Multimed Tools Appl (2012) 59:941–971
DOI 10.1007/s11042-011-0782-5

# Resource optimization in distributed real-time multimedia applications

**Ran Yang · Robert D. van der Mei · Dennis Roubos · Frank J. Seinstra · Henri E. Bal**

Published online: 23 March 2011


**Abstract** The research area of multimedia content analysis (MMCA) considers all aspects of the automated extraction of knowledge from multimedia archives and data streams. To adhere to strict time constraints, large-scale multimedia applications typically are being executed on distributed systems consisting of large collections of compute clusters. In a distributed scenario, it is first essential to determine the optimal number of compute nodes used by each cluster, properly balancing the complex tradeoff between computation and communication. This issue is referred as the

R. Yang (✉) · R. D. van der Mei · D. Roubos
Department of Mathematics, Faculty of Sciences, VU University, De Boelelaan 1081A,
1081 HV Amsterdam, The Netherlands
e-mail: ryang@few.vu.nl

R.D. van der Mei
e-mail: mei@few.vu.nl

D. Roubos
e-mail: droubos@few.vu.nl

R. Yang · R. D. van der Mei
Centre of Mathematics and Computer Science, Science Park 123,
1098 XG Amsterdam, The Netherlands

R. Yang
e-mail: r.yang@cwi.nl

R. D. van der Mei
e-mail: mei@cwi.nl

F. J. Seinstra · H. E. Bal
Department of Computer Science, Faculty of Sciences, VU University, De Boelelaan 1081A,
1081 HV Amsterdam, The Netherlands

F. J. Seinstra
e-mail: fjseins@cs.vu.nl

H. E. Bal
e-mail: bal@cs.vu.nl

"resource utilization" (RU) problem. Next, it is important to tune the transmission of newly generated data sent to each cluster, so as to obtain the highest *service utilization*, while minimizing the need for buffering. This latter issue is referred as the problem of "just-in-time" (JIT) communication. In this paper, we first present a simple and easy-to-implement method for the RU problem, which is based on the classical binary search method. Second, we address the JIT problem by introducing a smart adaptive control method that properly reacts to the continuously changing circumstances in distributed systems. Extensive experimental validation of the two approaches on a real distributed system shows that our optimization approaches are indeed highly effective.

**Keywords** Multimedia content analysis · Distributed computing · Resource optimization

## 1 Introduction

In recent years, the increasing role of multimedia data, in particular in the form of still pictures and video, has boosted demands for extraction, comparison, and processing of *features* from multimedia data sources. The domain of Multimedia Content Analysis (MMCA) aims to adhere to these demands, and to arrive at automated methods of extracting new knowledge from multimedia data. In part, the MMCA domain is driven by the requirements of emerging applications, ranging from the automatic comparison of forensic video evidence, to searching publicly available digital television archives, and real-time analysis of video data obtained from surveillance cameras in public locations [37].

In the very near future, 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 of storage (http://privacy.cs.cmu.edu/dataprivacy/projects/explosion). As individual compute clusters cannot satisfy the high computational demands, distributed supercomputing on large collections of compute clusters is rapidly becoming indispensable.

Moreover, applications in MMCA often must run under strict time constraints. For example, to avoid delays in queues of people waiting, a biometric authentication system must identify a person's identity within several seconds. Largely autonomous applications, such as the automatic detection of suspect behavior in video data obtained from surveillance cameras, may even need to work under real-time restrictions.

In a typical *services-based* 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 images or video frames (e.g., captured by a camera) to any number of available servers, each performing the analysis in a data parallel manner [32]. Note that applications running under this scenario form a specific class of applications, i.e. those having relatively static, repetitive workloads. Examples include (real-time) video processing applications in which the same data analysis is performed on each video frame in turn, and (off-line) image database applications in which the same

processing is performed on each image stored. In this paper we specifically target this class of applications.

For reasons of efficiency (be it in computational terms, economical, environmental, or otherwise), it is essential to find the best match between the available compute resources and the multimedia analysis problem at hand. In an execution scenario in which a number of multimedia servers are being executed on a distributed set of compute clusters, the *resource optimization* problem can be separated into two main parts. First, it is essential to determine the optimal number of compute nodes used by each individual multimedia server. This part of the optimization problem generally depends on a priori system information, including the multimedia server application itself, and the specifics of the computing environment (e.g., network characteristics, CPU power, memory, etcetera). In this context, it is essential to properly balance the following trade-off: if the number of compute nodes employed by a multimedia server is too low, the processing power is insufficient to meet the strict time constraints of real-time applications; if the number of compute nodes is too high, the parallelization overhead will cause a degradation of the computational performance. This problem is referred to as the *resource utilization* (RU) problem throughout this paper. Clearly, as researchers in the MMCA domain generally are not experts in parallel computing, there is an urgent need for *simple* and *easily implementable*, yet *effective* methods (in terms of the number of evaluation steps) for determining the optimal level of parallelism. Also, the method should be easily *adaptable* to inherently dynamic changes in the distributed environment.

Second, based on the result achieved from the RU problem, it is essential to employ the allocated resources efficiently by sending data (e.g., video frames) to each multimedia server at carefully determined moments in time, in order to obtain the highest *service utilization* possible, and to minimize the service response time. Clearly, if an available multimedia server is currently unoccupied, analysis results for a video frame can be obtained in the fastest possible way. Unfortunately, keeping a multimedia server mostly idle is a waste of compute resources. Alternatively, sending video frames to a multimedia server as soon as possible may cause a need for queuing of video frames at the server side. Having to wait for the processing of previously queued data may result in an unacceptably long delay between the moment of data generation and result calculation. Hence, to optimize server utilization and response time, it is essential to tune the transmission of video frames to the occupation of the remote multimedia servers. Due to variations in transmission latencies and other variabilities in the computing environment, however, it is difficult to accurately tune the sending of video frames to the variable response time of a multimedia server. In this paper we refer to this issue as the *just-in-time* (JIT) communication problem.

To solve the JIT problem, we need effective prediction methods that react to the continuously changing circumstances in distributed systems. An immediate consequence of a JIT approach is that a multimedia server always analyzes the most recently generated (or, "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 applications.

The main contributions of this paper are as follows: (1) we provide a solution to the JIT problem which is entirely new, as—to our knowledge—it has not been addressed in the literature before, and (2) we provide an innovative solution to the

RU problem, which—in contrast to existing methods—is a fully dynamic, runtime approach. Our solution requires only limited (run-time) benchmarking, which is performed in a transparent and portable manner. Also, our solution is independent of the specific implementation of the applications at hand, making our solution highly sustainable (as it is immediately applicable, even after the application is altered).

This paper is organized as follows. In Section 2 we present related work, and address the pros and cons of existing methods. Section 3 presents our proposed approaches, which are further formulated in Section 4. Section 5 presents the experimental setup, and describes example applications. Section 6 discussed our experimental results. Finally, Section 7 concludes.

## 2 Related work

Previous work in this field can be categorized into two groups. The first group, relevant to our RU problem, incorporates the general performance estimation and optimization problem of computer systems. The second group, relevant to our JIT problem, relies on statistical predictions of system behavior.

Roughly speaking, techniques to general performance estimation can be classified into one of three main categories: (1) *measurement*, (2) *modeling* and (3) *hybrid* methods. Estimation techniques that belong to the second category can be further divided into the subcategories of (2a) *mathematical analysis* and (2b) *simulation* [18].

Performance estimation by *measurement* is generally performed on a real system under conditions that reflect typical workload and behavior. Execution times of real problems are then inferred from measured results [22, 40]. Application of this approach has several drawbacks. First, in many cases the complete system to be evaluated has yet to be developed, and may change over time. Second, even if a complete system is available it is often not clear what workload is realistic or typical. Finally, if the measurement process is biased towards certain aspects of the underlying hardware, the measurement technique may not be applicable to other platforms.

*Benchmarking* is an alternative technique within the category of measurement, which is often used for comparison of multiple computer systems (e.g., see [4, 5, 9, 16, 41]). Rather than reflecting typical behavior, benchmarks often represent non-typical, artificial workloads. In comparison with direct measurement, benchmarking has the advantage that the system to be evaluated does not have to be available. The use of non-typical workloads, however, often has a negative effect on the accuracy of the performance estimations. A solution—albeit complex—is to capture results for small instruction mixes and a variety of workloads, and to interpret the measurement results with utmost care [8, 39].

Performance *modeling* can be applied in cases where direct measurement is too costly, or where the computer system to be evaluated is not available. In the category of *mathematical analysis*, models range from simple (linear) algebraic expressions to complex formalisms such as queueing networks [18, 29]. In general, such models have a high response time due to their ease of evaluation. An additional advantage is that parameter values may be varied to observe their relative impact on performance. However, to obtain high estimation accuracy, the large number of model parameters may violate the simplicity and applicability constraints.

In *simulation models* behavior and workloads are described (imitated) in a special computer program—usually an annotated or otherwise adapted version of a 'real' program [18, 26]. Performance predictions are obtained by monitoring the execution of the adapted program. The main advantage of simulation models is that dynamic system behavior is easily captured. Also, simulation makes it easy to 'zoom in' on interesting or expensive parts of a system. A disadvantage is that the system to be evaluated must be available, at least in some rudimentary form. Another drawback is that it is a costly method for obtaining even moderately accurate performance estimates.

In hybrid estimation techniques a combination of measurement and modeling is applied [24, 46]. Such techniques have the advantage that the complexity of using either measurement or modeling in isolation can be avoided, while a high level of estimation accuracy can still be obtained. As an example of an approach in this direction, Saavedra-Barrera et al. [28] have measured system performance for *sequential* Fortran programs in terms of an Abstract Fortran Machine (AFM), an approach referred to as narrow spectrum benchmarking. The AFM-based approach provides a solution to the problem of the high complexity of complete analytical study of computer systems. The drawback of the approach, however, is that system variance is almost completely ignored. For applications working on extensive dense data fields (e.g., image data structures) this is a too crude restriction as variations in the hit ratio of caches and system interrupts often have a significant impact on performance [12, 30].

Other performance estimation techniques that incorporate more detailed behavioral abstractions relating to the major components of a computer system [18, 23] need tens—if not hundreds—of platform-specific machine abstractions to obtain truly accurate estimations. Consequently, the requirements of simplicity and applicability to the MMCA domain are not satisfied. To overcome this problem, Seinstra et al. [33] have designed a new model for performance estimation of *parallel* image and video processing applications running on clusters, based on the Abstract Parallel Image Processing Machine (APIPM). The APIPM model has been used in a large set of realistic image and video processing applications to find the optimal number of compute nodes. The main advantage of this model is that predictions are based on the analysis of a small number of rather high level system abstractions (i.e., represented by the APIPM instruction set). The main limitation of this model, however, is that the instruction set and its related performance values are parameterized with a very large number of instruction behavior and workload indicators. As such, the model still does not meet our requirements, as obtaining accurate performance values for all possible parameter combinations is both costly and complex.

For our JIT problem, prediction techniques can be classified into analytical [20], artificial intelligence (AI), and statistical methods. The models in analytical techniques are constructed by hand or use automatic code instrumentation. AI methods, such as neural network-based method, predict the future performance of resources or applications by learning from historical data and classifying the information. Statistical approaches analyze the successive historical data using the statistical methods (e.g., time series analysis [34]) in an effort to predict the data in future. Experience has taught that even some seemingly random or very noisy series can be modeled and predicted to a usable error margin [10] using statistical methods. Therefore, we

restrict ourselves in this paper only on the statistical prediction methods to forecast the properties of a Grid.

To accurately predict job runtimes in a Grid environment, it is essential to have a method that effectively reacts to the peaks and level switches in job runtimes. For this purpose, Dobber et al. [7] developed Dynamic Exponential Smoothing (DES) methods based on traditional exponential smoothing (ES) method [2, 3, 17, 42]. Sonmez et al. [38] use mean-based, median-based and ES for predicting job runtimes and job queue waiting times, whilst Berman et al. [1] choose the Network Weather Service (NWS) prediction algorithm for the same purpose. To predict different properties of a Grid, the NWS algorithm selects between the following three prediction methods: mean-based method, median-based method and Autoregressive (AR) method. For instance, Wolski et al. [44] take the NWS algorithm to predict resource availability. Furthermore, Smith et al. [35] and Guim et al. [13] aim to predict the total running times of parallel applications. The former one uses the mean-based and the Linear Regression (LR) method, while the latter one uses mean-based and median-based methods. Moreover, AR is applied by Zhang et al. [49] and Wu et al. [45] to predict CPU load and by Qiao [27] to predict network traffic. To improve the accuracy of the prediction, the basic forecasting methods can be applied adaptively (e.g., adapted mean-based method [43], adapted median-based [43] method and adaptive ES-based method). These adapted prediction methods have shown to be very accurate. Apart from the basic forecasting methods, some research areas are interested in predictors that estimate the possibility of an event from its likelihood and prior probability as its probability conditional to its characteristics, such as Bayesian inference used in [25] to predict the resource availability in a Grid.

Recall that in the context of MMCA, prediction methods should be simple and easily implementable, yet effective because of the strict time requirement of the multimedia application. Therefore, in this paper we only use prediction methods (i.e., the adapted mean-based method, the adapted median-based method, ES, and the Robbins-Monro Stochastic Approximation method [21]) that are simple and fast, yet accurate.

For our JIT problem, we argue that existing statistical prediction methods are not capable of adhering to the specific requirements of JIT communication. One important problem with existing methods is that *random peaks* can be observed in the processing time of each multimedia server. These delays cause accumulative errors in predicting the exact moments of video frame transmission, resulting in significant deviations from the optimal strategy. Similarly, existing methods cannot deal with *periodic peaks* very well either. These observations have raised a need for additional policies to amend these particular problems.

## 3 General: proposed approaches

In practice, running CPU-intensive applications in large-scale distributed computing environments typically consists of two phases: (1) an *initialization phase* to determine the optimal number of compute nodes $L^*$, and (2) the *main phase* to actually run the application on the $L^*$ parallel nodes. In this paper, each of our proposed approaches is used in one of these phases, respectively.

### 3.1 Resource utilization (RU) problem

First, we propose a simple method for on-the-fly determination of the "optimal" level of parallelism. Unlike analytical methods, our parallel multimedia server together with the underlying execution platform is treated as a black box from the resource allocator's point of view. This is due to our need of obtaining a *general* and *robust* approach to solve the optimization problem.

With our software and hardware assumed as black boxes, we are faced with the problem of having to deal with a search space that is unlimited in theory (and in practice limited only by the total number of available nodes in a given cluster system). As a result, it is essential to apply heuristics that can reduce our search space significantly. In this context, extensive experimental observations for realistic, large-scale problems in MMCA have revealed the following three important properties of optimal resource allocations:

First, in many cases the optimal number of compute nodes is found to be a power of 2, i.e., of the form $2^m$ for some $m = 0, 1, \ldots$ [47]. This observation is important because it leads to a drastic reduction of the set of possible solutions. For example, if the number of available compute nodes is $L_{\max}$, the size of the solution space is reduced from $L_{\max}$ (i.e., the number of elements in the index set $\{1, \ldots, L_{\max}\}$) to $\lfloor \log_2(L_{\max}) \rfloor$ (i.e., the number of elements of the set $\{2^0, 2^1, \ldots, 2^K\}$ where $K = \lfloor \log_2(L_{\max}) \rfloor$). Here the symbol $\lfloor x \rfloor$ represents the largest integer $\leq x$.

Second, on compute nodes consisting of multiple CPUs (and potentially multiple cores), for a fixed number of compute elements, using more compute nodes and less CPUs per node yields better performance.

Third, if the compute cluster processing time is denoted by $S(L)$, with $L$ the number of compute nodes, then there exists a threshold value $L^*$ such that $S(L)$ decreases fast as a function of $L$ for $L < L^*$, whereas $S(L)$ flattens out, and may even increase, for $L > L^*$. $L^*$ is commonly referred to as the *engineering knee*. Moreover, in practice using too many compute nodes may be very costly. $L^*$ should be the smallest number that matches the conditions specified above.

It should be noted here that our first two observations above may not be (and probably are not) true for all potential target systems. For such systems, however, other heuristics will apply, which can then be used for our search space reduction. Such other heuristics do not affect the manner in which our search is applied.

Based on the above observations, our proposed method is aimed at determining $L^*$ as the optimal point of operation. The method takes the idea of the well-known classical binary search method for non-linear optimization, and converges if the relative improvement of $S(L)$ with respect to $L$ (on a log scale) is close enough to 0 (say 5–10%). In Section 4 we will give a complete formulation of our method.

### 3.2 JIT communication problem

A simple execution approach to solve the JIT communication problem, 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 Fig. 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.

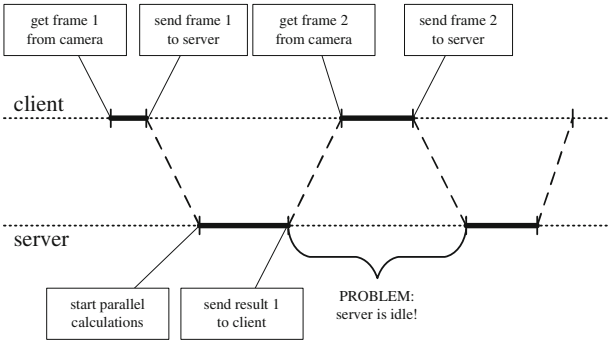



**Fig. 1** BBM approach for video frame transmission

In a bottleneck situation, the video frame transmission time from the client to the server ($Tc_1$) and the time to send a result back ($Tc_2$) may be long. In practice, $Tc_1$ is normally very close to $Tc_2$, thus we denote them by $Tc$. Then, the service utilization ($SU$) using BBM 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 communication time increases, service utilization decreases.

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 Fig. 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 $(i+1)$-st 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 $Ts_i$, sending the next frame after a period of $Ts_i$ will give an

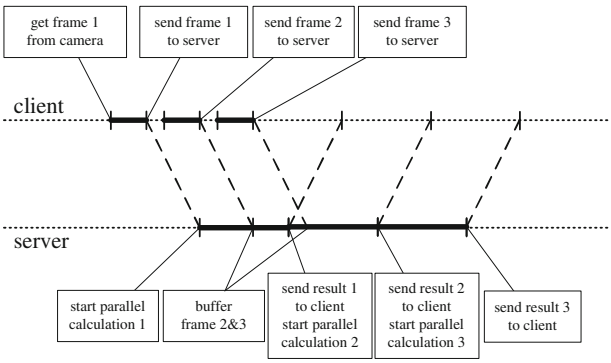



**Fig. 2** BSM approach for video frame transmission

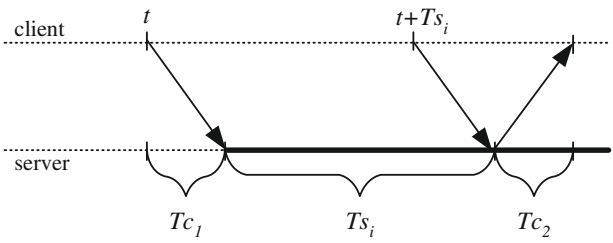



**Fig. 3** An optimal solution for video frame transmission

optimal solution. With this strategy, the server gets the most up-to-date frame and the service utilization is unity (see Fig. 3). Unfortunately, $Ts_i$ 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.

We have observed that existing predictive methods (i.e., the adapted mean-based method [43], the adapted median-based method [43], exponential smoothing [2, 3, 17, 42], and the Robbins-Monro Stochastic Approximation method [21]) are all capable of generating an accurate trend line based on the processing time of previous frames. However, for our JIT communication problem, these methods are not sufficiently optimized for particular cases. The first problem appears, when the processing time of certain frames suddenly become much longer (e.g., a peak) than the expected $Ts$ 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 $Ts$ (see Fig. 4).

Apart from random peaks, a second complication 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 methods mentioned above can effectively deal with random peaks very well, nor do these pay attention to periodic characteristics. See [48] for more details.

We propose two policies to amend these 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 methods, including the BLM and PP policies, provide good solutions for our JIT communication problem.

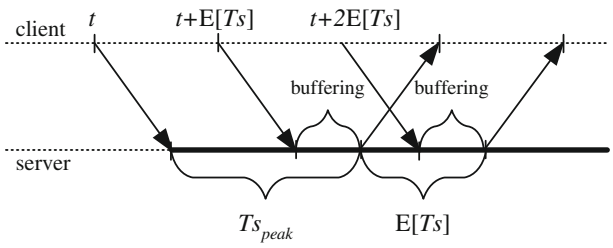



**Fig. 4** All frames are affected continuously by sudden long process times

## 4 Detail: method formulation

This section describes the two proposed modeling approaches in detail. The approaches are based on the results of extensive experimentation performed on the DAS-3 distributed cluster system (see Section 5).

### 4.1 Resource utilization (RU) problem

In our services-based execution scenario, video frames are being processed on a per-cluster basis, using a varying number of compute nodes on each cluster, each consisting of multiple CPUs. The compute cluster (or *service*) processing time is defined as a function $S(L, n)$ of the number of compute nodes $L = 1, \ldots, L_{\max}$ and the number of CPUs per node $n = 1, 2, \ldots, n_{\max}$. Our goal is to minimize the cost function $S(L, n)$ over the set of possible values of $(L, n)$; thus, we are searching for the point $(\hat{L}, \hat{n})$ where $S(L, n)$ attains its minimum.

As stated earlier, the set of possible combinations $(L, n)$ may be very large such that, in practice, finding the optimum $(\hat{L}, \hat{n})$ may be very time consuming. In the previous section, we have defined a number of heuristics that lead to a drastic reduction of the set of possible values of $(L, n)$. In a general form, our heuristics reduce the solution set to the combinations $\mathbb{X} = \{(2^p, 1), p = 0 \ldots P\} \cup \{(2^P, 2^q), q = 1 \ldots Q\}$, where $P := \lfloor \log_2(L_{\max}) \rfloor$, $Q := \lfloor \log_2(n_{\max}) \rfloor$. Therefore, the solution set is reduced drastically from $L_{\max} n_{\max}$ to $P + 1 + Q$. The cost function $S$ is a sorted list according to the observation in the last section. For simplicity, we use $(2^{(P+q)}, 1)$ instead of $(2^P, 2^q)$ for our notation, although $(2^{(P+q)}, 1)$ does not exist.

#### *4.1.1 Approximating the optimal* $(L, n)$

From the reduced solution space, we iteratively increase the total number of CPUs to find the optimal $(L, n)$. When the number of applied compute nodes becomes larger, the parallelization overhead increases, and may even become dominant. Our experimental results show that there exists a threshold value $m^*$ such that $S(2^m, 1)$ decreases fast for $m < m^*$, whereas $S(2^m, 1)$ flattens out, and may even increase, for $m > m^*$. As an illustration, Fig. 5 shows the average service processing times for an example application (described in detail in the next section) for different values of $L = 2^m$. We observe that there exists some *saturation point* $L^* = 2^{m^*}$ such that increasing the number of parallel nodes $L$ beyond $L^*$ does not lead to a significant reduction of the service processing times. Throughout, $L^* = 2^{m^*}$ will be referred to as the *engineering knee* and is regarded as the (near-) optimal point of operation. It is worthwhile to note that the optimal point is not fixed due to the dynamic changes in the distributed environment.

To find the engineering knee $L^*$, we have developed a *Logarithmic Dichotomy Search* (LDS) method. This method can fulfill the requirement of seeking the engineering knee in a dynamic environment. The LDS method follows the idea of a well-known conventional binary search (CBS) algorithm [19] which aims to find a particular value in a sorted list. Compared to the CBS strategy, the LDS method makes progressively better guesses, and proceeds closer to the optimal value. Let the elements in the solution set $\mathbb{X}$ be denoted by $(e_0, \ldots, e_K)$, with $K = P + Q$, $P$ and $Q$

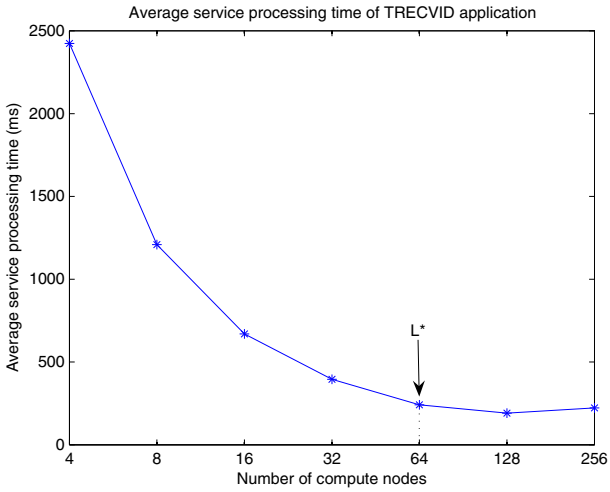



**Fig. 5** Engineering knee of example application

are defined above. The LDS strategy selects the median element in the set $\mathbb{X}$, denoted by $e_{\text{Mid}}$. Define $\epsilon$ as the desired minimal improvement in the service processing time by increasing the number of compute nodes. If $\frac{S(e_{\text{Mid}})-S(e_{\text{Mid}+1})}{S(e_{\text{Mid}})} > \epsilon$, then we repeat this procedure with a smaller list, and we keep only the elements $(e_{\text{Mid}+1}, \ldots, e_K)$. If $\frac{S(e_{\text{Mid}})-S(e_{\text{Mid}+1})}{S(e_{\text{Mid}})} \leq \epsilon$ then the list in which we search becomes $(e_1, \ldots, e_{\text{Mid}})$. Pursuing this strategy iteratively, it narrows the search by a factor of two each time, and finds the minimum value that satisfies our requirement after $\log_2(K)$ iterations.

Note that the selection of $\epsilon$ is very important in finding the engineering knee. A large $\epsilon$ means that we are easily satisfied with the improvement. However, the result may not be close to the actual optimum. Setting $\epsilon$ to a very small value or even zero certainly will let us find the engineering knee (which is close to, or equal to, the optimal number of compute nodes), but this may take an undesirably long time. Hence, in practice $\epsilon$ is always a small positive number which is close to, but not equal to, zero. The pseudo code for our LDS method for the solution space $\mathbb{X}$ is given in Algorithm 1.

**Algorithm 1** Pseudo code of LDS strategy.

```
Low := 0
High := K
While (Low < High) {
        Mid := ⌊ (Low+High)/2 ⌋
        if S(e_Mid) ≤ S(e_Mid+1)/(1−ε) {High = Mid;}
        else {Low = Mid +1;}
        end if;
}
Optimal number of compute nodes := High.
```

## 4.2 JIT communication problem

The following continues with a detailed fomulation of the proposed solution for the JIT problem. The notations used here 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 result from the server.

### *4.2.1 Preliminaries*

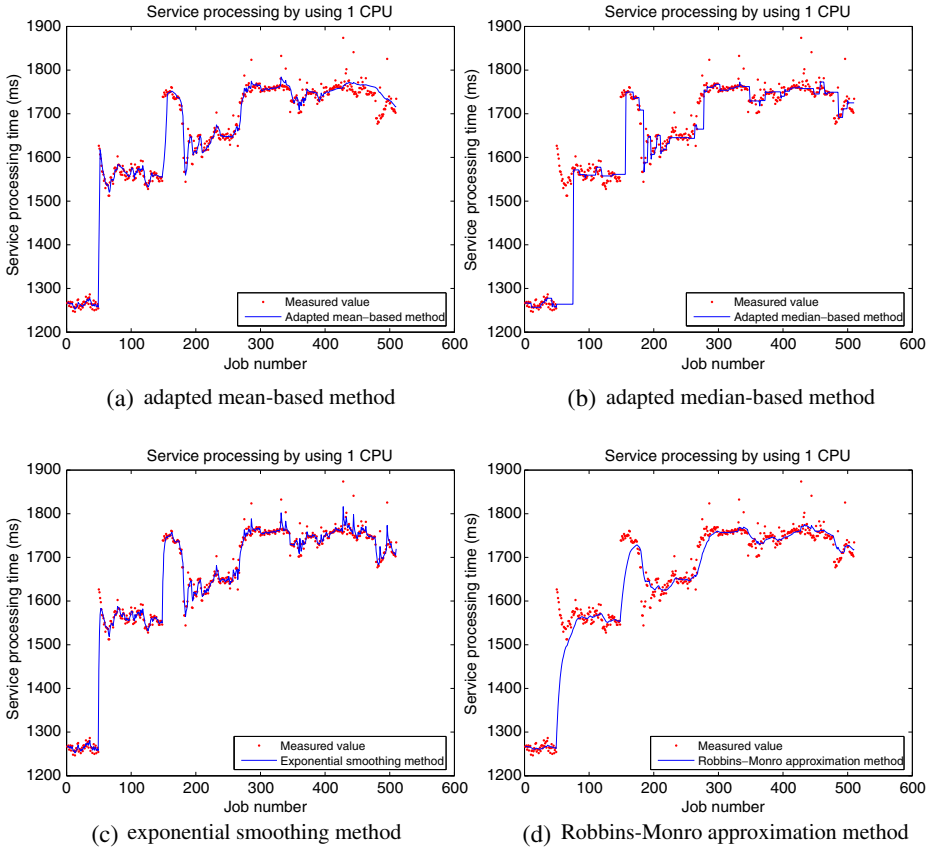
*Trend line* As shown in Fig. 3, if we can predict the service processing time of the current frame accurately, then sending the next frame after the predicted time unit should provide an optimal solution. Therefore we investigated several conventional prediction methods (i.e., adapted mean-based methods, 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 methods, an accurate trend line can be generated. Figure 6 gives an illustration of the predicted service processing time versus the measured value of running an example application using one compute node and a single CPU only.

*Periodicity of the peaks* Another important observation from our experimental results is the occurrence of periodic peaks when using large numbers of compute nodes. Because our multimedia applications are partially implemented in Java, the *Java garbage collector* (http://www.artima.com/underthehood/gc.html) 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 Fig. 6. In contrast, when the service processing time is small compared to the garbage collection time, the periodic peaks are significant. We ran an example application using 64 compute nodes (using one CPU per node) during three different periods in time. From these data sets, we notice that there is a deterministic period of the occurrences of certain specific peaks (see Fig. 7).

### *4.2.2 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 methods and our BLM and PP policies in detail.

*Prediction methods* 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 [43] uses arithmetic averages over some portion of the measurement history to predict the next measurement.




(a) adapted mean-based method (b) adapted median-based method

(c) exponential smoothing method (d) Robbins-Monro approximation method

**Fig. 6** Trend line generated by different prediction methods

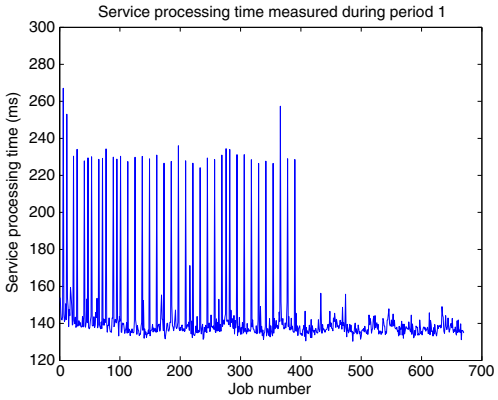
In particular, the extent of the history taken into account depends on a parameter $K$, specifying the number of previous measurements for the arithmetic average. The parameter $K$ is changed by $-1$, 0, or $+1$ over time based on the prediction error. In our experiments, the initial value of $K$ is set to 20.

Adapted median-based methods [43] 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.

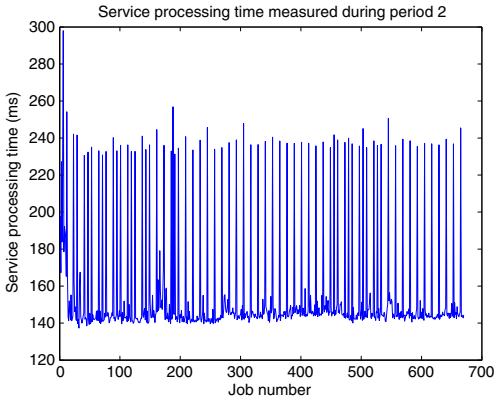
In exponential smoothing [2, 3, 17, 42] earlier measurements are not weighted equally as in the case of a mean-based method, but with exponentially decreasing weights as the measurements get older. More specifically, denote by $w(i)$ the weight for the $i$-th previous measurement. Then, $w$ is the following function

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

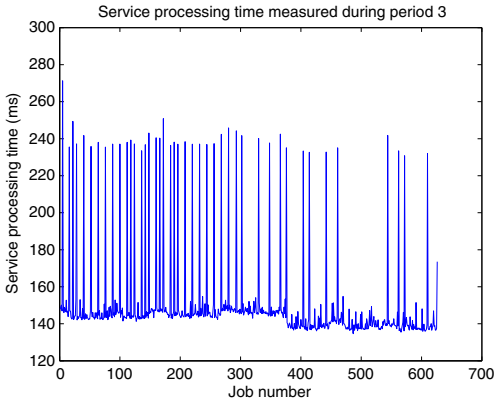
with $\alpha$ a parameter determining the rate of decay of the function. In our experiments, we set $\alpha = 0.5$. As in the previous methods, the parameter $K$ determines the number




(a) Measurement 1




(b) Measurement 2




(c) Measurement 3

**Fig. 7** Service processing time taken at different times

of earlier measurements that we intend to use. In case $K >$ {# 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 [21] is a stochastic method. If we denote by $\hat{Ts}_i$ the estimation of the $i$-th processing time, then the estimation is updated according to the following relation

$$\hat{Ts}_{i+1} = \hat{Ts}_i + \varepsilon_i \left( Ts_i - \hat{Ts}_i \right),$$

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

*BLM policy* Our first policy to deal with peaks is called "one-before-last-measurement" (BLM) policy. This policy determines the optimal sending time under the following three cases.

**Case 1: waiting for sending**

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 cannot use the last measurement data as a predictor (also indicated by Harchol-Balter and Downey [14]). Therefore $k$ must be larger or equal to 2. Throughout this paper, we focus on the case that $E[Tc] \leq \frac{E[Ts]}{2}$. Here $E[Ts]$ and $E[Tc]$ represent the expected service processing time and the communication time respectively. 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 $Tc > \frac{E[Ts]}{2}$, we only need to enlarge the value of $k$. Hence, for $k = 2$, we have the following equation,

$$t_i \geq r_{i-2}. \tag{4.1}$$

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.

**Case 2: sending immediately**

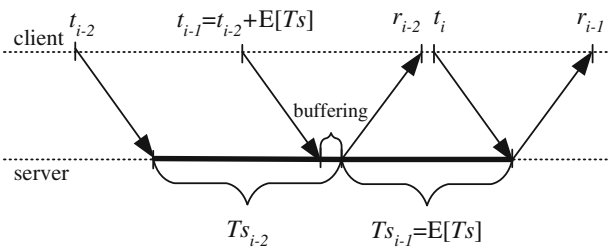
Obviously, if the result of the $(i-1)$-th frame is received, the $i$-th frame must be sent immediately. Therefore, we have

$$t_i \leq r_{i-1}. \tag{4.2}$$

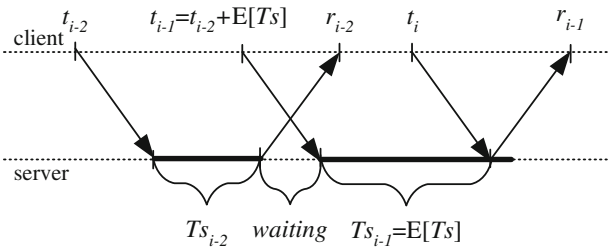
**Case 3: adjusting sending time**

The sending time of the $i$-th frame is also decided by the relationship between the expected service processing time and measured service processing time of the $(i-2)$-th frame $Ts_{i-2}$. If $Ts_{i-2} > E[Ts]$, then it is optimal to send the $i$-th frame at $r_{i-2} + E[Ts] - 2 \cdot E[Tc]$. Figure 8a gives an example. In case $Ts_{i-2} \leq E[Ts]$, the optimal sending moment is at $t_{i-1} + E[Ts]$. See Fig. 8b. Hence we get the following equation,

$$t_i = \begin{cases} r_{i-2} + E[Ts] - 2 \cdot E[Tc] & \text{if } Ts_{i-2} > E[Ts], \\ t_{i-1} + E[Ts] & \text{otherwise.} \end{cases} \tag{4.3}$$




(a) Optimal sending time in case of $TS_{i-2} > E[Ts]$




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

**Fig. 8** Overview of the 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 (4.1), (4.2), and (4.3), the optimal sending time of $i$-th frame is given by

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

*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 Fig. 7 for the experimental results. Denote $\mathbb{P} = \{i | Ts_i \text{ is peak}\}$ as the set of peaks and denote by $\widetilde{p}_j$ the $j$-th element of $\mathbb{P}$. Let $k$ be an integer number. If $\widetilde{p}_j - \widetilde{p}_{j-1} = \cdots = \widetilde{p}_{j-(k+1)} - \widetilde{p}_{j-k}$ then we say that there is a deterministic period of length $d = \widetilde{p}_j - \widetilde{p}_{j-1}$, 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 model to deal with the JIT communication problem in real-time applications.

## 5 Experimental setup

In a Grid environment, resources have different capacities and many fluctuations exist in load and performance of geographically distributed nodes [6]. As the availability of resources and their load continuously vary over 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 DAS-3 (the Distributed ASCI Supercomputer 3) Grid test bed (http://www.cs.vu.nl/das3/).

DAS-3, see Table 1 and Fig. 9, is a five-cluster wide-area distributed system, with individual clusters located at four different universities in The Netherlands: VU University 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, located at the University of Amsterdam. As one of its distinguishing features, DAS-3 employs a novel internal wide-area interconnect based on optical 10G links (StarPlane http://www.starplane.org/).

### 5.1 Example applications

In our experiments, we use DAS-3 to run a real-time multimedia application (referred to as "Aibo"), as well as an off-line application (referred to as "TRECVID").

The Aibo application demonstrates real-time object recognition performed by a Sony Aibo robot dog [32] (see Fig. 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

**Table 1** Overview DAS-3 cluster sites

| Cluster | Nodes | Type | Speed (GHz) | Memory (GB) | Storage (TB) | Node HDDs (GB) | Network |
|---|---|---|---|---|---|---|---|
| VU | 85 dual | Dual-core | 2.4 | 4 | 10 | 85 × 250 | Myri-10G and GbE |
| LU | 32 dual | Single-core | 2.6 | 4 | 10 | 32 × 400 | Myri-10G and GbE |
| UvA | 41 dual | Dual-core | 2.2 | 4 | 5 | 41 × 250 | Myri-10G and GbE |
| TUD | 68 dual | Single-core | 2.4 | 4 | 5 | 68 × 250 | GbE (no Myri-10G) |
| UvA-MN | 46 dual | Single-core | 2.4 | 4 | 3 | 46 × 1,500 | Myri-10G and GbE |

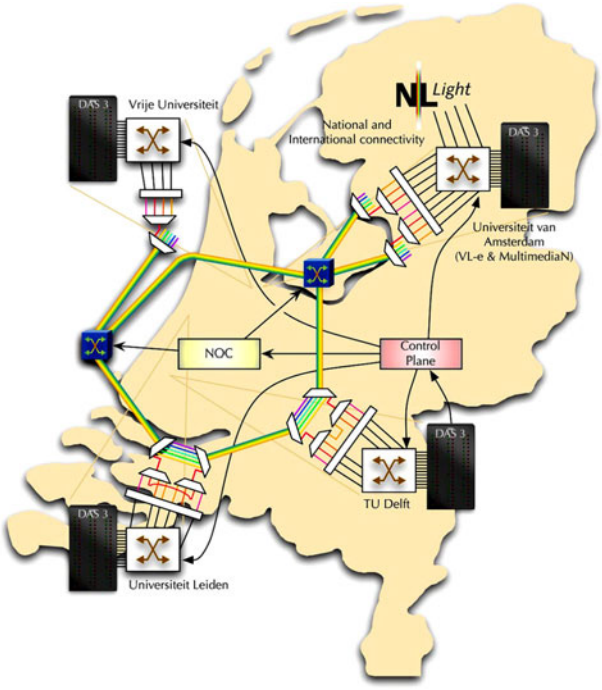



**Fig. 9** The distributed ASCI supercomputer 3

repositories of 'known' objects [11], 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 Fig. 10) indispensable.

The TRECVID application represents a multimedia computing system that has been applied successfully in recent editions of the international NIST TRECVID benchmark evaluation for content-based video retrieval [15, 36]. The aim of the TRECVID application is to find semantic concepts (e.g., vegetation, cars, people, etc.) in hundreds of hours of news broadcasts, a.o., from ABC and CNN. The

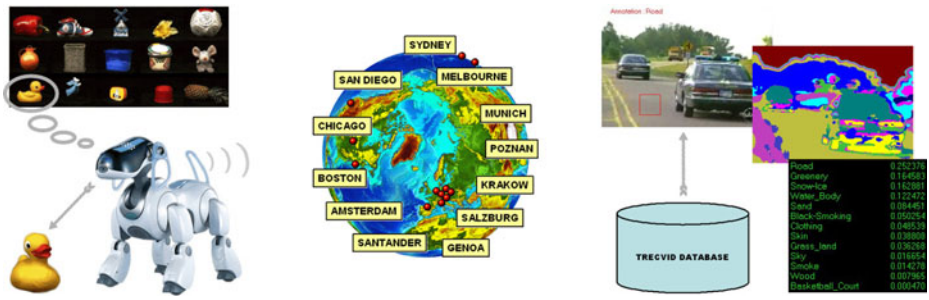



**Fig. 10** Our example real-time (*left*) and off-line (*right*) distributed multimedia applications, which are capable of being executed on a world-wide scale. The real-time application constitutes a visual object recognition task performed by a robot dog (Aibo). The off-line application constitutes our TRECVID system

TRECVID concept detection task is, in general terms, defined as follows: Given the standardized TRECVID video data set, a common shot boundary reference for this data set, and a list of feature definitions, participants must return for each concept a list of at most 2000 shots from the data set, ranked according to the highest possibility of detecting the presence of that semantic concept. TRECVID is computationally intensive; for thorough analysis it easily requires about 16 s of processing per video frame on the fastest sequential machine at our disposal [31]. Consequently, the required time for participating in the TRECVID evaluation using a single computer easily can take over one year of processing.

Both applications have been implemented using the so-called Parallel-Horus software architecture, that allows programmers to write parallel and distributed multimedia applications in a fully sequential manner [32]. The automatic parallelization and distribution of both applications 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 both applications, 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. Each 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 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 $Ts$. 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 methods combined with our BLM and PP policies.

## 6 Numerical results

In this section we present the results of our experiments performed on the DAS-3 system. Even though our methods have been applied successfully on all DAS-3 clusters, results are shown here only for the largest cluster (VU University Amsterdam) consisting of 85 compute nodes with 4 CPUs per node. For application-specific performance results on DAS-3 as a whole, and even on a world-wide set of compute clusters, we refer to [32].

### 6.1 Resource utilization (RU) problem

We start our discussion with the numerical results of the average service processing times versus a varying total number of compute nodes. In addition, the simplicity of the LDS strategy to determine the optimal number of compute nodes is validated.

First, denote the possible solution space of the compute nodes and the number of CPUs per node as $\mathbb{O}$, where $\mathbb{O} = \{(L, n),\ L \in [1, \ldots, 85] \text{ and } n \in [1, \ldots, 4]\}$. To show that using more compute nodes and less CPUs per node provides better performance in general, we ran our real-time "Aibo" application on a varying numbers of CPUs

(2, 4, 8, 16, 32, 64, and 128 CPUs). We compared the obtained service processing times for a fixed total number of CPUs, while varying the number of CPUs per nodes. The results are shown in Fig. 11. In this figure we notice that for small numbers of CPUs (say, $\leq$16), the service processing time is largely independent of the ratio between the total number of employed CPUs and the number of employed CPUs per node. As the number of CPUs increases, it becomes obvious that wider distribution of the CPUs, that is, using less CPUs per node and more compute nodes, provides better performance.

We also compared the service processing time for our off-line TRECVID application, on a varying total number of CPUs (16, 64 and 128 CPUs). The results are tabulated in Table 2. For this application we have a similar conclusion: more compute nodes and less CPUs per node provides the best performance results.

In Section 3, we mentioned that the optimal number of compute nodes is consistently found to be a power of 2. Combining this result and the observations above, we reduced the original space $\mathbb{O}$ with $85 \times 4 = 340$ possible solutions to the space $\mathbb{X}$ with nine possible solutions, where $\mathbb{X} = \{(2^i, 1), i \in [0, \ldots, 6]\} \cup (64, 2) \cup (64, 4)$. Based on $\mathbb{X}$, we apply our LDS method to find the minimum value after $\lfloor \log_2 9 \rfloor = 3$ steps. We

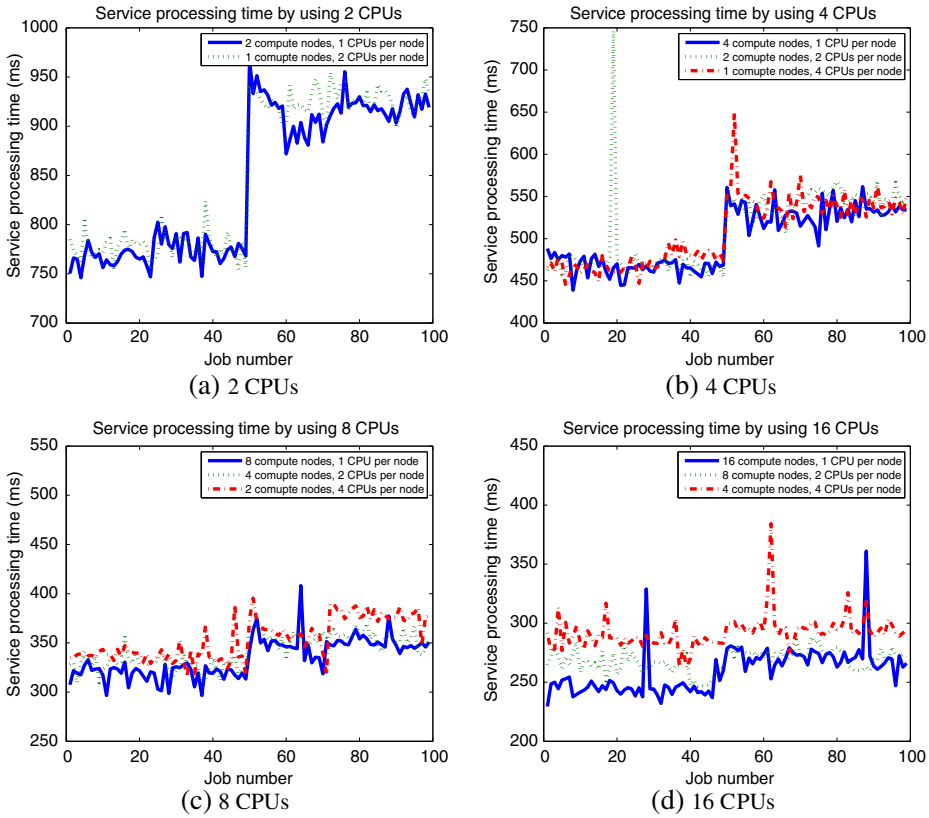
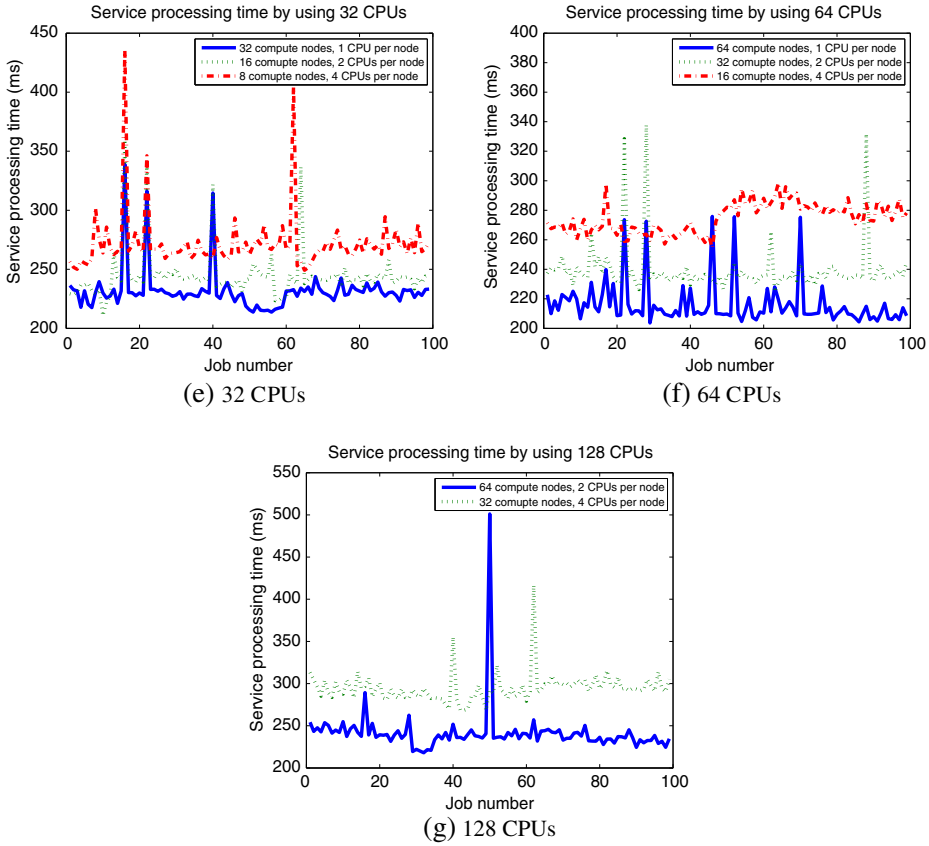



**Fig. 11** Service processing time of the Aibo application using different numbers of CPUs




(e) 32 CPUs (f) 64 CPUs

(g) 128 CPUs

**Fig. 11** (continued)

use Table 3 to explain the three steps taken in the Aibo application when $\epsilon = 0.1$. We continue to approach the optimal number of compute nodes $L^*$ by doubling the total number of compute nodes, until the relative improvement is less than 10%. Here the index of the elements of $\mathbb{X}$ is denoted as $[0, 1, \ldots, 8]$. Then the LDS method is applied. In the first step, we have $Low = 0$ and $High = 8$, and thus

$$Mid = \left\lfloor \frac{Low + High}{2} \right\rfloor = 4.$$

Therefore, we measure the service processing time using $2^4 = 16$ and $2^5 = 32$ compute nodes and 1 CPU per node. The measured average service processing times and the calculated relative improvement are shown in the first row of Table 3. Because the relative improvement using 32 compute nodes compared to 16 compute nodes is

**Table 2** Average service processing time of the TRECVID application (in ms)

| $(L, n)$ | (16, 1) | (8, 2) | (4, 4) | (64, 1) | (32, 2) | (16, 4) | (64, 2) | (32, 4) |
|---|---|---|---|---|---|---|---|---|
| $S(L, n)$ | 669.28 | 682.44 | 736.56 | 241.62 | 244.90 | 263.01 | 190.70 | 218.27 |

**Table 3** Three steps to approach the optimal $(L, n)$

| Step | Low | High | Mid | $S(e_{\text{Mid}})$ | $S(e_{\text{Mid}+1})$ | Relative improvement | Action |
|---|---|---|---|---|---|---|---|
| 1 | 0 | 8 | 4 | (16, 1) 152.26 | (32, 1) 110.64 | 0.27 | Keep high half |
| 2 | 5 | 8 | 6 | (64, 1) 93.58 | (64, 2) 108.55 | −0.15 | Keep low half |
| 3 | 5 | 6 | 5 | (32, 1) 110.64 | (64, 1) 93.58 | 0.15 | Finish, return index 6 |

0.27 ($> \epsilon$), we conclude that 16 compute nodes is not optimal. Therefore, we continue searching for the optimal. In the second step, the index value 5 (= 32 compute nodes) is set as the value of Low. The value of High remains the same. Therefore $Mid = 6$. When calculating the relative improvement using 64 compute nodes and 2 CPUs per node compared to $2^6$ compute nodes, we find that the improvement (−0.15) is less than $\epsilon$. Therefore, in the third step, the value of High is reset to 6, and Low remains the same. In this case, $Mid = 5$. The improvement of using $2^6$ compute nodes compared to $2^5$ is more than $\epsilon$. Thus, Low is reset to 6, such that Low is equal to High, and the whole procedure is finished. The LDS method returns index 6 as the optimal solution. This means, for $\epsilon = 0.1$, the optimal number of CPUs is $2^6 = 64$ compute nodes.

For different $\epsilon$ (0.1, 0.2 and 0.3), the $(L, n)$ to be evaluated and the corresponding average service processing time of both applications are reported in Tables 4 and 5, respectively. The optimal $L^*$ that we found for both applications for different values of $\epsilon$ are listed in Table 6. In this table, we notice that with larger $\epsilon$, the $L^*$ remains the same or decreases.

As shown above, we notice that our method is very simple to implement. Besides this, it is very effective because of the small number of steps required to find the optimal number of compute nodes. In addition, by varying $\epsilon$, we are able to obtain the optimal result related to the desired improvement in the service processing time by increasing the number of compute nodes.

### 6.2 JIT communication problem

The following presents the results of our experiments relating to the JIT problem. The results are also used as input for a trace-driven simulation in order to validate our final model for determining the exact transmission moments of video frames. We limit our experiments to the Aibo application, as this is the one that needs to

**Table 4** Average service processing time of the Aibo application (in ms)

| | | | | | |
|---|---|---|---|---|---|
| $\epsilon = 0.1$ | $(L, n)$ | (16, 1) | (32, 1) | (64, 1) | (64, 2) |
| | $S(L, n)$ | 152.26 | 110.64 | 93.58 | 108.55 |
| $\epsilon = 0.2$ | $(L, n)$ | (16, 1) | (32, 1) | (64, 1) | (64, 2) |
| | $S(L, n)$ | 152.26 | 110.64 | 93.58 | 108.55 |
| $\epsilon = 0.3$ | $(L, n)$ | (4, 1) | (8, 1) | (16, 1) | (32, 1) |
| | $S(L, n)$ | 448.57 | 247.72 | 152.26 | 110.64 |

**Table 5** Average service processing time of the TRECVID application (in ms)

| | | | | | | |
|---|---|---|---|---|---|---|
| $\epsilon = 0.1$ | $(L, n)$ | (16, 1) | (32, 1) | (64, 1) | (64, 2) | (64, 4) |
| | $S(L, N)$ | 669.28 | 395.79 | 241.62 | 190.70 | 222.61 |
| $\epsilon = 0.2$ | $(L, n)$ | (16, 1) | (32, 1) | (64, 1) | (64, 2) | (64, 4) |
| | $S(L, N)$ | 669.28 | 395.79 | 241.62 | 190.70 | 222.61 |
| $\epsilon = 0.3$ | $(L, n)$ | (16, 1) | (32, 1) | (64, 1) | (64, 2) | |
| | $S(L, N)$ | 669.28 | 395.79 | 241.62 | 190.70 | |

run under strict real-time requirements. The application is ran on 64 compute nodes using 1 CPU per node.

First, we apply the BBM method (Fig. 1). In our experiment, we found that the average service processing time ($E[Ts]$) and the average communication time ($E[Tc]$) between client and server amount to 143.629 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 $E[Ts]/(E[Ts] + 2 \cdot E[Tc])$. This implies that when $Tc$ is negligible, the BBM method approaches the optimal strategy. However, in a bottleneck situation where $E[Tc]$ is long relative to $E[Ts]$, the BBM method performs badly.

The 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 five steps according to Table 7. $E[Ts]$ and $E[Tc]$ in Table 7 are adjusted by one of the prediction methods. Since Fig. 6 shows that all prediction methods 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 method. In Fig. 12, it is shown that the average waiting time increases significantly as the service utilization approaches 100%. Hence, the prediction methods are not sufficient for our just-in-time communication problem.

In our final model, in which one of the prediction methods 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

**Table 6** Value of the engineering knee

| $\epsilon$ | $L^*$ |
|---|---|
| (a) Aibo | |
| 0.1 | 64 (64,1) |
| 0.2 | 32 (32,1) |
| 0.3 | 16 (16,1) |
| (b) TRECVID | |
| 0.1 | 128 (64,2) |
| 0.2 | 128 (64,2) |
| 0.3 | 64 (64,1) |

**Table 7** 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 | $E[Ts]$ |

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

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

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

$$Gain(SU) = \frac{\text{service utilization with final model}}{\text{service utilization with BBM method}}. \tag{6.1}$$

Figure 13 shows the gain of our final model related to the BBM method for different values of $\frac{Tc}{Ts}$.

In this figure, we notice 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 simple strategy can be approximated by $E[Ts]/(E[Ts] + 2 \cdot E[Tc])$. Hence, based on (6.1), we have

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

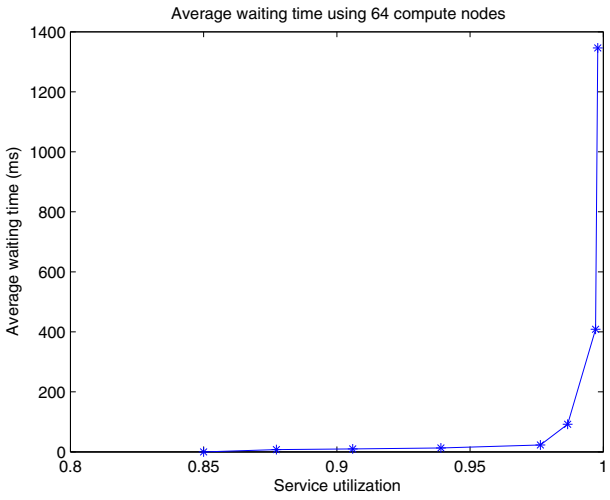



**Fig. 12** Average waiting time using 64 compute nodes

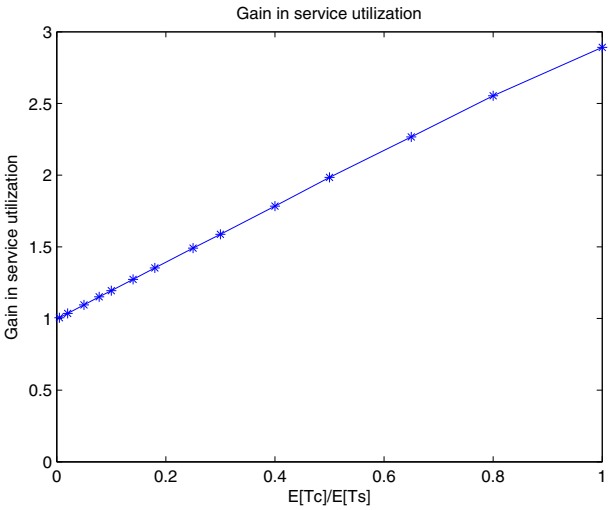



**Fig. 13** Gain in the service utilization

For this reason, 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 model to the prediction method by looking at the average waiting time. Define the gain in the average waiting time $Gain(w)$ as follows,

$$Gain(w) = \frac{\text{avg. waiting time with prediction method}}{\text{avg. waiting time with final model}}.$$

The results of this comparison are shown in Fig. 14. The reason why the final model can gain so much, can be explained by the following example. Assume that during

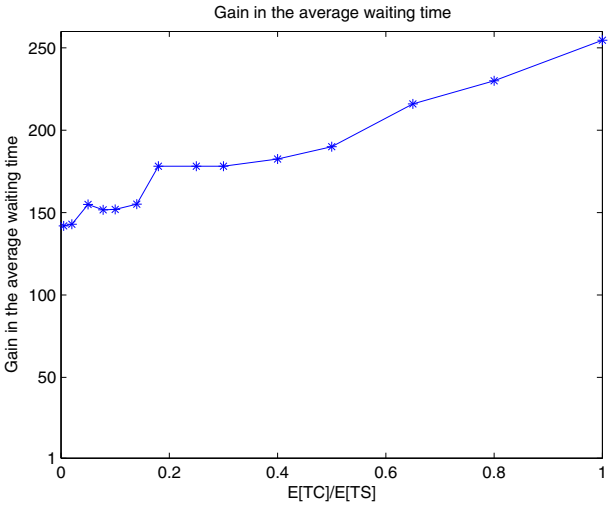



**Fig. 14** Gain in the average waiting time

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 methods 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.

## 7 Conclusions and future work

In this paper we first explored the relation between the service processing time of distributed multimedia applications and the number of compute nodes for a varying number of CPUs. We observed that there exists an engineering-knee threshold value $L^*$ such that the service processing time decreases fast as a function of $L$ for $L < L^*$, whereas the service processing time flattens out, and may even increase, for $L > L^*$. To find $L^*$, we first reduce the possible solution set, and then apply our LDS method to find $L^*$. Extensive validation has shown that our method is fast and effective.

Specifically, we have found that our method can find optimal resource utilization for an average-sized cluster system in no more than three evaluation steps. As a result, we conclude that our method adheres to all requirements as stated in the introduction: it is simple, easily implementable, and effective. In addition, our method takes into account system variation. Even though our focus was on the MMCA domain, our approach is general enough to be applicable in other domains as well.

Second, we have explored the JIT communication problem, that requires high service utilization on the one hand, and short service response time on the other. 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 methods 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 methods 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 methods described in this paper, we achieve our final model to solve the just-in-time communication problem.

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

The work described in this paper is part of a larger strive to bring the benefits of high-performance computing to the multimedia community. One important aim, in this respect, is to make large-scale distributed multimedia applications variability tolerant by way of controlled adaptive resource utilization. This raises the need

for new stochastic control methodologies that react to the continuously changing circumstances in large-scale Grid systems. Whereas the current paper focuses on optimization of resource utilization under a rather static repetitive workload, whilst taking into account system variations, further sources of variability exist.

First, in MMCA applications the amount of data that needs to be processed often changes wildly over time. For one, this is because data compression techniques cause video streams to have variable bit rates. Also, in certain specific settings, cameras may only start producing data after motion has been detected. In other cases, such as iris scans performed at airports, the amount of data to be analyzed depends on external variations.

Second, MMCA algorithms themselves are a source of variability. While many algorithms working on the pixel values in images and video streams have predictable behavior, algorithms working on derived structures, such as feature vectors describing part of the content of an image, often are data-driven. A common example is support vector machine (SVM) based classification, which tries to find an optimal separation in high-dimensional clouds of labeled data points. The identification of all support vectors that fully describe the separation depends on the positioning of the labeled data points in the high-dimensional space. Consequently, the time required to find all support vectors is largely data dependent. In the near future we will incorporate such sources of variability in our current optimization method. In addition, we will test our method on a much larger scale for a much larger variety of state-of-the-art multimedia applications. The presented example applications merely represent two of these.

**Acknowledgements** This work is supported by the Netherlands Organisation for Scientific Research (NWO), GLANCE project 643.000.602: "JADE-MM: Adaptive High-Performance Multimedia Computing".



## References

1. Berman F, Wolski R, Casanova HWC (2003) Adaptive computing on the grid using apples. IEEE TPDS 14(4):369–382
2. Brown R (1959) Statistical forecasting for inventory control. McGraw-Hill New York
3. Brown RG (1963) Smoothing, forecasting and prediction of discrete time series. Prentice-Hall
4. Cascaval C, DeRose L, Padua D, Reed D (2000) Compile-time based performance prediction. Languages and compilers for parallel computing, pp 365–379
5. Curnow H, Wichmann B (1976) A synthetic benchmark. Comput J 19(1):43–49
6. Dobber M, Koole G, van der Mei R (2004) Dynamic load balancing for a grid application. In: Proc. international conference on high performance computing (HiPC), vol 1, pp 342–352
7. Dobber M, Mei Rvd, Koole G (2007) A prediction method for job running times on shared processors: survey, statistical analysis and new avenues. Perform Eval 64:755–781
8. Dongarra J, Martin J, Worlton J (1987) Computer benchmarking: paths and pitfalls. IEEE Spectrum 24(7):38–43
9. Fahringer T (1994) Evaluation of benchmark performance estimation for parallel Fortran programs on massively parallel SIMD and MIMD computers. In: IEEE proceedings of the 2nd Euromicro workshop on parallel and distributed processing, pp 449–456
10. Farmer J, Sidorowich J (1987) Predicting chaotic time series. Phys Rev Lett 59(8):845–848

11. Geusebroek JM, Burghouts GJ, Smeulders AWM (2005) The Amsterdam library of object images. Int J Comput Vis 61(1):103–112
12. Grelck C (2000) Array padding in the functional language SAC. In: Proc. international conference on parallel and distributed processing techniques and applications (PDPTA), vol 5, pp 2553–2560
13. Guim F, Goyeneche A, Corbalan J, Labarta J, Terstyansky G (2006) Grid computing performance prediction based in historical information. In: Proceedings of the 7th IEEE/ACM international conference on grid computing
14. Harchol-Balter M, Downey AB (1997) Exploiting process lifetime distributions for dynamic load balancing. ACM Trans Comput Syst 15(3):253–285
15. Hauptmann A, Baron RV, Chen MY, Christel M, Duygulu P, Huang C, Jin R, Lin WH, Ng T, Moraveji N et al (2003) Informedia at TRECVID 2003: analyzing and searching broadcast news video. In: Proc. of TRECVID
16. Hockney R, Berry M (1994) Public international benchmarks for parallel computers. Tech. rep., PARKBENCH Committee: Report-1
17. Holt CC (1957) Forecasting trends and seasonals by exponentially weighted moving averages. ONR Memorandum, vol 52
18. Jain R (1991) The art of computer systems performance analysis. John Wiley & Sons
19. Knuth DE (1998) The art of computer programming, vol 3: sorting and searching. Addison Wesley Longman Publishing Co., Inc. Redwood City, CA
20. Kurowski K, Oleksiak A, Nabrzyski J, Kwiecien A, Wojtkiewicz M, Dyczkowski M, Guim F, Corbalan J, Labarta J, Supercomputing P (2005) Multi-criteria grid resource management using performance prediction techniques. In: Proceeding of the coregrid integration workshop
21. Kushner HJ, Yin G (2003) Stochastic approximation and recursive algorithms and applications. Springer-Verlag
22. Lee J, Asanovic K (2006) METERG: measurement-based end-to-end performance estimation technique in QoS-capable multiprocessors. In: Proceedings of the 12th IEEE real-time and embedded technology and applications symposium, pp 135–147
23. Maggs BM, Matheson LR, Tarjan RE (1995) Models of parallel computation: a survey and synthesis. In: Proc. international conference on system sciences, vol 2, pp 61–70
24. Moore M, Sztipanovitz J, Karsai G, Nichols J (1997) A model-integrated program synthesis environment for parallel/real-time image processing. In: Parallel and distributed methods for image processing. Proceedings of SPIE, vol 3166, pp 31–45
25. Nadeem F, Prodan R, Fahringer T, Iosup A (2008) A framework for resource availability characterization and online prediction in the grids. In: CoreGRID integration workshop, pp 209–224
26. Pimentel A (1998) A computer architecture workbench. Ph.D. thesis, University of Amsterdam, The Netherlands
27. Qiao Y, Dinda P (2003) Network traffic analysis, classification, and prediction. Tech. rep., Northwestern University
28. Saavedra-Barrera RH, Smith AJ, Miya E (1989) Machine characterization based on an abstract high-level language machine. IEEE Trans Comput 38(12):1659–1679
29. Sauer C, Mani Chandi K (1981) Computer systems performance modeling. Prentice-Hall Series in Advances in Computing Science and Technology, Prentice-Hall
30. Schutte K, van Kempen GMP (1997) Optimal cache usage for separable image processing algorithms on general purpose workstations. IEEE Trans Signal Process 59(1):113–122
31. Seinstra F, Snoek C, Koelma D, Geusebroek J, Worring M (2005) User transparent parallel processing of the 2004 NIST TRECVID data set. In: Proceedings of the international parallel & distributed processing symposium (IPDPS)
32. Seinstra FJ, Geusebroek JM, Koelma D, Snoek CGM, Worring M, Smeulders AWM (2007) High-performance distributed image and video content analysis with parallel-horus. IEEE Multimed 14(4):64–75
33. Seinstra FJ, Koelma D, Geusebroek JM (2002) A software architecture for user transparent parallel image processing. Parallel Comput 28(7–8):967–993
34. Shumway R, Stoffer D (2006) Time series analysis and its applications: with R examples. Springer Texts in Statistics
35. Smith W, Foster I, Taylor V (2004) Predicting application run times using historical information. J Parallel Distrib Comput 64:1007–1016
36. Snoek CGM, van Gemert J, Geusebroek JM, Huurnink B, Koelma D, Nguyen G, De Rooij O, Seinstra F, Smeulders A, Veenman C et al (2005) The MediaMill TRECVID 2005 semantic video search engine. In: Proceedings of the 3rd TRECVID workshop

37. Snoek CGM, Worring M, Geusebroek JM, Koelma DC, Seinstra FJ, Smeulders AWM (2006) The semantic pathfinder: using an authoring metaphor for generic multimedia indexing. IEEE Trans Pattern Anal Mach Intell 28(10):1678–1689
38. Sonmez O, Yigitbasi N, Epema D, Iosup A (2009) Trace-based evaluation of job runtime and queue wait time predictions in grids. HPDC
39. Steen A (1990) Is it really possible to benchmark a supercomputer? A graded approach to performance measurement. In: van der Steen A (ed) Evaluating Supercomputers: strategies for exploiting, evaluating and benchmarking computers with advanced architectures, chap 14, pp 190–212. Chapman and Hall
40. Veeravalli B, Chen L, Kwoon H, Whee G, Lai S, Hian L, Chow H (2006) Design, analysis, and implementation of an agent driven pull-based distributed video-on-demand system. Multimed Tools Appl 28(1):89–118
41. Weicker R (1984) Dhrystone: a synthetic systems programming benchmark. Commun ACM 27(10):1013–1030
42. Winters PR (1960) Forecasting sales by exponentially weighted moving averages. Manage Sci 6(3):324–342
43. Wolski R (1997) Forecasting network performance to support dynamic scheduling usingthe network weather service. In: Proc. international conference on high performance computing (HiPC), pp 316–325
44. Wolski R, Spring N, Hayes J (1999) The network weather service: a distributed resource performance forecasting service for metacomputing. J Future Gener Comp Sy 15:757–768
45. Wu Y, Yuan Y, Yang G, Zheng W (2007) Load prediction using hybrid model for computational grid. IEEE/ACM international workshop on grid computing, pp 235–242
46. Xu Z, Zhang X, Sun L (1996) Semi-empirical multiprocessor performance predictions. J Parallel Distrib Comput 39(1):14–28
47. Yang R, van der Mei RD, Roubos D, Seinstra FJ, Koole GM (2008) On the optimization of resource utilization in distributed multimedia applications. In: Proceedings of the 8th IEEE international symposium on cluster computing and the grid (CCGrid). Lyon, France, pp 358–365
48. Yang R, van der Mei RD, Roubos D, Seinstra FJ, Koole GM, Bal H (2008) Modeling "Just-in-Time" communication in distributed real-time multimedia applications. In: Proceedings of the 8th IEEE international symposium on cluster computing and the grid (CCGrid). Lyon, France, pp 518–525
49. Zhang Y, Sun WYI (2008) Predict task running time in grid environments based on cpu load predictions. FGCS 24(6):489–497

**Ran Yang** received her Master of Science degree from VU University Amsterdam, the Netherlands in 2004. Then she joined industry for 1 year. Since 2006, she is a PhD researcher in Faculty of Sciences in VU University Amsterdam, The Netherlands and PNA2 (Probability and Stochastic Networks) department in CWI (Centre of Mathematics and Computer Science), Amsterdam, The Netherlands.

**Robert D. van der Mei** is the leader of the research cluster Probability, Networks and Algorithms (PNA), the leader of the research theme Societal Logistics within CWI, and a (part-time) full professor at the VU University, Amsterdam. Before going to academia, he has been working for over a decade as a consultant and researcher in the ICT industry, working for PTT, KPN, AT&T Bell Labs and TNO ICT. He has been a member of the editorial board of Performance Evaluation and the AEUE Journal on Electronics and Communications, and is currently serving the Editorial board of the journal ICST Transactions on Network Optimization and Control. He is a co-founder and board member of the national expertise centre E-Quality on performance and Quality of Service (QoS) issues in ICT, which aims to enhance the transfer of knowledge transfer between the Dutch ICT industry and the leading knowledge institutes in the field of Quality of service (QoS) in the Netherlands. He is also a co-founder and general chair of the recently recognized ICT Innovation Platform (IIP) "Vital ICT Infrastructures", a platform for exchanging knowledge and experience in the area of QoS of ICT systems between academia a wide range of ICT companies. He is a co-founder and board member of the recently recognized Dutch Mathematics cluster Stochastics—Theoretical and Applied Research (STAR). His research interests include performance modeling and scalability analysis of ICT systems, logistics, grid computing, revenue management, sensor networks and queueing theory. He is the (co-)author of over 90 papers in journals and refereed proceedings.

**Dennis Roubos** (1982) received his M.Sc. degree in Business Mathematics and Informatics from the VU University Amsterdam, The Netherlands. In 2010 he received his Ph.D. degree from the same university for his Ph.D. research on "The application of Approximate Dynamic Programming techniques". His research interests are mainly, but not limited to, Markov decision processes, and in particular, Approximate Dynamic Programming and Reinforcement Learning. His interests are both on the theoretical part as well on the application part, and he likes to apply optimization techniques in call centers, health care, and computer science.

**Frank J. Seinstra** is an assistant professor in the Department of Computer Science at VU University Amsterdam, working in the High Performance Distributed Systems research group headed by Prof. H.E. Bal. In 2003, he received his Ph.D. degree from the University of Amsterdam for his research on "User Transparent Parallel Image Processing". His current research efforts are focused on the design of programming models for large-scale heterogeneous parallel and distributed systems.

**Henri E. Bal** is a full professor in the Department of Computer Science at VU University Amsterdam, where he heads the High Performance Distributed Systems research group. His research interests include parallel and distributed computing, mobile computing, programming languages, and compiler design. He is the initiator of several high-performance distributed systems for computer science research in The Netherlands (e.g. DAS-3 and DAS-4). He is the author of three books on compiler design, programming languages, and distributed systems programming.