Performance model for “Just-in-Time” Problem in Real-Time Multimedia Applications

R. Yang[*†], R.D. van der Mei[*†], D. Roubos[*], F.J. Seinstra[*‡], G.M. Koole[*] and H. Bal[*]
[*]Vrije Universiteit Amsterdam, Faculty of Sciences
De Boelelaan 1081a, 1081HV, Amsterdam, The Netherlands
Email: ryang@few.vu.nl
[†]Centre for Mathematics and Computer Science
Kruislaan 413, 1098SJ, Amsterdam, The Netherlands
[‡]University of Amsterdam, Faculty of Science
Kruislaan 403, 1098SJ, Amsterdam, The Netherlands

***Abstract*—Over the last few years, the use of large-scale multimedia data applications has been growing tremendously, and this growth is not likely to slow down in the near future. Many multimedia applications operate in a real-time environment (e.g., surveillance cameras, iris scans), which must meet strict time constraints, i.e. to analyze video frames at the same rate as a camera produces them. To meet this requirement, Grid computing is rapidly becoming indispensable. However, the variabilities of the software and the hardware in grid environment cause the strong burstiness in the transmission delay of video frames. Because the burstiness is unknown beforehand, it is difficult to determine the right sending moments of video frames. If the time interval between sending two sequential frames is too large, then the service utilization may be low. If use large buffer to guarantee the service utilization, then video frames may be out-of-date because of the long waiting time at buffer in the server side. This problem is referred to as “Just-in-time” problem. To solve this problem, it is essential to determine the right sending moments of video frames, properly dealing with the trade-off between the service utilization and the “up-to-date” of video frames.**

Motivated by this, in this paper we develop an adaptive control method that react to the continuously changing circumstances in grid system so as to obtain the highest service utilization on the one hand and to keep the video frame up-to-date on the other hand. Extensive experimental validation in our DAS-3 testbed and the trace-driven simulation show that our method is indeed highly effective.

I. Introduction

Recently, multimedia data is rapidly gaining importance along with the deployment of publicly accessible digital television archives, surveillance cameras in public locations, and automatic comparison of forensic video evidence. In a few years, the digital video may produce high data rates, and multimedia archives steadily run into petabytes (10^{15}) of storage space. To keep the pace with the demand of these applications, the storage space asks an explosive growth to accommodate the data in the form of video and audio. Apart from the huge scale, it is necessary that the data is being automatically processed within a desirable time frame. Recent results from image analysis show that access to the content of large data sets is a hard problem [1]. One way to deal with this is to apply approximation algorithms, at the cost of losing useful details and accuracy. A better solution is to exploit distribution of data and computation over a network over compute nodes. Consider a multi-media application using a surveillance camera. The client sends image frames captured by the camera to the server for content analysis. To meet time constraints, the analyzing have to be done by several nodes in parallel. However, the variabilities in the computation environment (e.g., network characteristics, CPU power, memory, I/O) cause the strong burstiness in service processing time of video frames and the communication time between the client and the server. This raises the need for the development of methods that react to the continuously changing circumstances. In this context, given a fixed amount of computing capacity, it is essential to find an optimal way of sending the frames to the server such that the service utilization is as high as possible on the one hand and the frame processed by the server is kept up-to-date on the other hand. In our paper, this problem is referred to as “just-in-time” problem.

A simple method called back-to-back method (BBM) is to arrange the sending moments to send a new frame exactly after it receives a result from the server. Figure 1 gives an illustration. Using BBM method, video frames processed by the server is most up-to-date. However, the server is idle when it has processed the frame and is waiting for the next frame. In a bottleneck situation, the communication time to send a frame from the client to the server (Tc_1) and the time to send a result back (Tc_2) may be long. For simplicity, we assume $Tc_1 = Tc_2 = Tc$. Then, the service utilization (SU) using BBM method is given by

$$SU = \frac{Ts}{Ts + 2 \cdot Tc},$$

where Ts is denoted as the service processing time of a video frame. Obviously, if the the communication time is long, then service utilization becomes lower.

An alternative approach called 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,

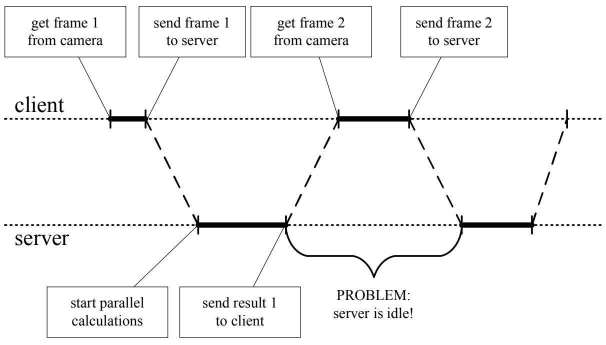


Figure 1. BBM method for arrange sending moments

the service utilization can approach to 100%. However, the drawback is that the data in the buffer may be out-of-date because of the long waiting time before being processed.

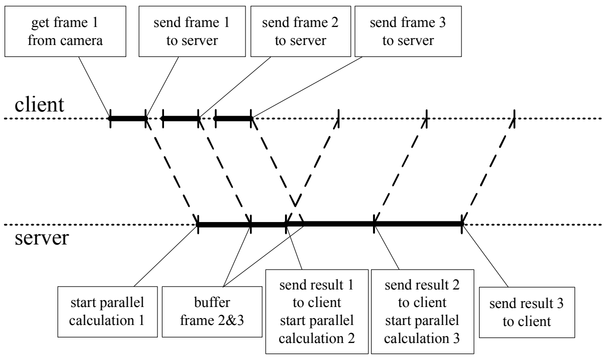


Figure 2. ALT method for arrange sending moments

Based on the discussion of previous two methods, the optimal strategy would be sending 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 is equal to Ts_i, then sending the next frame Ts_i later than when sending the current frame will give an optimal solution. With this strategy, the server gets the most up-to-date frame and the service utilization is unity. This is illustrated in Figure 3. Unfortunately, Ts_i is unknown before the result of the current frame is return to the client side. Hence, a good prediction of the processing time of the frames is desired.

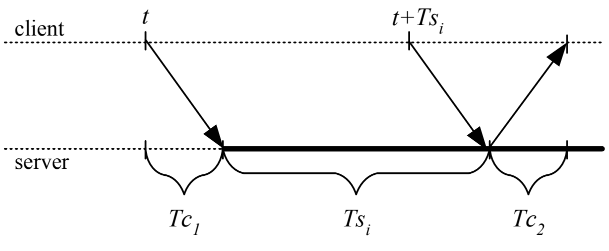


Figure 3. An Optimal solution for sending video frames

In our experimental results (see Section 2 below), we observe that predictive methods: adapted mean-based method [2], adapted median-based method [2], exponential smoothing method [3]–[6] and Robbins-Monro Stochastic Approximation method [7], are all able to generate an accurate trend line based on the processing time of previous frames. However, for just-in-time problem, these methods are not sufficiently optimized considering particular cases. (1) This happens when the processing time of some frames becomes suddenly much longer (e.g. a peak) than the expected Ts obtained by trend line. The sudden changes break the rhythm of sending frames and cause accumulative waiting time for all coming frames. That makes the performance suddenly worse even after the processing time has back to expected Ts. Figure 4 gives an illustration. (2) Besides the random peak, we also observe

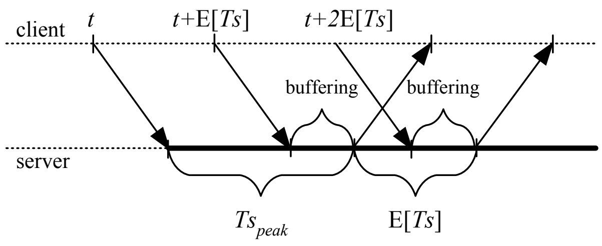


Figure 4. All frames are affected continuously by sudden long process time

the processing time has periodic peaks from our experiment results. If the service processing time of i-th frame is predicted as a peak, then a smart way to send $(i+1)$-th frame is letting the sending time some time later to prevent the long waiting time in the buffer. However, none of the prediction models mentioned above can deal with peaks very well and pay attention to utilize the periodic characteristic of the processing time.

In this paper, based on our experimental observations, we propose two policies to amend the particular situations. The first one, *one-before-last-measurement* (BLM) policy, is to restore the rhythm of sending frames by vanish the extra waiting time after a few frames. The second one, peak-prediction (PP) policy, is to find the periodic characteristic of the peaks in processing times and then to predict when the next peak takes place. A proposed prediction method including BLM and PP policies provides a good solution for just-in-time problem.

The remainder of this paper is organized as follows. In Section II the method is formulated. In Section III the experiments setup and the application are described, and in Section IV the experimental results are discussed. Finally, in Section V we address a number of topics for further research.

II. Method formulation

In this section, the proposed methods are formulated based on two important observations during experiment results. The experiment setup is described in detail in Section III. A real-time multimedia application called "Object Recognition" 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 some conventional prediction methods, 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 out based on the previous service processing time, by using these prediction models, an accurate trend line can be generated. Figure 5 gives an illustration of the predicted service processing time versus the measured value of running object recognition application on 1 compute node with one CPU.

2) Periodic of the peaks: Another important observation from our experimental results is the occurrence of periodic peaks using large number of compute nodes. Because our multimedia application [8] is implemented in Java, the mechanism of *java garbage collection* [9] has influence in the service processing time. In case of large service processing time, the effect of *java garbage collection* can be ignored. Therefore, the periodic peaks are not obvious. The measured data in Figure 5 is right this situation. However, when the service processing time is small compared to the java garbage collection time, the periodic peaks are obvious. We run the object recognition application on 64 compute node with one CPU per node in three different days. From these three different data sets, we notice that there is a deterministic period of the occurrences of some peaks. See Figure 6.

B. Method

Based on the experimental results above, we conclude that a method to obtain a good performance in our real-time application must have the following characteristics: (1) it is 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 can predict when the next peak appears. In this section, we discuss the prediction models and our two policies to deal with peaks in detail.

1) Prediction models: Among the predictive methods there is a huge distinction between them in the way they handle previous data to make the prediction. 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.

Adapted mean-based method uses arithmetic averages over some portion of the measurement history to predict the next measurement. In particular, the amount of history that is 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. See [2] for more details.

Adapted median-based methods, as described in [2], use a portion of the measurement history defined by the parameter K to calculate the median which is used for the prediction. Like the previous method, the parameter K is adapted in the same way as before. 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, the previous 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 α a parameter determining the rate of decay of the function. In our experiments, we set $\alpha = 0.5$. Like in the previous methods, the parameter K determines the number of previous measurements we want 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 one.

The Robbins-Monro approximation method 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

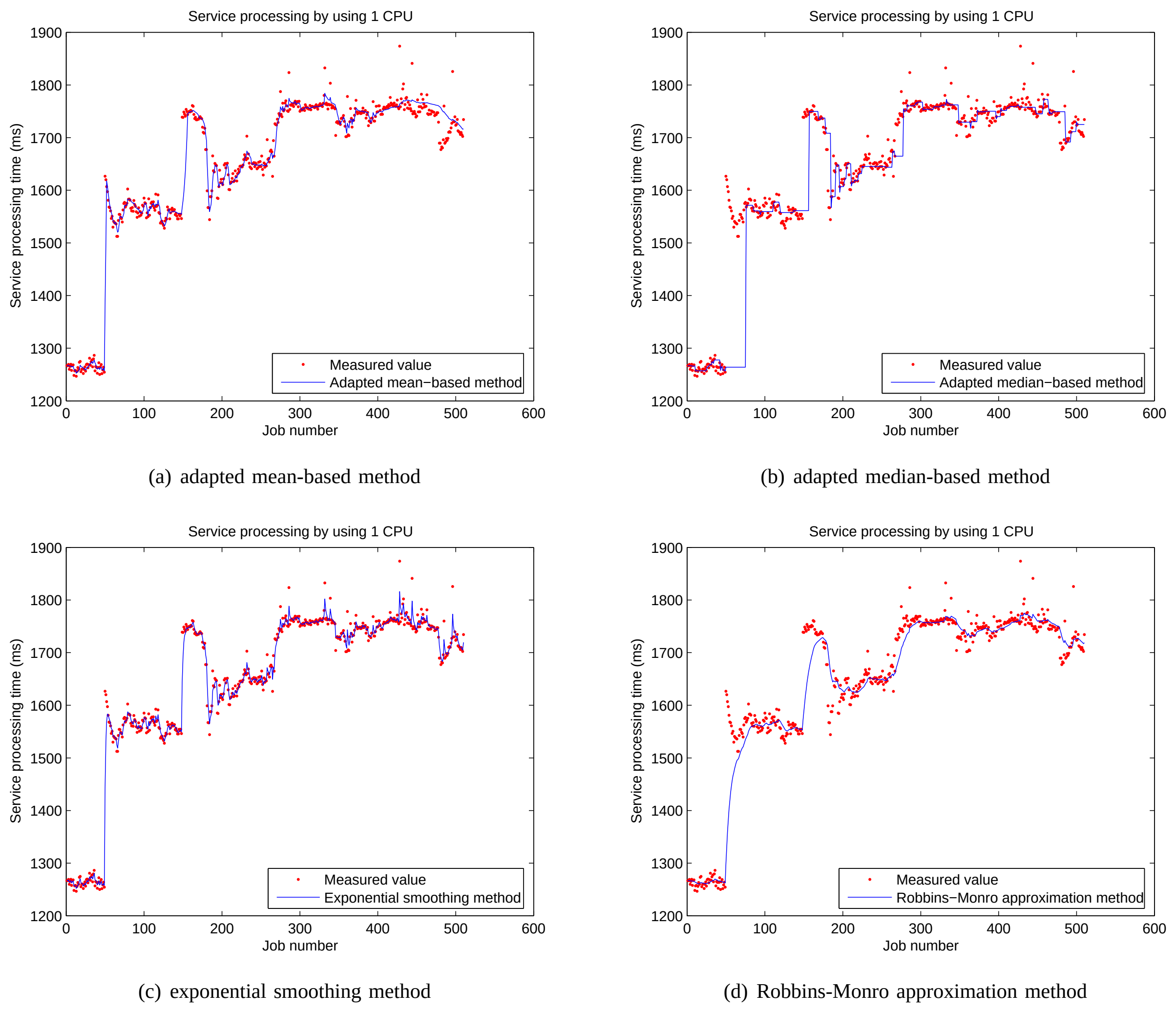
$$\hat{T}s_{i+1} = \hat{T}s_i + \varepsilon_i(Ts_i - \hat{T}s_i),$$

where ε_i is a parameter possibly depending on i. The intuition behind the update rule is the following. In case the observed processing time is larger than the one estimated, the prediction for the next processing time is increased by a small amount of that difference, and vice verse. When $\varepsilon_i = 1$ for all i, then the prediction for the next processing time is equal to last one observed. 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 info obtained by last measurement as a predictor mentioned by Harchol-Balter and Downey [10]. Therefore k must be larger or equal to 2. Throughout this paper, we focus on the case that $E[Tc] \leq \frac{E[Ts]}{2}$. 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 time can be prevented. In the case that $T_c > \frac{E[Ts]}{2}$, we only need to enlarge the value of k. Hence, for $k = 2$, we have the following equation,

$$t_i \geq r_{i-2}. \tag{1}$$



(a) adapted mean-based method

(b) adapted median-based method

(c) exponential smoothing method

(d) Robbins-Monro approximation method

Figure 5. Trend line generated by different prediction models.

This equation implies that i-the video frame is sent after that the result of $(i-2)$-th frame is received by the client. Figure 7 gives an illustration.

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

$$t_i \leq r_{i-1}. \tag{2}$$

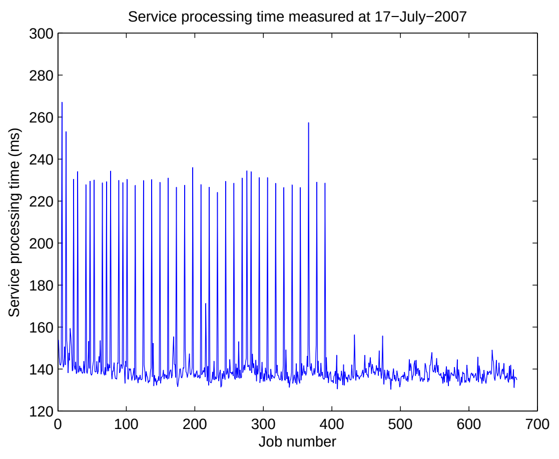
(c) Consider the difference between the send time of $(i-1)$-th frame and $(i-2)$-th frame. Denote the expected service processing time and the communication time as $E[Ts]$ and $E[Tc]$ respectively. If $Ts_{i-2} > E[Ts]$, then it is optimal to send i-th frame at $r_{i-2} + E[Ts] - 2 \times E[Tc]$. Figure 7 gives an example. In case that $Ts_{i-2} \leq E[Ts]$, the optimal sending moment is at $t_{i-1} + E[Ts]$. See Figure 7(b). Hence we get the following equation,

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

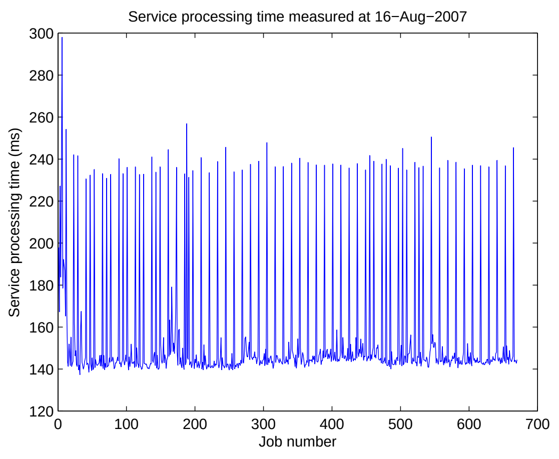
Note using the receiving time of $(i-2)$-th frame to determine the sending time of i-th frame indirectly takes into account the variation of the communication time between the client and the server. Therefore, the assumption $Tc_1 = Tc_2$ is not necessary any more. Combining Equation 1, 2 and 3, the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

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

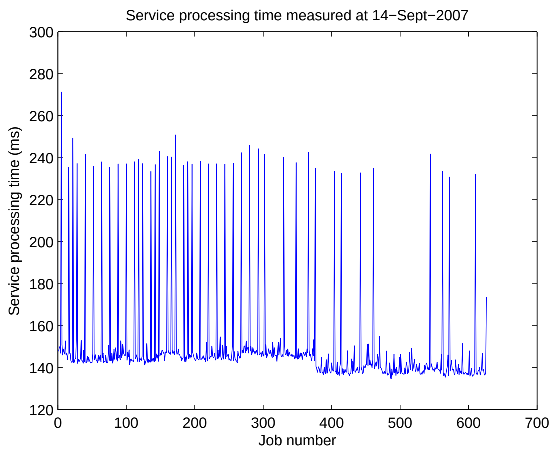
3) PP Policy: Our second method, peak policy, tries to predict the next outlier based on historical observations. We define an outlier, in particular a peak, as significantly different from the average processing time if the observation is much larger than the average (say 1.2 times larger). Based on 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 6(a), 6(b), and 6(c) for the experimental results. Denote by $P = \{i | Ts_i \text{ is peak}\}$ as the set of peaks and denote by p_j the j-th element of P. Let k be an integer number. If $p_j - p_{j-1} = \cdots = p_{j-(k+1)} - p_{j-k}$ 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



(a) 17-July-2007

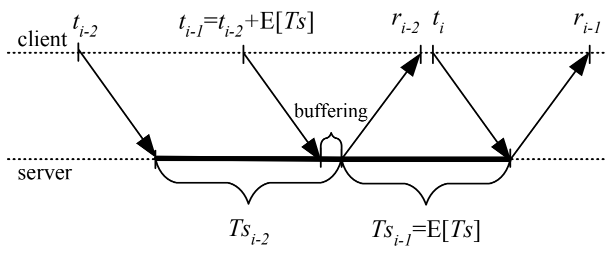


(b) 16-August-2007

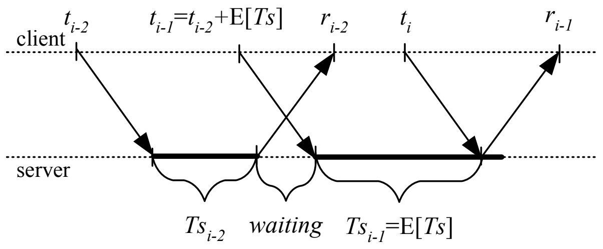


(c) 14-September-2007

Figure 6. Service processing time taken at different times.



(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 7. BLM Policy

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 $[3, \infty)$.

Combining BLM and PP policies with one of the prediction methods to predict service processing time, we achieve the final method to deal with the just-in-time problem in real-time applications.

III. Experimental Setup

In a grid environment, resources have different capacities and many fluctuations exist in load and performance of geographically distributed nodes [11]. As the availability of resources and their load continuously vary with time, the repeatability of the experimental results is hard to be guaranteed 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 our experiments on the test bed that contains the key characteristics of a grid environment on the one hand, and that could be managed easily on the other hand. To meet these requirements, we perform our experiments on DAS-3 Grid test bed [12]. DAS-3 (The Distributed ASCI Supercomputer 3) is a wide-area distributed system designed by the Advanced School for Computing and Imaging (ASCI [13]). It consists of 272 dual AMD Opteron compute nodes. The compute nodes spread out over five clusters located at five locations: VU University Amsterdam (VU), Leiden University (LU), University of Amsterdam (UvA), Delft University of Technology (TUD) and the MultimediaN Consortium (UvA-MN). Unlike its predecessor, DAS-2, DAS-3 is rather heterogeneous in design. Table I provides an overview of all 5 clusters.

In our experiments, we use the VU-cluster to carry out our real-time application called “Object Recognition”. The “Object Recognition” application is implemented in a Robot Dog. It consists of the following operations:

1) An object is held in front of the dog’s camera. The video frames or images are captured by the camera and sent to the servers.
2) The video frames are processed in parallel on the available compute nodes.
3) Based on the key characteristics calculated from the video frames, a database of learned objects is searched.
4) In case of recognition, the dog reacts accordingly.

Before the processing of video frames, the connection between the client (the application) and the communication server (a compute node) is established. As long as the link is connected, the client can send a video frame to this server. The received video frame is scattered by this server into many pieces according to the available compute nodes. Normally, each compute node gets one piece of data segment 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. The time to process a video frame is defined as the service processing time. The individual values of Ts_i are collected as data source for our trace-driven simulation. In our simulation, the service utilization and total waiting times are calculated by using different prediction models in combination with BLM policy and PP policy.

IV. Numerical Results

In this section we present the results of our experiments done in the DAS-3 environment. The results are also used as the input for our trace-driven simulation in order to validate our final method for determining the sending moments of video frames from the client to the server. In our experiments, the object recognition application is run on 64 compute nodes with 1 CPU per node.

First, we apply BBM method (See Figure 1) to our object recognition application for arranging the sending times. In our experiment, it is shown that the average service processing time ($E[Ts]$) and the average communication time ($E[Tc]$) between the client and the server are equal 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 BBM method is given by $E[Ts]/(E[Ts] + 2 \cdot E[Tc])$. That implies that when Tc is negligible, the BBM method approaches the optimal strategy. However, in the bottleneck situation where $E[Tc]$ is long relative to $E[Ts]$, the BBM method will perform bad.

The server utilization can be increased by sending the frames faster after each other. However, if a sudden change (a peak) of service time takes place, all incoming frames are affected. A worse situation is when a series of long service time occur, the waiting time of the frames increase rapidly because the time gaps can be accumulated. In our experiments, we used simulation to evaluate the impact of changing the time interval between sending two sequential frames. The time interval is reduced in 5 steps according to Table II. $E[Ts]$ and $E[Tc]$ in Table II are adjusted by one of the prediction models. Because Figure 5 shows that all prediction models are able to generate accurate trend line. Therefore, in this paper, we only choose one of them: the exponential smoothing method, as a representative prediction model to use. In Figure 8, it is shown that the average waiting time increases enormously as the service utilization approaches 100%. Hence, the predicition models are not suficient for our just-in-time problem.

Table II
Time interval between sending two sequential frames

Simulation index	Time interval
1	Ts_{BBM}
2	$2E[Tc] + E[Ts]$
3	$1.5E[Tc] + E[Ts]$
4	$E[Tc] + E[Ts]$
5	$0.5E[Tc] + E[Ts]$
6	$0.375E[Tc] + E[Ts]$
7	$0.25E[Tc] + E[Ts]$
8	$E[Ts]$

Using our final model (one of prediction models in combination with BLM and PP policies), we can achieve high service utilization on the one hand and keep the average waiting time low on the other hand. By using the exponential smoothing method with our policies, we get the service utilization to about 98%, average waiting time per frame to around 7 *ms*. Define the waiting time percentage (WP) as

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

Then we obtain WP around 3.5%. Because of lower value of WP, we are allowed to compare the performace of our final mehtod to BBM method by looking at the service utilization.

Table I
OVERVIEW OF NODES

Cluster	Nodes	Type	Speed	Memory	Storage	Node HDDs	Network
VU	85 dual	dual-core	2.4 GHz	4 GB	10 TB	85 × 250 GB	Myri-10G and GbE
LU	32 dual	single-core	2.6 GHz	4 GB	10 TB	32 × 400 GB	Myri-10G and GbE
UvA	41 dual	dual-core	2.2 GHz	4 GB	5 TB	41 × 250 GB	Myri-10G and GbE
TUD	68 dual	single-core	2.4 GHz	4 GB	5 TB	68 × 250 GB	GbE (no Myri-10G)
UvA-MN	46 dual	single-core	2.4 GHz	4 GB	3 TB	46 × 1.5 TB	Myri-10G and GbE

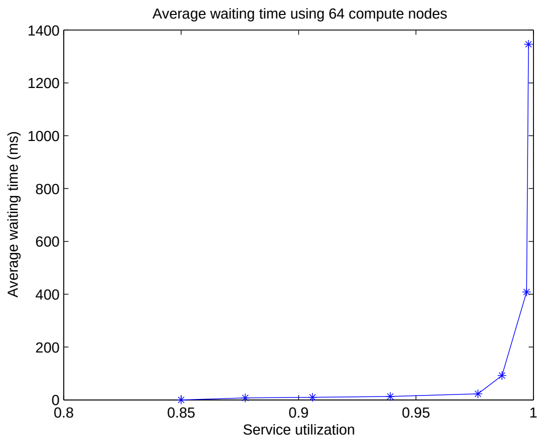


Figure 8. Average waiting time using 64 compute nodes

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 9 shows the gain of our final method related to BBM method for different values of $\frac{Tc}{Ts}$. In this figure, we notice

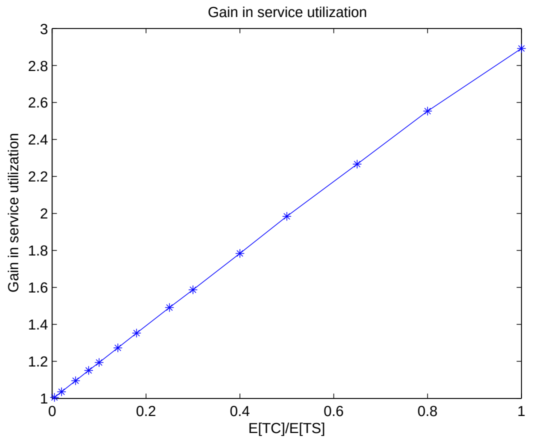


Figure 9. Gain in the service utilization

that the gain in utilization is almost linear in $\frac{Tc}{Ts}$. This 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 simply strategy can be approximated by $E[Ts]/(E[Ts] + 2 \cdot E[Tc])$. Hence, Based on Equation 5, we have

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

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. The prediction method is using the exponential smoothing method. 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 $Gain(w)$ as follows,

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

We get the results shown in Figure 10. The reason why

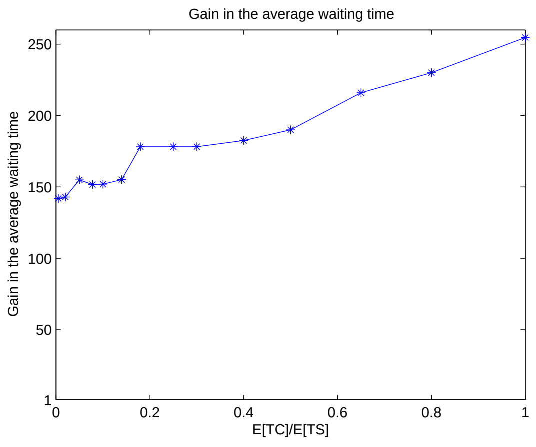


Figure 10. Gain in the average waiting time

the final model can gain so much, can be explained by the following example. Assume during processing, only one peak takes place and after that peak there are still 100 frames need to be processed, then 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. After that, the sending times of the next 99 frames are corrected. Thus no accumulative error happens. Therefore, we conclude that our policies are indispensable and effective for just-in-time problem.

V. Conclusions and further research

In this paper we explored the just-in-time problem that requires the high service utilization on the one hand, and to keep the video frame up-to-date on the other hand. Using BBM method, the waiting time is zero. However, the service utilization will be worse if the communication time between the client and server becomes longer. Applying the prediction models to this problem, the service utilization can be increased. However, at the same time, the increasing of the average waiting time of a frame is even faster. That can be explained by the fact that none of the prediction models pay attention to dealing with the peaks of the service processing time. Therefore we developed two policies, BLM policy and PP policy. Using the first policy, the cumulative waiting time can be avoided by postponing the send time of the new job when a peak is detected. The second policy is used to predict the possible peaks. If we can predict the moment when a peak occurs, then we can manage to send the new job in the right time. Combining these two policies with any of the prediction models, we achieve the final method to solve just-in-time problem.

Our final method is validated in our experiments. We have illustrated the experimental results above. Moreover, we have extensively investigated the gain of our final model related to the BBM model and the prediction methods without using our policies. From our experimental results, it is shown that our final method outperforms those methods.

References

[1] C. G. Snoek, M. Worring, J.-M. Geusebroek, D. C. Koelma, F. J. Seinstra, and A. W. Smeulders, “The semantic pathfinder: Using an authoring metaphor for generic multimedia indexing,” *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 28, no. 10, pp. 1678–1689, 2006.

[2] R. Wolski, “Forecasting network performance to support dynamic scheduling using the network weather service,” *in Proc. International Conference on High Performance Computing (HiPC)*, pp. 316–325, 1997.

[3] R. Brown, *Statistical forecasting for inventory control*. McGraw-Hill New York, 1959.

[4] ——, *Smoothing, Forecasting and Prediction of Discrete Time Series*. Prentice-Hall, 1963.

[5] C. Holt, “Forecasting Trends and Seasonals by Exponentially Weighted Moving Averages,” *ONR Memorandum*, vol. 52, 1957.

[6] P. Winters, “Forecasting Sales by Exponentially Weighted Moving Averages,” *Management Science*, vol. 6, no. 3, pp. 324–342, 1960.

[7] H. Kushner and G. Yin, *Stochastic Approximation and Recursive Algorithms and Applications*. Springer-Verlag, 2003.

[8] F. Seinstra, “User Transparent Parallel Image Processing,” Ph.D. dissertation, University of Amsterdam, the Netherlands, 2003.

[9] online, “http://www.artima.com/underthehood/gc.html,” 2007.

[10] 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.

[11] 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.

[12] online, “http://www.cs.vu.nl/das3/,” 2007.

[13] ——, “http://www.asci.tudelft.nl,” 2007.