On the Optimization of Resource Utilization in Distributed Multimedia Applications

R. Yang*†, R.D. van der Mei*†, D. Roubos*, F.J. Seinstra*, and G.M. Koole*

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

†Centre for Mathematics and Computer Science
Kruislaan 413, 1098 SJ, Amsterdam, The Netherlands

***Abstract*—The application and research area of Multimedia Content Analysis (MMCA) considers all aspects of the automated extraction of new knowledge from large multimedia data streams and archives. In recent years, there has been a tremendous growth in the MMCA application domain (for real-time and off-line execution scenarios alike), and this growth is likely to continue in the near future. Multimedia applications operated in a real-time environment pose very strict requirements on the obtained processing times, while off-line applications have to perform within 'tolerable' time frames. To meet these requirements, large-scale multimedia applications typically are being executed on Grid systems consisting of large collections of compute clusters. For optimized use of resources, it is essential to determine the optimal number of compute nodes per cluster, properly dealing with the perceived computation versus communication ratio. This ratio generally depends on the characteristics of the application at hand, and on the software and hardware specifics of the computational environment.**

Motivated by these observations, in this paper we develop a simple and easy-to-implement method to determine the "optimal" number of parallel compute nodes. The method is based on the classical binary search method for non-linear optimization, and does not depend on the, usually unknown, specifics of the system. Extensive experimental validation on a real distributed system shows that our method is indeed highly effective.

I. Introduction

Today, multimedia data is rapidly gaining importance along with recent deployment of publicly accessible digital television archives, surveillance cameras in public locations, and automatic comparison of forensic video evidence [18]. In a few years, computerized access to the content of multimedia data will be a problem of phenomenal proportions, as digital video may produce high data rates, and multimedia archives steadily run into petabytes (10^{15}) of storage space. As individual compute clusters can not satisfy the increasing computational demands, distributed supercomputing on large collections of clusters (Grids) is rapidly becoming indispensible.

Problems in Multimedia Content Analysis (MMCA) often must work under strict time constraints. For example, to avoid delays in queues of people waiting, results of iris-scans should be presented within a period of no more than 5 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. To (automatically) optimize the resource utilization of such applications executing on large collections of compute clusters, efficient methods must be available that determine the optimal number of nodes per compute cluster. This optimization problem generally depends on the application at hand, and on the specifics of the computation environment (e.g., network characteristics, CPU power memory, I/O). In this context, it is essential to properly balance the following trade-off: if the number of compute nodes is too low, then the processing power is insufficient to meet strict processing-time requirements of real-time applications; if the number of compute nodes is too high, the parallelisation overhead will cause a degradation of the computational performance. Hence, there is a urgent need for *simple* and *easily implementable*, yet *effective* methods (in terms of the number of evaluation steps), to determine the optimal level of parallelism. Also, the method should be *adaptive* to system variation.

Saavedra-Barrera et al. [12] 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 is that system variance is almost completely ignored. For applications working on extensive dense data fields (e.g., image data structures) this is too crude a restriction as variations in the hit ratio of caches and system interrupts often have a significant impact on performance [4], [13].

Many other performance estimation techniques that incorporate more detailed behavioral abstractions relating to the major components of a computer system [6], [8], however, need tens, if not hundreds, of platform-specific machine abstractions to obtain truly accurate estimations. Consequently, the essential requirements of simplicity and applicability are not satisfied. To overcome this problem, Seinstra et al. [14] have designed a new model for performance estimation of *parallel* image 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 processing application to find the optimal number of compute

Table I
OVERVIEW OF DAS-3 CLUSTER SITES

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

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 does not meet our requirements, as obtaining accurate performance values for all possible parameter combinations is both costly and complex.

In this paper, we propose a simple method to determine the "optimal" level of parallelism, in which the number of evaluation steps is small. In this context, experimental observations for realistic, large-scale problems in multimedia content analysis have revealed three important optimization properties. First, the optimal number of parallel compute nodes is generally found to be a power of 2, i.e. of the form 2^m for some $m = 0, 1, \ldots$. This observation is important because it leads to a dramatic reduction of the set of possible solutions. For example, if the number of available compute nodes is m_{max}, the size of solution space is reduced from m_{max} (i.e., the number of elements in the index set $\{1, \ldots, m_{max}\}$) to $\lfloor log_2(m_{max}) \rfloor$ (i.e., the number of elements of the set $\{2^0, 2^1, \ldots, 2^K\}$ where $K = \lfloor log_2(m_{max}) \rfloor$). 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 $S(L)$, 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.

Based on these 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 $5 - 10\%$). To validate the effectiveness of the proposed method, we have performed extensive experimentation on a realistic distributed system (DAS-3 [9], [10]) for both real-time and off-line applications. The results show that our method is indeed highly effective.

In practice, running CPU-intensive applications in large-scale distributed computing environments typically consists of two phases: (1) an *initialisation 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. We emphasize that the method proposed in this paper is to be used during the initialisation phase.

The remainder of this paper is organized as follows. Section II presents the experimental setup, and describes our example applications. In Section III our new modeling approch is formulated. Section IV discussed our experimental results. Finally, in Section V we present our conclusions and address topics for further research.

II. EXPERIMENTAL SETUP

In a Grid environment, resources have different capacities and many fluctuations exist in load and performance of geographically distributed nodes [1]. As the availability of resources and their load continuously vary with time, the repeatability of the experimental results is hard to guarantee under different scenarios in a real Grid environment. Also, the experimental results are very hard to collect and to observe. Hence, it is wise to perform experiments on a testbed that

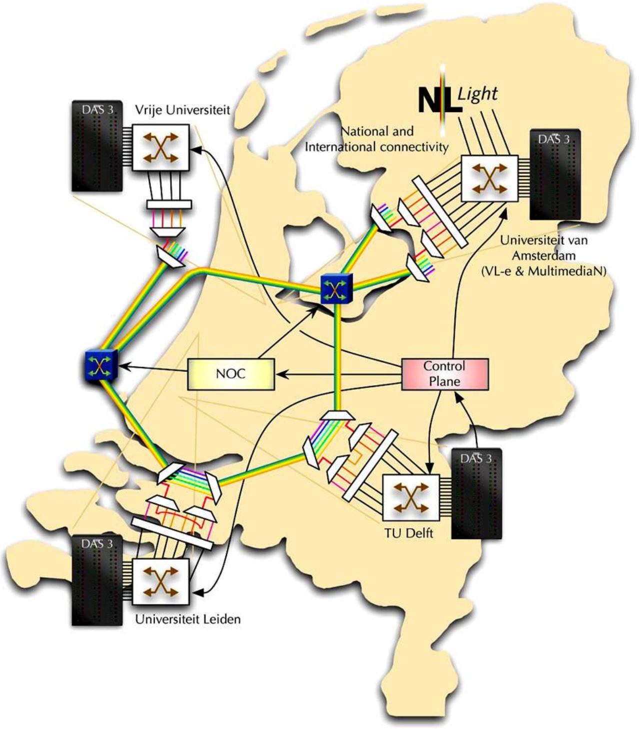
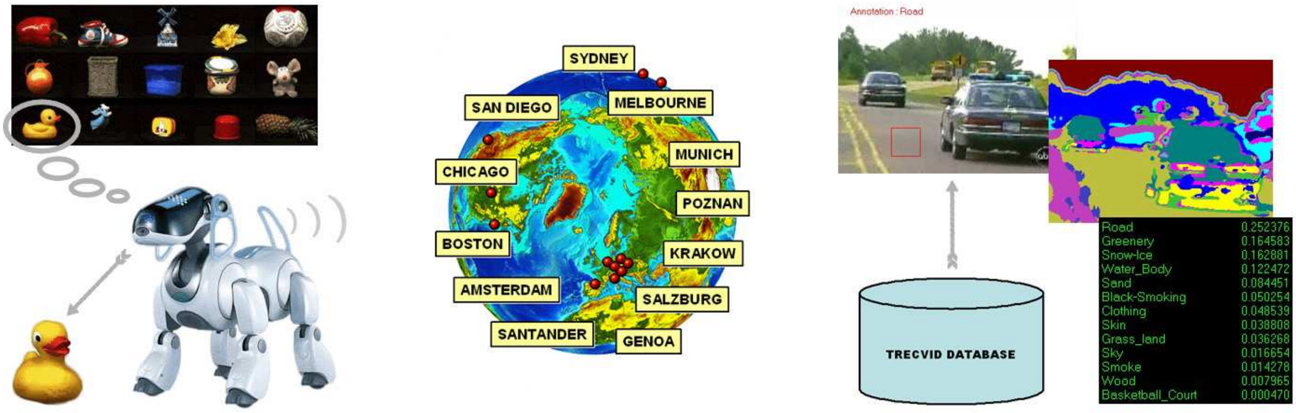


Figure 1. The Distributed ASCI Supercomputer 3 with the StarPlane wide-area optical interconnect.



Road	0.252376
Greenery	0.164583
Snow-Ice	0.162881
Water_Body	0.122472
Sand	0.084451
Black-Smoking	0.050254
Clothing	0.048539
Skin	0.038808
Grass_land	0.036268
Sky	0.016654
Smoke	0.014278
Wood	0.007965
Basketball_Court	0.000470

Figure 2. 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. The off-line application constitutes our TRECVID system.

contains the key characteristics of a Grid environment on the one hand, and that can be managed easily on the other hand. To meet these requirements, we perform all of our experiments on the recently installed DAS-3 Grid test bed [10].

The Distributed ASCI Supercomputer 3 (DAS-3, see Table I and Figure 1) is a five-cluster wide-area distributed system, with individual clusters located at four different universities in The Netherlands: Vrije Universiteit Amsterdam (VU), Leiden University (LU), University of Amsterdam (UvA), and Delft University of Technology (TUD). The MultimediaN Consortium (UvA-MN) also participates with one cluster, 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 [11]), causing DAS-3 sometimes to be referred to as ”the world’s fastest Grid” [2].

A. Example applications

In our experiments, we use the DAS-3 system 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 [16] (see Figure 2). 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 tradeoff between specificity of recognition (e.g., discrimination between different faces) and invariance (e.g., to shadows, or to differently colored light sources). Due to the rapid increase in the size of multimedia repositories of ’known’ objects [3], 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 Figure 2) indispensible.

The “TRECVID” application represents a multimedia computing system that has been applied successfully in the 2004, 2005, and 2006 editions of the international NIST TRECVID benchmark evaluation for content-based video retrieval [5], [17]. 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 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. Our “TRECVID” application is computationally intensive; for thorough analysis it easily requires about 16 seconds of processing per video frame on the fastest sequential machine at our disposal [15]. 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 Parallel-Horus software architecture, that allows programmers to write parallel and distributed multimedia applications in a fully sequential manner [16]. 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. Every received video frame is scattered by this server into many pieces over the available compute nodes. Normally, each compute node receives one partial video frame for processing. The computations at all compute nodes take place in parallel. When the computations are completed, the partial results are gathered by the communication server again and the final result is returned to the client. In this paper, the time to process a single video frame in this manner is defined as the *service processing time*.

III. Method formulation

This section describes our newly proposed modeling approach in detail. The approach is based on the results of extensive experimentation performed on DAS-3 (see Section II).

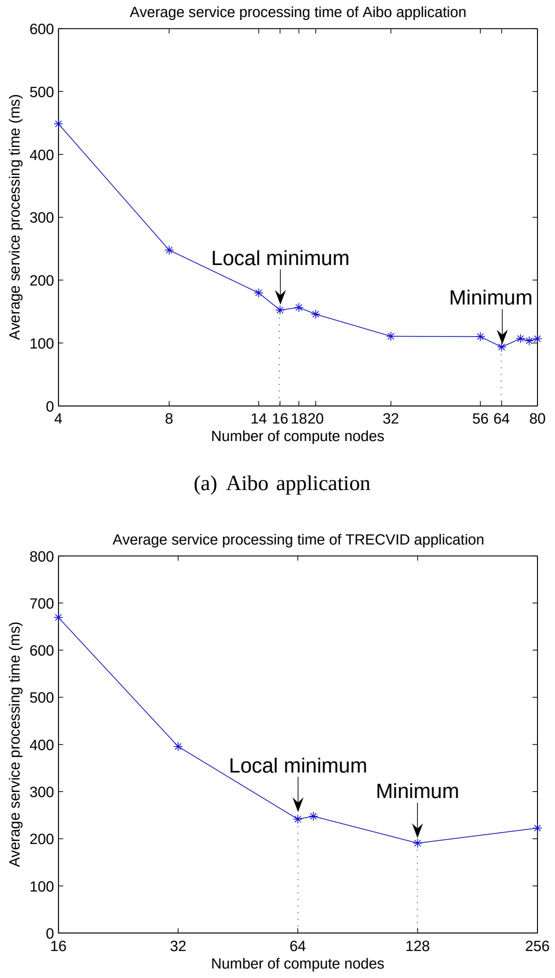
In our example applications, video frames are being processed on a per-cluster basis, using a varying number of compute nodes on each cluster, each consisting of multiple CPUs. The compute cluster (or *service*) processing time is defined as a function $S(L, n)$ of the number of compute nodes $L = 1, \ldots, m_{max}$ and the number of CPUs per node $n = 1, 2, \ldots$. 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 the function $S(L, n)$ attains its minimum. In this context, it is important to note that the set of possible combinations (L, n) may be very large, and that in practice, finding the optimum $(\hat{L}, \hat{n})$ may be very time consuming. Therefore, our goal is to develop a simple but effective heuristic method to obtain a nearly-optimal solution within a short time frame. To this end, we first discuss a number of observations that we collected during our extensive experiments, leading to a dramatic reduction of the set of possible value of (L, n). Subsequently, the method to approach the optimal (L, n) is described in detail.

A. Reduction of solution space

Many combinations (L, n) lead to the same total number of $T = L \cdot n$ of CPUs. The following observations, made for our particular example applications, rule out many possibilities:

1) The total number of CPUs should be a power of 2: In our experiments, we consistently observed that the optimal number of CPUs is found to be a power 2. For example, Figure 3 shows the average processing times for our two example applications: (a) Aibo, and (b) TRECVID. The results show that both *local* and *global* minima are consistently found when the total number of CPUs is a power of 2. This observation leads to dramatic reduction of the set of possible solutions. Namely, if the number of available compute nodes is L_{max}, the number of available CPUs in each compute node is n_{max}, then the solution set is reduced to $\mathbb{X} := \{(2^p, 2^q), p = 0, \ldots, P, q = 0, \ldots, Q\}$, where $P := \lfloor log_2(L_{max}) \rfloor$, $Q := \lfloor log_2(n_{max}) \rfloor$.

2) Using more compute nodes, yet less CPUs per node, is better: Another important observation from our experimental results is that for the same total number of CPUs $T = L \cdot n$, using more compute nodes L and fewer CPU's per node n provides better performance. That is, for the same total number of CPUs $T = 2^m$, where the solution set should be $\mathbb{X} := \{(2^p, 2^q), p + q = m\}$, among them, q should be as small as possible. This observation is illustrated by Figure 4, where we consider the case $T = 64$ for three combinations $(L, n) \in \{(64, 1), (32, 2), (16, 4)\}$; the results show that the combination (64,1) has better performance than (32,2) and (16,4). This is explained by the fact that the compute nodes in DAS-3 are linked by a fast local Myrinet interconnect, whereas the CPUs within a single node communicate over a shared memory bus, which is less efficient.



(a) Aibo application

(b) TRECVID application

Figure 3. Average service processing time v.s. number of compute nodes.

Based on these observations, the space set can be reduced drastically. For instance, for a system having 85 nodes and 4 CPUs per node, the reduced solution space is $\mathbb{X} = \{(2^p, 1), p = 0 \cdots 6\} \cup \{(64, 2), (64, 4)\}$.

In a general form, to determine the optimal number of compute nodes and CPUs per node, the solution space is reduced to the combinations $\mathbb{X} = \{(2^p, 1), p = 0 \cdots P\} \cup \{(2^P, 2^q), q = 1 \cdots Q\}$, where $P := \lfloor log_2(L_{max}) \rfloor$, $Q := \lfloor log_2(n_{max}) \rfloor$. 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.

B. Steps to approach the optimal (L, n)

1) Engineering knee: 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, Figure 5

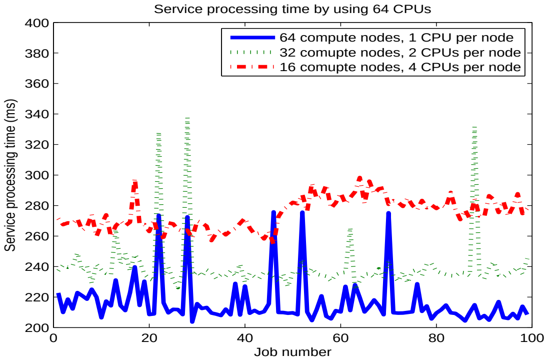
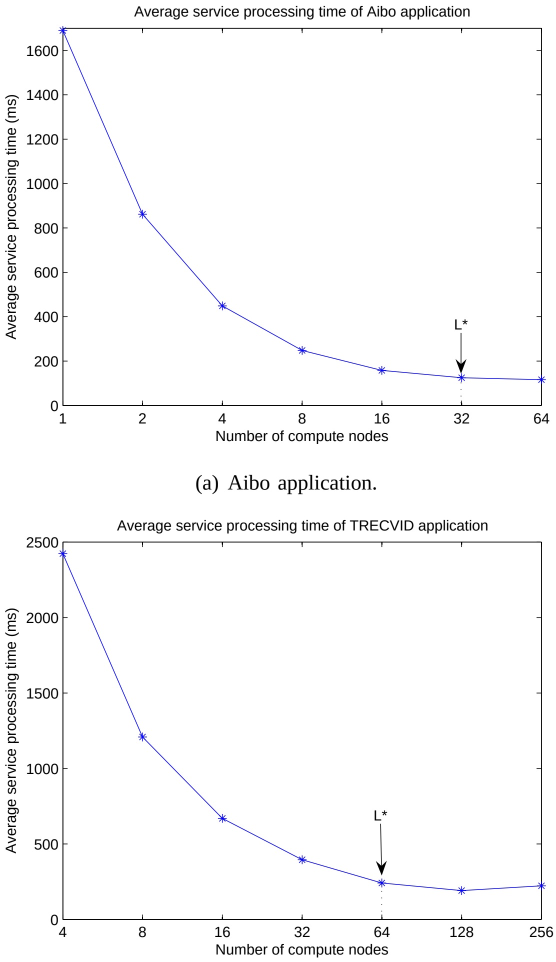


Figure 4. More compute nodes and less CPUs per node is better

shows the average service processing times for the Aibo- and TRECVID-applications for different values of $L = 2^m$. In both cases, 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.



(a) Aibo application.

(b) TRECVID application

Figure 5. Engineering knee of Aibo and TRECVID applications

C. LDS method

To find the engineering knee L^*, we have developed an *Logarithmic dichotomy search* (LDS) method. The LDS method follows the idea of a well-known conventional binary search (CBS) algorithm [7] 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 are defined in Section III-A2. The LDS strategy selects the median element in the set $\mathbb{X}$, denoted by e_{Mid}. Define ϵ as the desired 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 keep only the elements $(e_{\text{Mid+1}}, \ldots, e_K)$. If $\frac{S(e_{\text{Mid}})-S(e_{\text{Mid+1}})}{S(e_{\text{Mid}})} \geq \epsilon$ 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. The pseudo code for our LDS method for solution space $\mathbb{X}$ is given in Algorithm 1.

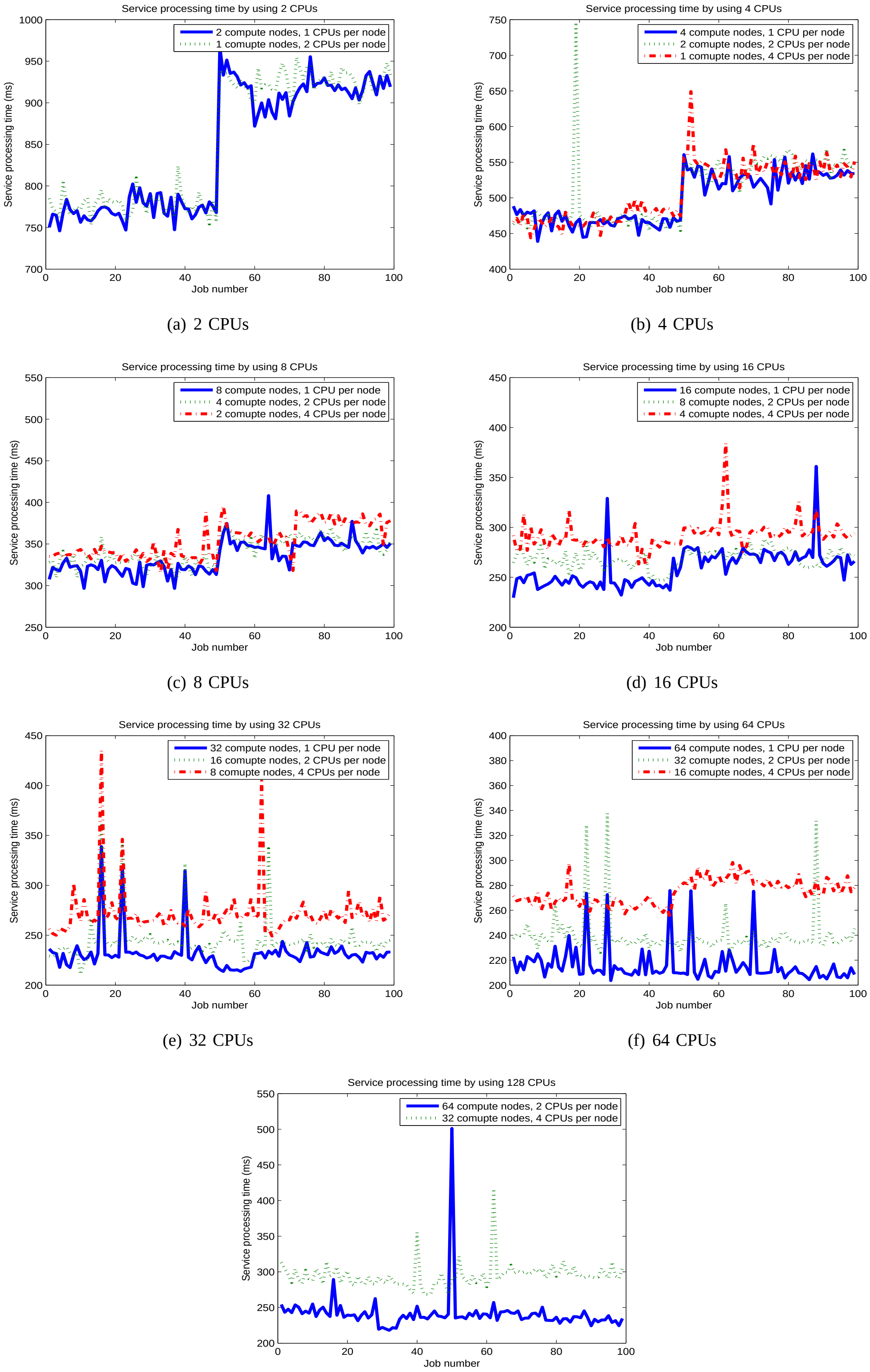
```
Low := 0
High := K
While (Low $<$ High) {
        Mid := $\left\lfloor \frac{\text{Low}+\text{High}}{2} \right\rfloor$
        if $S(e_{\text{Mid}}) < \frac{S(e_{\text{Mid+1}})}{1-\epsilon}$ {High = Mid;}
        else {Low = Mid +1;}
        end if;
}
Optimal number of compute nodes := High.
```

Algorithm 1: Pseudo code of LDS strategy

IV. Numerical Results

Even though our LDS method has been applied successfully for all DAS-3 clusters, the detailed experiments presented here have been carried out on the largest cluster at the Vrije Universiteit, that consists of 85 compute nodes with 4 CPUs per node. In this section, we show the numerical results of the average service processing times versus a varying total number of compute nodes. In addition, the simplicity of LDS stratey 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 $\mathbb{O} = \{(L, n), L \in [1, \cdots, 85]$ and $n \in [1, \cdots, 4]\}$. 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 Figure 6. In this figure we notice that for small numbers of CPUs (say, ≤ 16), the service processing time is largely independent of



(a) 2 CPUs

(b) 4 CPUs

(c) 8 CPUs

(d) 16 CPUs

(e) 32 CPUs

(f) 64 CPUs

(g) 128 CPUs

Figure 6. Service processing time of the Aibo application using different total amount of CPUs

Table II
AVERAGE SERVICE PROCESSING TIME OF TRECVID APPLICATION

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

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 a 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 Tabel II. For this application we have a similar conclusion: more compute nodes and less CPUs per node provides the best performance results.

In Section III-A1, 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 9 possible solutions, where $\mathbb{X} = \{(2^i, 1)$ where $i \in [0, \cdots, 6]\} \cup (64, 2) \cup (64, 4)$. Based on the $\mathbb{X}$, we apply our LDS method to find the minimum value after $\lfloor log_2 9 \rfloor = 3$ steps. We use Table III to explain the three steps taken in AIBO application when ϵ = 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 $[0, 1, \cdots, 8]$. Then the LDS method is applied. In the first step, we have $Low = 0$ and $High = 8$. Because $Low < High$, we have

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

Therefore, we measure the service processing time using $2^4 = 16$ and $2^5 = 32$ compute nodes and 1 CPU per node. The related average service processing times are shown in the first row of Table IV. Because the relative improvement using 32 compute nodes compared to 16 compute nodes is 0.27 ($> \epsilon$), we conclude that 16 compute nodes is not optimal. Therefore, we continue saerching 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 ϵ. 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 ϵ. 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 ϵ (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 Table IV and Table V, respectively. The found optimal L^* for both applications for different values of ϵ are listed in Table VI. In this table, we notice that with larger ϵ , the found L^* remains the same or decreases.

Table III
THREE STEPS TO APPROACH THE OPTIMAL (L, n)

Step	Low	High	Mid	compare to	relative improvement	Action
1	0	8	4	5	0.27	keep high half
2	5	8	6	7	-0.15	keep low half
3	5	6	5	6	0.15	finish, return index 6

Table IV
AVERAGE SERVICE PROCESSING TIME OF AIBO APPLICATION

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

Table V
AVERAGE SERVICE PROCESSING TIME OF TRECVID APPLICATION

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

Table VI
THE VALUE OF ENGINEERING KNEE

(a) Aibo application

ϵ	L^*
0.1	64 (64,1)
0.2	32 (32,1)
0.3	16 (16,1)

(b) TRECVID application

ϵ	L^*
0.1	128 (64,2)
0.2	128 (64,2)
0.3	64 (64,1)

As shown above, we notice that our method is very simple to implement. Besides this, it is very effecitve because of the small number of steps required to find the optimal number of compute nodes. In addition, by varying ϵ, 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.

V. CONCLUSIONS AND FURTHER RESEARCH

In this paper we explored the relation between the service processing time and the number of compute nodes using different number of CPUs. In our experiments, we observed that there exists a threshold value L^* (referred to as the *engineering knee*) 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^*, first we reduce the possible solution set. Then we apply our LDS method to found L^* in a fast way. Extensive validation has shown that our method is highly 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 also takes into account system variation.

The work described in this paper is part of a much large strive to bring the benfits of high-performance computing to the multimedia computing 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 continouosly 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 generally 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 situations, such as iris scans performed at airports, the amount of data to be analyzed simply depends on the time schedule of arriving airplanes.

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.

References

[1] 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.

[2] M. Feldman, “Grid Envy”, *ClusterVision News*, pp. 6–7, 2006.

[3] J.M. Geusebroek, G.J. Burghouts, and A.W.M. Smeulders, “The Amsterdam Library of Object Images,” International Journal of Computer Vision, vol. 61, no. 1, pp. 103–112, 2005.

[4] C. Grelck, “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, 2000.

[5] A. Hauptman et al, ”Informedia at TRECVID 2003: Analyzing and Searching Broadcast News Video”, In *Proceedings of TREC 2003*, Gaithersburg, USA, November 2003.

[6] R. Jain, *The Art of Computer Systems Performance Analysis*. John Wiley & Sons, 1991.

[7] D. Knuth, *The Art of Computer Programming, Volume 3: Sorting and Searching*. Addison Wesley Longman Publishing Co., Inc. Redwood City, CA, USA, 1998.

[8] B. Maggs, L. Matheson, and R. Tarjan, “Models of parallel computation: a survey and synthesis,” *in Proc. International Conference on System Sciences*, vol. 2, pp. 61–70, 1995.

[9] Online, “http://www.asci.tudelft.nl/”, 2007.

[10] Online, “http://www.cs.vu.nl/das3/,” 2007.

[11] Online, “http://www.starplane.org/”, 2007.

[12] R. Saavedra-Barrera, A. Smith, and E. Miya, “Machine characterization based on an abstract high-level language machine,” *IEEE Trans. Computers*, vol. 38, no. 12, pp. 1659–1679, 1989.

[13] K. Schutte and G. van Kempen, “Optimal Cache Usage for Separable Image Processing Algorithms on General Purpose Workstations,” *IEEE Transactions on Signal Processing*, vol. 59, no. 1, pp. 113–122, 1997.

[14] F.J. Seinstra, D. Koelma, and J.M. Geusebroek, “A Software Architecture for User Transparent Parallel Image Processing,” *Parallel Computing*, vol. 28, nos. 7-8, pp.967–993, 2002.

[15] F.J. Seinstra, C.G.M. Snoek, D. Koelma, J.M. Geusebroek, and M. Worring, ”User Transparent Parallel Processing of the 2004 NIST TRECVID Data Set”, In *Proceedings of the International Parallel & Distributed Processing Symposium (IPDPS 2005)*”, Denver, Colorado, USA, April 2005.

[16] F.J. Seinstra, J.M. Geusebroek, D. Koelma, C. Snoek, M. Worring, and A. Smeulders, “High-Performance Distributed Image and Video Content Analysis with Parallel-Horus,” *IEEE Multimedia*, vol. 14, no. 4, pp. 64–75, 2007.

[17] C.G.M. Snoek et al, ”The MediaMill TRECVID 2005 Semantic Video Search Engine”, In *Proceedings of the 3rd TRECVID Workshop*, Gathersburg, USA, November 2005,

[18] C.G.M. Snoek, M. Worring, J.M. Geusebroek, D.C. Koelma, F.J. Seinstra, and A.W.M. Smeulders, “The Semantic Pathfinder: Using an Authoring Metaphor for Generic Multimedia Indexing,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 28, no. 10, pp. 1678–1689, 2006.