.M. Geusebroek, D.C. Koelma, F.J. Seinstra, and A.W.M. Smeulders, "The Semantic Pathfinder: Using an Authoring Metaphor for Generic Multimedia Indexing," *IEEE Trans. Pat. Anal. and Mach. Intel.*, vol. 28, no. 10, pp. 1678–1689, 2006.
- [12] P. Winters, "Forecasting Sales by Exponentially Weighted Moving Averages," *Management Science*, vol. 6, no. 3, pp. 324–342, 1960.
- [13] R. Wolski, "Forecasting Network Performance to Support Dynamic Scheduling using the Network Weather Service," in *Proc. Int. Conference on High Performance Computing (HiPC)*, pp. 316–325, 1997.