High-Performance Distributed Video Content Analysis with Parallel-Horus

Frank J. Seinstra, Jan-Mark Geusebroek, Dennis Koelma, Cees G.M. Snoek, Marcel Worring, and Arnold W.M. Smeulders
University of Amsterdam

As the world uses more digital video that requires greater storage space, Grid computing is becoming indispensable for urgent problems in multimedia content analysis. Parallel-Horus, a support tool for applications in multimedia Grid computing, lets users implement multimedia applications as sequential programs for efficient execution on clusters and Grids, based on wide-area multimedia services.

Information—scientific, industrial, or otherwise—is increasingly composed of multimedia items—that is, a combination of pictorial, linguistic, and auditory data. Smooth, efficient, computerized access to such information is a tremendous challenge.

The automatic deduction of semantics from multimedia data requires sophisticated techniques for data structuring, transformation, analysis, classification, and learning. And the nature of information creation is such that most of the data is irrelevant to a specific question. Therefore, the challenge is to discover and interpret tiny fractions of useful information in a whirlwind of meaningless noise. As the storage and connectivity of multimedia data increases, automatic multimedia content analysis (MMCA) is becoming an ever-more important area of research.

MMCA considers all aspects of the automated extraction of knowledge from large multimedia data sets. From these, image and video archives contain the bulk of all data and, thus, form the biggest challenge. Research in this area recently made a giant step forward with the introduction of machine-learned multimedia analysis, yielding automatic categorization by visual object types, such as human faces, bicycles, and interviews.[1] Fundamental MMCA research questions include the following:

- Can we automatically find genres and subgenres in images from a statistical evaluation of large image sets?
- Can we automatically learn to find objects in images and video streams from partially annotated image sets?

Solutions to these questions are essential in a myriad of applications. The Netherlands Forensic Institute (NFI), for example, has an urgent need for detecting objects and individuals in video streams obtained from surveillance cameras. Similarly, the Netherlands Institute for Sound and Vision (Beeld & Geluid) aims to store digitized television broadcasts in large repositories, generating heavy demands on accessibility. Also, the need to identify materials and land-cover classes in terabytes of hyperspectral data obtained from NASA satellites is one of many notable examples from the scientific literature.[2]

Despite several obstacles with Grid computing (for more details, see the sidebar "High-Performance Multimedia Processing and Grid Computing"), if we ever hope to meet the urgent demands for handling large amounts of multimedia data, we must turn to Grid computing as a solution. With this in mind, we developed and integrated a fully user-transparent programming model (called Parallel-Horus) with a Grid-execution model based on *wide-area multimedia services*.

When we say wide-area multimedia services, we mean high-performance, data-parallel multimedia functionality that can be invoked from sequential applications running on a desktop machine. We evaluate our approach by applying it to two state-of-the-art multimedia applications.

Parallel-Horus

Parallel-Horus[3-6] is a cluster programming library that lets programmers implement parallel multimedia applications as fully sequential programs. The Parallel-Horus library consists of commonly used multimedia data types and associated operations, implemented in C++ and an additional message passing system (MPI). The library's API is made identical to that of an existing sequential

1070-986X/07/$25.00 © 2007 IEEE

Published by the IEEE Computer Society

High-Performance Multimedia Processing and Grid Computing

Digital video often produces data at extremely high data rates, causing multimedia archives to steadily run into petabytes of storage space. For example, the Netherlands Institute for Sound and Vision (Beeld & Geluid), has 750,000 hours of high-resolution TV data waiting to be archived (see http://www.radionetherlands.nl/features/media/061207beg).

Distributed surveillance cameras generate even larger quantities of data. Following the terrorist attacks in the London Underground, for example, the British police had to investigate video footage obtained from over 80,000 closed-circuit TV (CCTV) cameras.[1] And NASA sends to Earth more than 850 Gbytes of hyperspectral image data every day for analysis and processing.[2] Clearly, multimedia content analysis (MMCA) problems are generally colossal, often in the petascale range.

Recent results from image analysis show that access to the content of large data sets is a difficult problem.[3] One way to deal with this is to apply approximate algorithms, albeit at the expense of accuracy and the loss of useful access results.

A better solution is to exploit the distribution of data and computation over computing networks. Also, in many emerging MMCA problems, generation, processing, storage, indexing, querying, retrieval, and visualization of multimedia data are integrated aspects, all taking place at the same time and, potentially, at multiple *administrative domains*. The only way for us to satisfy this increasing need for computing and storage resources is to adopt techniques from Grid computing.

In recent years, Grid computing[4] has become an active area of research. Researchers have put tremendous efforts into installing heterogeneous wide-area systems, as well as developing Grid middleware and programming environments and Grid-enabled applications. Essentially, all of these efforts are performed in pursuit of the single, foremost goal of the Grid: to provide inexpensive and easy-to-use "wall socket" computing over a distributed set of resources.

Despite this progress, Grid systems still lack the functionality needed for extensive use by nonexperts. The difficulty of employing Grids has been acknowledged in the field, however, giving rise to component frameworks that come closer to realizing seamless integration across multiple computing platforms. Especially with the advent of unified interfaces tailored to the needs of Grid application programmers, such as Ibis[5] and the Grid Application Toolkit (GAT),[6] Grids are slowly becoming as accessible as high-performance cluster computers are today.

One fundamental problem remains, however. Although many multimedia applications are ideal for parallel execution,[2,7] most multimedia researchers don't apply high-performance computing at all—not even on cluster computers, which are much easier to use than large-scale Grid systems. This is primarily because no programming tools exist that can effectively help nonexperts in writing high-performance multimedia applications.[7] Hence, the key to effective support for high-performance multimedia computing lies in the availability of a familiar (that is, user transparent) programming model that hides the difficulties of parallel implementation from its users. To further stimulate the use of widely distributed Grid resources in the multimedia community, this programming model must be supported by an easy-to-use Grid execution model as well.

References

1. N. Dean, "Bombers Staged Dry Run Before London Attacks," *The Independent*, online edition, 20 Sept. 2005; http://news.independent.co.uk/uk/crime/article313884.ece.
2. A. Plaza et al., "Commodity Cluster-Based Parallel Processing of Hyperspectral Imagery," *J. Parallel Distributed Computing*, vol. 66, no. 3, 2006, pp. 345-358.
3. C.G.M. Snoek et al., "The Semantic Pathfinder: Using an Authoring Metaphor for Generic Multimedia Indexing," *IEEE Trans. Pattern Analysis and Machine Inteligence*, vol. 28, no. 10, 2006, pp. 1678-1689.
4. D. Butler, "Tomorrow's Computing Today," *Nature*, vol. 422, 2003, pp. 799-800.
5. R. van Nieuwpoort et al., "Ibis: A Flexible and Efficient Java-Based Grid Programming Environment," *Concurrency and Computation: Practice and Experience*, vol. 17, nos. 7-8, 2005, pp. 1079-1107.
6. G. Allen et al., "The Grid Application Toolkit: Towards Generic and Easy Application Programming Interfaces for the Grid," *Proc. IEEE*, vol. 93, no. 3, 2005, pp. 534-550.
7. F.J. Seinstra et al., "User Transparency: A Fully Sequential Programming Model for Efficient Data Parallel Image Processing," *Concurrency and Computation: Practice and Experience,* vol. 16, no. 6, 2004, pp. 611-644.

library: Horus. We integrated the parallel functionality with Horus such that the original sequential code remains unchanged. This approach offers a major advantage, because then the sequential Horus library's most important properties (such as maintainability, extensibility, and portability) to a large extent transfer to Parallel-Horus as well.

Similar to other frameworks, Horus is based on the abstractions of Image Algebra,[7] a mathematical notation for specifying image and video processing algorithms. Image Algebra defines a small set of algorithmic patterns that cover the bulk of commonly applied image and video processing functionality. Horus implements any operation that maps onto the functionality as provided by such a pattern by instantiating it with the proper parameters, including the function to be applied to the individual data elements.

Results have shown the feasibility of the Parallel-Hurus approach, resulting consistently in optimal data parallel performance.

Horus includes patterns for every commonly used operation, such as unary and binary pixel operation, global reduction, neighborhood operation, generalized convolution, and geometric transformations. Recently, additional patterns have been introduced, including iterative and recursive neighborhood operations. Current extensions include patterns for operations on large data sets and patterns on increasingly important data structures, such as feature vectors obtained from earlier calculations on image and video data.

For reasons of efficiency, all Parallel-Horus operations are capable of adapting to the performance characteristics of a parallel machine at hand, such as being flexible in the partitioning of data structures.[4,5] Moreover, we realized that it's insufficient to consider parallelizing library operations *in isolation*. Therefore, we extended the Parallel-Horus library with a runtime approach for communication minimization, which automatically parallelizes a fully sequential program by inserting communication primitives and additional memory management operations whenever necessary.

This approach, called *lazy parallelization*,[6] is based on a finite-state-machine (fsm) specification to exploit and integrate the inherent data parallelism present in the algorithmic patterns. One of two fsm ingredients is a set of states, each corresponding to a valid internal representation of a distributed data structure at runtime. The other is a set of state transition functions, each of which defines how to transform a valid data structure state into another representation.

As each algorithmic pattern has a set of allowed input states for each data structure used as input to the operation and a set of potential output states for each data structure produced as output by the operation, the system minimizes communication efforts by resolving data structure state inconsistencies via state transition functions. With many possible runtime execution scenarios, the simplest is then as follows: An input data structure is first scattered throughout the system, after which the sequence of required calculations is performed at each node using local (partial) data, performing additional inter-node communication only when necessary. Then, if the system needs to write a distributed data structure out to file, it gathers this information into a single compute node that performs the I/O. We'll incorporate future additions to Parallel-Horus internally in the same manner, making extensions to the fsm definition whenever needed.

Applying our techniques in other (existing) multimedia processing libraries likewise would require the identification of algorithmic patterns, data structure states, and state transitions.[4,6] Independently, others (such as Skipper[8] or EasyPipe[9]) share this conclusion—that it's essential to base a full library implementation on code fragments that we can combine at will. Parallel-Horus differs from such related work in that it's the only system in which *all* parallelization and optimization is fully hidden from the user.

Results for realistic multimedia applications have shown the feasibility of the Parallel-Horus approach, resulting consistently in optimal data parallel performance with respect to the abstraction level of message passing programs.[3] Notably, Parallel-Horus was applied in the 2004, 2005, and 2006 National Institute of Standards and Technology's yearly Conference on Video Retrieval (NIST Trecvid) benchmark evaluations for content-based video retrieval,[1] playing a crucial role in achieving our top-ranking results in a field of strong international competitors (see also the "Evaluation" section).

Prototypical code (in C and MPI) of an earlier proof-of-concept implementation of Parallel-Horus, as well as of several example image processing applications, is available at http://www.science.uva.nl/~fjseins/ParHorusCode/.

Services-based multimedia Grid computing

To arrive at a system that integrates a user-transparent programming model with an efficient and easy-to-use Grid execution model, we combined the Parallel-Horus programming model with the now popular services-based approach to wide-area computing.[10] A services-based approach coincides well with the most

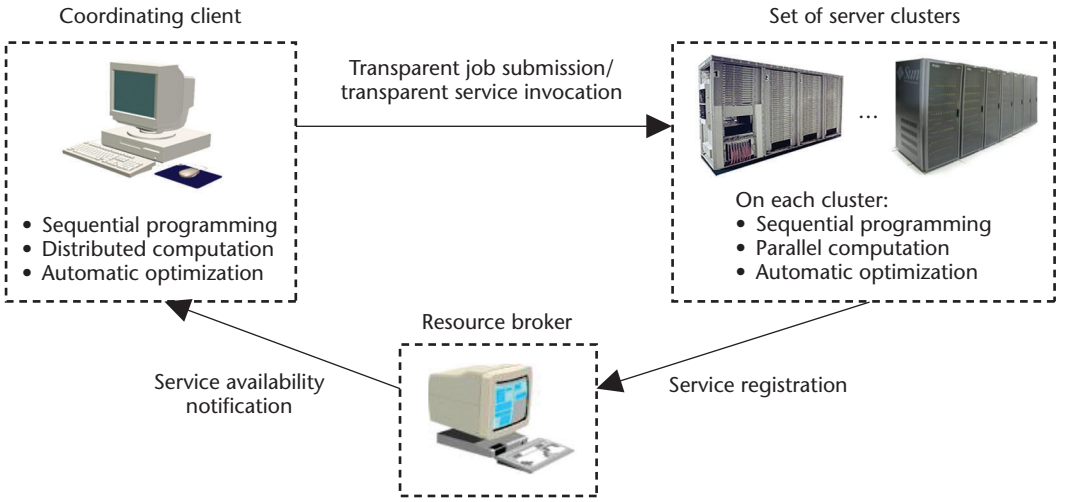


Figure 1. Collaboration between a client, a set of multimedia compute servers, and a resource broker. Each cluster executes a server program, which the system implements in a fully sequential manner. Parallel-Horus parallelizes and optimizes the server programs automatically at runtime. Services may be made available by third parties, or may be started transparently from the client (by using the Grid Application Toolkit, or GAT,[11] for example). Services register themselves at the resource broker. In turn, the client obtains registration information and server capability information from the resource broker. The client applies asynchronous services invocation for distributed execution.

common types of multimedia applications, in which distributing tasks, as well as parallelizing them, is relevant (examples include video processing, processing of large multimedia archives, and parameter sweeps over multimedia data).

Specifically, we designed a client–server-based framework extended with a resource broker implementation for resource registration and resource availability notification (see Figure 1). We provide three logical components (the client, server, and resource broker) via three APIs that allow for services provisioning, services calling, and services registration. With these APIs, converting Parallel-Horus code to a client and corresponding server implementation is straightforward.

On the server side, the user needs to indicate that the I/O is to take place through send and receive ports, which is as simple as indicating file-based I/Os. This is identical at the client side, with the additional requirement that a response routine must be implemented that's called every time the system receives result data from a service.

We designed the system to hide resource detection and notification from the user completely. This helps us create dynamic systems of distributed multimedia services in a matter of minutes, in which clients and servers can participate at will, without any parallelization and distribution effort from the user. We discuss state-of-the-art applications that we implemented as dynamic system examples in the following sections.

A services-based approach isn't new in the field of multimedia.[12,13] The most important difference between our work and related work, however, is that Parallel-Horus is a unique attempt to provide a transparent programming solution for distributed sets of cluster computers. In other words, our research contrasts most significantly with that of others in that it deals with

- the Grid's programmability,
- transparent, yet coordinated, use of distributed resources, and
- providing efficient parallel and distributed performance at all levels of granularity.

To our knowledge, no other system combines all of these aims.

Evaluation

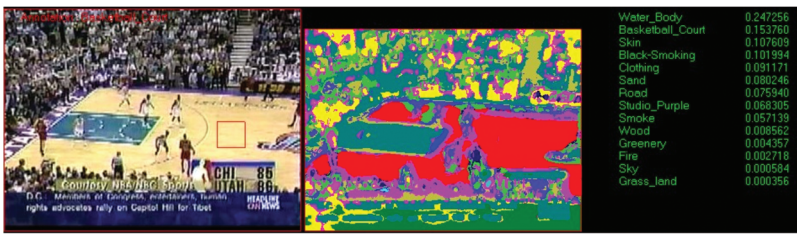
After setting up the Parallel-Horus system, we decided to assess its framework to see how effective it was at providing significant performance gains. To this end, we evaluated the wide-area execution of two realistic multimedia applications.

The first application is the software we used for our participation in Trecvid.[1] The second application is an object-recognition task performed by a Sony Aibo robot dog. In both applications, the processing on each cluster is fully data parallel. All wide-area data transfer (for now, although we're considering other possibilities for transferring data) is between the client and a single compute node (that is, the *root* of the multimedia service) at each cluster only.

We tested the applications on multiple cluster systems located at research institutes all over the world. We performed the bulk of these measurements using the Distributed ASCI Supercomputer 2 (DAS-2), a 200-node system located at five universities in The Netherlands. We performed all

Table 1. Overview of each cluster system that tested the efficiency and performance gains of the Parallel-Horus framework.

Institute	City	Country	Name	Computing Nodes	Interconnect	RAM (in Gbytes)
Vrije Universiteit	Amsterdam	Netherlands	DAS-2	72 × 2 1-GHz Pentium-III	Myrinet-2000	2
Leiden University	Leiden	Netherlands	DAS-2	32 × 2 1-GHz Pentium-III	Myrinet-2000	1
University of Amsterdam	Amsterdam	Netherlands	DAS-2	32 × 2 1-GHz Pentium-III	Myrinet-2000	1
Delft University of Technology	Delft	Netherlands	DAS-2	32 × 2 1-GHz Pentium-III	Myrinet-2000	1
University of Utrecht	Utrecht	Netherlands	DAS-2	32 × 2 1-GHz Pentium-III	Myrinet-2000	1
SARA	Amsterdam	Netherlands	Lisa	272 × 2 3.4-GHz Pentium-IV	800 Mbps InfiniBand	2
Salzburg University	Salzburg	Austria	Gaisberg	36 × 2 AMD Athlon MP2800+	6 × 6 Dolphin SGI torus	2
Conservatore Nazionale di Ricerche	Genoa	Italy	–	16 × 1 2.66-GHz Pentium-IV	Gigabit Ethernet	0.5
Cyfronet AGH	Krakow	Poland	Zeus	8 × 2 2.4-GHz Pentium-IV	Gigabit Ethernet	1
Monash University	Melbourne	Australia	Mahar	50 × 1 3.0-GHz Pentium-IV	Gigabit Ethernet	1



(a) **(b)**

Figure 2. (a) Example from the Trecvid data set and (b) its labeled segmentation.

additional measurements using four different cluster systems in Europe and one in Australia. Table 1 provides the specific characteristics of each cluster. More recently, we ran these applications at an even larger scale, concurrently using more than 20 cluster systems located on three different continents (see http://www.science.uva.nl/~fjseins/aibo.html).

Example 1: Trecvid content-based video retrieval

The Trecvid[1] conference series aims to encourage research in video retrieval by providing a large test collection, uniform scoring procedures, and a forum for organizations to compare their results.

The Trecvid high-level feature extraction task is as follows: Given the Trecvid video data set, a common shot boundary reference for this data set, and a list of feature definitions, participants must return for each feature a list of at most 2,000 shots from the data set, ranked according to the highest possibility of detecting the presence of that feature. The provided video data set consists of around 200 hours of news episodes—mostly from ABC and CNN—an example of which we show in Figure 2a. In addition, each competitor receives a set of feature definitions, which over the years has included Bill Clinton, beach, mountain, basket scored, explosion or fire, and people walking.

We based our approach to the high-level feature extraction problem on the Semantic Pathfinder, a novel method for generic semantic concept detection in multimodal video repositories.[1] The Semantic Pathfinder extracts semantic concepts from three consecutive analysis steps:

- content analysis,
- style analysis, and
- context analysis.

The content analysis step works on the video data itself, whereas the style and context analysis steps work on higher-level semantic representations. Here we focus on the video data analysis

```
inputVideo = openFile(videoFileName);
semanticConcepts = readFile(conceptsFileName);
svMachine = initSupportVectorMachine(semanticConcepts);
WHILE (NOT endOfVideo(inputVideo)) DO
       inputFrame = getNextFrame(inputVideo, skipFrames = 15);
       invFeatureVector = buildInvariantFeatureVector(inputFrame);
       labeledFrame = doSVMlabeling(invFeatureVector, svMachine);
       rankHistogram = getHistogram(labeledFrame);
       writeFile(rankHistogram);
OD
```

Figure 3. Pseudocode for the support vector machine-based visual analysis.

in the content analysis step, as this is by far the most time-consuming part of the system.

We analyzed the video data per frame (see Figure 3). For each 15th video frame, we extracted visual features using Gaussian color invariant measurements.[14] Then we decorrelated red-green-blue (RGB) color values via transformation to an opponent color system. Subsequently, we used Gaussian smoothing to suppress acquisition and compression noise.

By varying the size of the Gaussian filters, we obtained a color representation that was consistent with variations in target object size. The system then suppressed global and local intensity variations by normalizing each color value by its intensity, resulting in two chromaticity values per color pixel.

Furthermore, we obtained rotationally invariant features by taking Gaussian derivative filters and combining the responses into two chromatic gradient magnitude measures. These seven features, calculated over three scales, yielded a combined 21-dimensional feature vector per pixel. Figure 3 shows this sequence of operations, represented by a single call to `buildInvariantFeatureVector`.

The invariant feature vector serves as input for a multiclass support vector machine (SVM) that associates each pixel to one of the predefined regional visual concepts. In Figure 3 this is performed by `doSVMlabeling`. The SVM labeling results in a weak semantic segmentation of a video frame in terms of regional visual concepts. This result is written out to file in a condensed format (that is, a histogram) for subsequent processing.

Segmenting video frames into regional visual concepts is computationally intensive. This is especially true if you aim to analyze as many frames as possible. In our approach, visually analyzing a single video frame requires around 16 seconds on the fastest sequential machine at our disposal. Consequently, when processing two frames per second at a frame rate of 30, the required processing time for the entire Trecvid data set would be around 250 days.

Measurements. The sequential code for the Parallel-Horus service that implements this application immediately constitutes a program that executes efficiently on a cluster system. In fact, as we previously noted, the automatic data parallel execution is optimal with respect to the abstraction level of message-passing programs.[7]

The server application takes uncompressed video frames as input and produces uncompressed labeled frames as output (see Figure 2). Sending uncompressed video data between the server and client introduces a large amount of unnecessary communication overhead. For this first example application, we still chose to do so, as a relatively high two-way communication load gives us better insight in the system's characteristics. Moreover, if we can show our approach to be effective under these circumstances, we make a strong case for our system's potential.

Table 2a (next page) shows the measurements we obtained using each of the five DAS-2 clusters, one at a time. Our results show the time spent on processing a single frame, measured both on the server's side and with the client's application.

As you can see, the wide-area communication overhead is around 50–60 milliseconds (ms) at all times. Because we sent uncompressed frame data totaling more than 0.5 Mbytes per service call, this wide-area overhead is certainly acceptable. Also, the wide area overhead is marginal in comparison to the overall execution time per frame, irrespective of the actual number of nodes used. As a result, we conclude that our framework is highly effective when using the DAS-2 clusters.

Table 2b gives similar results for the other clusters. It shows that the wide-area overhead depends on the location of each cluster. The overhead is acceptable for the Stichting Academish Rekencentrum Amsterdam (SARA) cluster, as well

*Table 2. Client- and server-side parallel runtime results for the Trecvid application using input frames of size 352 × 240 (3-byte) pixels, given in seconds per video frame.**

Number of CPUs	Vrije Universiteit		Leiden University		University of Amsterdam		Delft University of Technology		University of Utrecht	
	Server	Client	Server	Client	Server	Client	Server	Client	Server	Client
1	32.11	32.16	34.43	34.50	33.17	33.22	31.67	31.72	33.95	34.01
2	13.37	13.43	15.14	15.20	14.10	14.15	13.25	13.31	13.25	13.31
4	7.32	7.38	7.62	7.68	7.59	7.55	7.54	7.59	7.41	7.47
8	3.68	3.74	3.51	3.57	3.53	3.58	4.02	4.08	3.59	3.65
16	2.45	2.50	2.50	2.56	2.58	2.64	2.53	2.59	2.52	2.59
24	1.99	2.05	2.00	2.07	2.08	2.13	2.08	2.14	1.97	2.04
32	1.60	1.66	—	—	—	—	—	—	—	—
40	1.32	1.38	—	—	—	—	—	—	—	—
48	1.17	1.23	—	—	—	—	—	—	—	—
56	1.08	1.14	—	—	—	—	—	—	—	—
64	0.97	1.03	—	—	—	—	—	—	—	—

(a)

Number of CPUs	SARA		Salzburg University		Conservatore Nazionale di Ricerche		Cyfronet AGH		Monash University	
	Server	Client	Server	Client	Server	Client	Server	Client	Server	Client
1	9.41	9.54	10.30	11.06	10.93	11.29	13.31	14.48	40.25	47.01
2	4.70	4.84	5.15	5.89	6.10	6.45	7.75	8.92	19.69	26.46
4	2.56	2.70	2.83	3.56	3.37	3.70	4.93	6.11	10.74	17.54
8	1.29	1.42	1.42	2.15	1.83	2.17	2.78	3.96	5.41	12.20
16	0.74	0.87	0.75	1.46	1.17	1.51	—	—	3.36	10.24
24	0.51	0.65	0.53	1.27	—	—	—	—	3.06	9.84
32	0.41	0.55	0.41	1.14	—	—	—	—	3.34	10.13
40	—	—	0.36	1.11	—	—	—	—	—	—
48	—	—	0.32	1.05	—	—	—	—	—	—
56	—	—	0.29	1.03	—	—	—	—	—	—
64	—	—	0.26	1.02	—	—	—	—	—	—

(b)

* Client located at the University of Amsterdam; resource broker located at Cyfronet AGH, Krakow, Poland.

as the clusters in Salzburg and Genoa—ranging from 130–760 ms.

However, the impact of wide-area communication is fairly large for the clusters in Krakow and Melbourne—even when you consider that we continuously sent uncompressed data at all times. Even though clusters with such large wide-area overhead might not be the most obvious choice for application in large-scale multimedia problems, we can still effectively use these clusters to lower the overall execution times of full applications.

Figure 4a shows the performance speed-up obtained when using the DAS-2 cluster at the Vrije Universiteit in Amsterdam. If you compare Figure 4a to Figure 4b, you can see the speed-up characteristics obtained when using four DAS-2 clusters at the same time.

We obtained the base of Figure 4b's graph by taking the average single-node execution time. The speed-up graph of Figure 4b (ranging over 96 CPUs) has an identical shape to the first part of Figure 4a (ranging over 24 CPUs). This indicates that there's no additional overhead at the client side when applying more than one multimedia server. In other words, the obtained speed-up with respect to the number of applied multimedia servers is fully linear.

We should note here that a similar speed-up comparison is difficult to make when using non-DAS-2 clusters, as each of these has different performance and speed-up characteristics. Therefore, we refrain from making such a comparison.

Example 2: Object recognition by a robot dog

Our second application demonstrates object recognition performed by a Sony Aibo robot dog

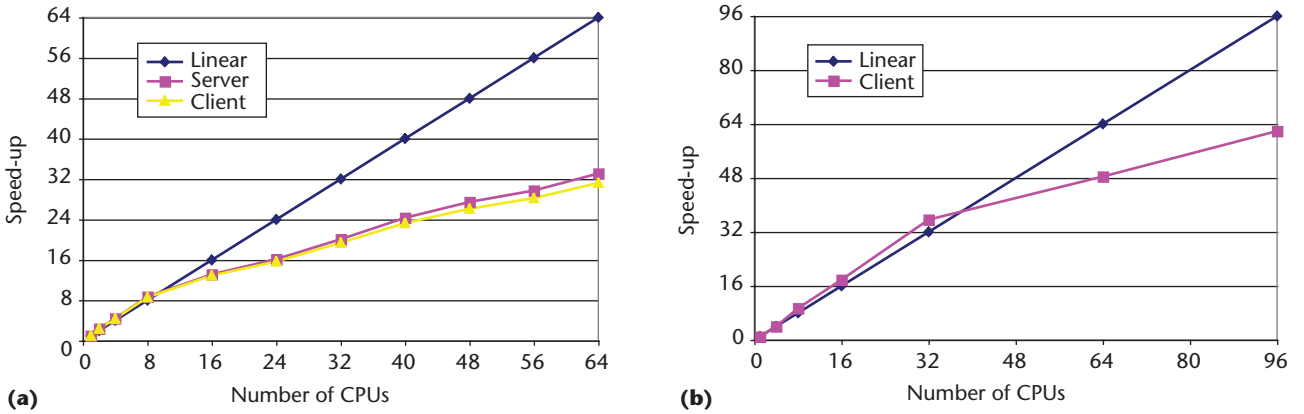


Figure 4. (a) Client- and server-side speed-up for the Vrije Universiteit results shown in Table 2. (b) Client-side speed-up when employing four DAS-2 clusters (Vrije Universiteit, Leiden, University of Amsterdam, and Delft) simultaneously.

(see Figure 5). Although the application is a toy application in itself, the applied techniques are identical to more relevant surveillance camera problems that we're working on as well.

In this application, the initial goal for this dog is to obtain local histograms of invariant features for each aspect of an object. Specifically, we obtained photometric invariance by considering two nonlinear transformations.[14]

The first color invariant, W, isolates intensity variation from chromatic variation—for example, edges because of shading, cast shadow, and albedo changes of the object surface. The second invariant feature, C, measures all chromatic variation in the image, disregarding intensity variation—for instance, all variation where the color of the pixels changes. These invariants measure point properties of the scene and are referred to as *point-based invariants*.

Point-based invariants can be unstable and noise sensitive.[15] Therefore, we constructed local histograms of responses for the color invariants. We obtained localization by estimating the histogram under a kernel. We used kernel-based descriptors because they're highly discriminative, and they've been successfully applied in tracking applications. Additionally, we used an opponent color space with additional photometric invariant transformations, such that color values are decorrelated.

From natural image statistics research, we know that we can model histograms of derivative filters using simple distributions.[16] In our case, we succeeded in modeling the local histogram of invariants of derivative filters using an integrated Weibull-type distribution. The resulting Weibull density parameters ß and γ indicated the (local) edge contrast and the (local) roughness or texture, respectively.[14]

We applied a simple algorithm for object recognition, based on the described invariant features. The Weibull parameters characterize an object at a fixed set of locations in the image. For each histogram, we estimated the Weibull parameters.

In the "learning" phase, we presented a set of 1,000 objects to the Aibo dog under a single visual setting. For each of these objects, it stored the learned set of Weibull parameters in a database. In the "recognition" phase, we showed the same objects again, under 50 different appearances, with varying lighting direction, lighting color, and viewing position.

In this manner, our dog learned each of the 1,000 objects from one example, while being capable of recognizing more than 300 of these under multiple natural imaging conditions. In a

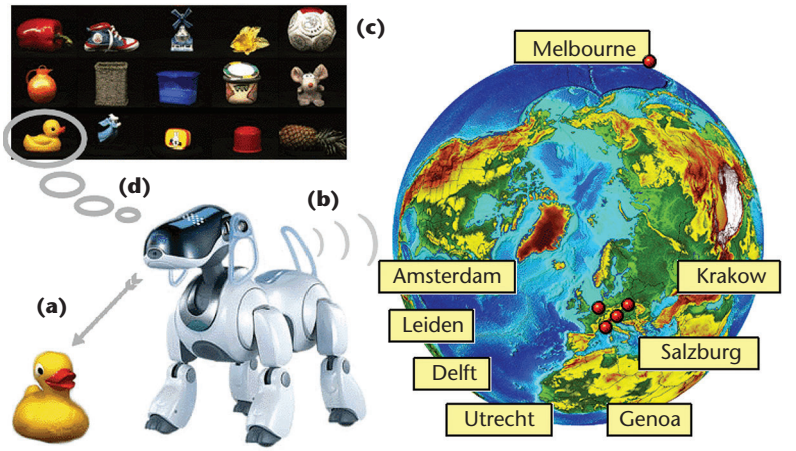


Figure 5. (a) An object is held in front of the dog's camera. (b) The system processes video frames on available clusters. (c) Based on the scene descriptions calculated from the video frames, the system searches a database of learned objects, and (d) upon recognition, the dog reacts accordingly. A video presentation (for which we received the "most visionary research" award at the Association for the Advancement of Artificial Intelligence 2007 conference) is available at http://www.science.uva.nl/~fjseins/aibo.html.

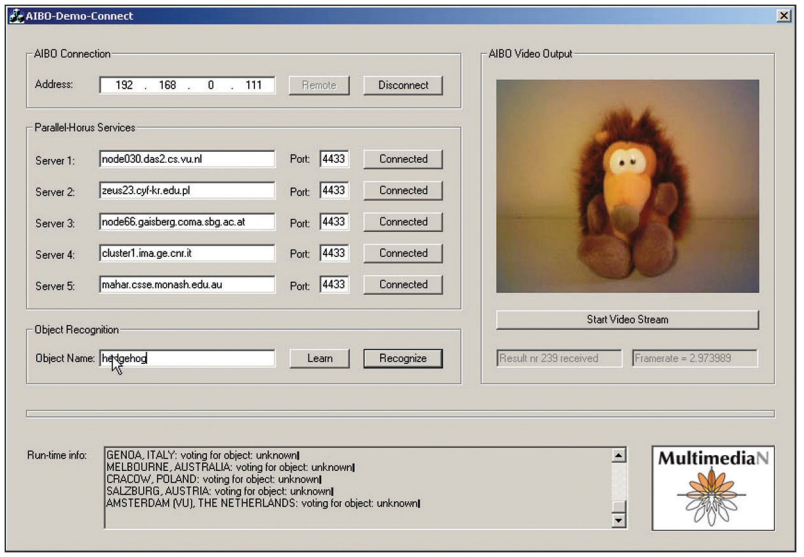


Figure 6. The Aibo object recognition client GUI, with five available Parallel-Horus services shown. The dog "learns" when you type an object name in the appropriate text box. Subsequently, recognition is immediate (by all participating services), as we show in the runtime info box. See also http://www.science.uva.nl/~fjseins/aibo.html.

controlled laboratory setting this approach proved to outperform related work significantly[17] (see also Figure 6).

Measurements. As Table 3 shows, on average the sequential algorithm runs at around 1 frame per 4 seconds. Because we average the results obtained from 8 consecutive frames, object recognition takes approximately 30 seconds, which is far from real time.

Table 3a shows data parallel execution results obtained for each of the five different DAS-2 clusters. As you can see, the wide-area communication overhead is around 10 ms at all times, which is clearly an acceptable overhead. In comparison to the previous example application, the wide-area communication is much lower, as we're now sending frame data in a JPEG compressed form—that is, around 10–20 Kbytes per server call.

Based on these results, we conclude again that our framework is highly effective when using DAS-2 clusters. The results in Table 3b also show much more acceptable wide-area overheads, even in the case of Krakow and Melbourne. Interestingly, however, the application isn't always very effective in obtaining high performance on all of the clusters. This is probably because of the nonoptimal configurations of the interconnection network, as well as high network traffic caused by other users.

If you look at Figure 7, you'll see that it's similar to Figure 4. Again we see an identical shape in Figure 7b, when compared to the graph of Figure 7a (ranging over the first 24 CPUs). As a result, additional DAS-2 clusters deliver linear speed-ups at the client application side. Finally, a test where we applied all of the evaluation clusters resulted in a maximum rate of more than 10 frames per second, which clearly indicates that real-time multimedia Grid computing is within reach.

Conclusions and future work

Parallel-Horus allows researchers in multimedia content analysis to implement high-performance applications as sequential C++ programs, using a carefully designed set of building block operations. In this article, we described results obtained by matching the Parallel-Horus programming model with an execution model based on wide-area multimedia services. The resulting framework, which integrates the efficient task-parallel invocation of multimedia services with the automatic data-parallel execution of these services, requires no parallelization or distribution effort from its users.

Additionally, our example applications have shown that with our wide-area extensions to Parallel-Horus, we can greatly improve speed-up characteristics obtained on single-cluster systems. The distributed set up of our framework adds extra network overhead, but this tends to be marginal in comparison to the total execution times.

Considering these features, we feel that we've succeeded in building a framework that integrates a user-transparent programming model with an efficient and easy-to-use Grid execution model. We should note, however, that the process of convincing the user to apply Parallel-Horus isn't only driven by reason or quality. Users are generally reluctant to change to something new. The best we can do is to listen carefully to the demands of users and to adhere to these demands as well as possible.

With this in mind, we feel that Parallel-Horus is quite capable of having a stimulating effect on the study of the many computationally demanding problems in multimedia content analysis. The presented example applications are merely two of these.

We would also like to note that the work we describe is part of a larger endeavor to bring the benefits of high-performance distributed computing to the multimedia community. While we showed that the deployment of Parallel-Horus on wide-area systems is straightforward, many fundamental research problems must be solved before the system can make effective use of available Grid resources.

One of our goals to this end is to improve the performance of wide-area Parallel-Horus programs further by making them variability-toler-

*Table 3. Client- and server-side parallel runtime results (in seconds) for the Aibo application using input frames of size 412 × 318 (3-byte) pixels.**

Number of CPUs	Vrije Universiteit		Leiden University		University of Amsterdam		Delft University of Technology		University of Utrecht	
	Server	Client	Server	Client	Server	Client	Server	Client	Server	Client
1	7.62	7.63	7.98	7.99	7.75	7.76	7.80	7.81	7.92	7.93
2	4.11	4.12	4.25	4.28	4.29	4.30	4.15	4.17	4.33	4.34
4	2.29	2.31	2.31	2.33	2.28	2.29	2.27	2.28	2.38	2.39
8	1.29	1.30	1.28	1.29	1.30	1.32	1.30	1.31	1.31	1.32
16	0.73	0.74	0.73	0.75	0.73	0.74	0.74	0.75	0.73	0.74
24	0.69	0.70	0.69	0.70	0.69	0.70	0.70	0.71	0.69	0.70
32	0.53	0.54	—	—	—	—	—	—	—	—
40	0.60	0.61	—	—	—	—	—	—	—	—
48	0.57	0.58	—	—	—	—	—	—	—	—
56	0.55	0.56	—	—	—	—	—	—	—	—
64	0.47	0.48	—	—	—	—	—	—	—	—

(a)

Number of CPUs	SARA		Salzburg University		Conservatore Nazionale di Ricerche		Cyfronet AGH		Monash University	
	Server	Client	Server	Client	Server	Client	Server	Client	Server	Client
1	1.71	1.73	2.45	2.52	2.36	2.41	4.37	4.53	9.25	9.91
2	1.01	1.03	1.29	1.36	1.31	1.36	2.75	2.91	5.37	6.02
4	0.62	0.64	0.71	0.78	0.91	0.96	2.13	2.29	3.85	4.50
8	0.46	0.48	0.41	0.47	0.76	0.81	2.11	2.27	3.36	4.02
16	0.41	0.43	0.29	0.36	0.73	0.78	—	—	3.31	3.97
24	0.40	0.42	0.31	0.37	—	—	—	—	4.03	4.70
32	0.37	0.39	0.24	0.31	—	—	—	—	6.18	6.84
40	—	—	0.32	0.39	—	—	—	—	—	—
48	—	—	0.32	0.38	—	—	—	—	—	—
56	—	—	0.32	0.38	—	—	—	—	—	—
64	—	—	0.27	0.33	—	—	—	—	—	—

(b)

* Client located at the University of Amsterdam; resource broker located at Cyfronet AGH, Krakow, Poland.

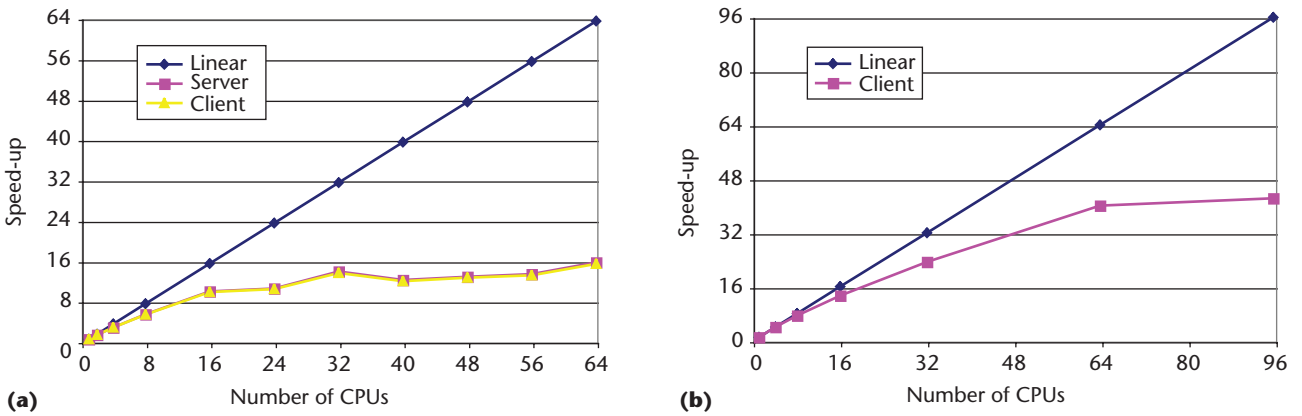


Figure 7. (a) Client- and server-side speed-up for the Vrije Universiteit results shown in Table 3. (b) Client-side speed-up when employing the clusters at Vrije Universiteit, Leiden, University of Amsterdam, and Delft simultaneously.

One of our goals is to improve the performance of wide-area Parallel-Horus programs further by making them variability tolerant.

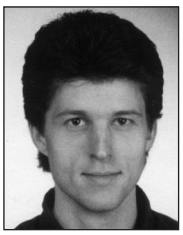
ant through controlled adaptive resource use. This raises the need for stochastic runtime performance control methodologies that react to the dynamic circumstances in large Grid systems.

Another goal is to develop an efficient, fully sequential, programming model for describing client programs, based on the definition of algorithmic patterns of distributed execution. In general, each such pattern defines which set of computations should be performed on which input data, the dependencies that exist among the computations and the data, and whether results must be returned to the client or elsewhere.

Our recent research has uncovered distribution patterns in several multimedia applications, primarily with respect to the ordering of results data (including globally ordered, ordered under time constraints, and unordered—with and without allowing a partial loss of results). We hope to explore these patterns further in the near future. **MM**

References

1. C.G.M. Snoek et al., "The Semantic Pathfinder: Using an Authoring Metaphor for Generic Multimedia Indexing," *IEEE Trans. Pattern Analysis and Machine Inteligence.*, vol. 28, no. 10, 2006, pp. 1678-1689.
2. A. Plaza et al., "Commodity Cluster-Based Parallel Processing of Hyperspectral Imagery," *J. Parallel Distributed Computing*, vol. 66, no. 3, 2006, pp. 345-358.
3. F.J. Seinstra et al., "User Transparency: A Fully Sequential Programming Model for Efficient Data Parallel Image Processing," *Concurrency and Computation: Practice and Experience*, vol. 16, no. 6, 2004, pp. 611-644.
4. F.J. Seinstra et al., "A Software Architecture for User Transparent Parallel Image Processing," *Parallel Computing*, vol. 28, no. 12, 2002, pp. 1685-1708.
5. F.J. Seinstra et al., "P-3PC: A Point-to-Point Communication Model for Automatic and Optimal Decomposition of Regular Domain Problems," *IEEE Trans. Parallel and Distributed Systems*, vol. 13, no. 7, 2002, pp. 758-768.
6. F.J. Seinstra et al., "Finite State Machine-Based Optimization of Data Parallel Regular Domain Problems Applied in Low-Level Image Processing," *IEEE Trans. Parallel and Distributed Systems*, vol. 15, no. 10, 2004, pp. 865-877.
7. G.X. Ritter et al., *Handbook of Computer Vision Algorithms in Image Algebra*, CRC Press, 1996.
8. J. Serot et al., "Skeletons for Parallel Image Processing: An Overview of the Skipper Project," *Parallel Computing*, vol. 28, nos. 7-8, 2002, pp. 967-993.
9. C. Nicolescu et al., "A Data and Task Parallel Image Processing Environment," *Parallel Computing*, vol. 28, nos. 7-8, 2002, pp. 945-965.
10. G. Alonso, *Web Services—Concepts, Architectures and Applications*, Springer-Verlag, 2004.
11. G. Allen et al., "The Grid Application Toolkit: Towards Generic and Easy Application Programming Interfaces for the Grid," *Proc. IEEE*, vol. 93, no. 3, 2005, pp. 534-550.
12. S. Basu et al., "MmGrid: Distributed Resource Management Infrastructure for Multimedia Applications," *Proc. 17th Int'l Parallel and Distributed Processing Symp.*, IEEE CS Press, 2003, CD-ROM.
13. A. Zaia et al., "A Scalable Grid-Based Multimedia Server," *Proc. 13th Int'l Workshops Enabling Technologies: Infrastructure for Collaborative Enterprises*, IEEE CS Press, 2004, pp. 337-342.
14. J.M. Geusebroek et al., "A Six-Stimulus Theory for Stochastic Texture," *Int'l J. Computer Vision*, vol. 62, nos. 1-2, 2005, pp. 7-16.
15. J. van de Weijer et al., "Color Edge and Corner Detection by Photometric Quasi-invariants," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 27, no. 4, 2005, pp. 625-630.
16. A. Srivastava et al., "On Advances in Statistical Modeling of Natural Images," *J. Mathematical Imaging and Vision*, vol. 18, no. 1, 2003, pp. 17-33.
17. J.M. Geusebroek, "Compact Object Descriptors from Local Colour Invariant Histograms," *Proc. British Machine Vision Conf.*, British Machine Vision Assoc., vol. 3, 2006, pp. 1029-1038.



Frank J. Seinstra is a senior researcher in the Intelligent Systems Lab at the University of Amsterdam and in the Department of Computer Science at the Vrije Universiteit. His research focuses on high-performance multimedia

computing, and he's currently leading a multidisciplinary project on adaptive resource utilization in multimedia Grid computing. Seinstra received his PhD in computer science from the University of Amsterdam.

Jan-Mark Geusebroek is an assistant professor in the Intelligent Systems Lab at the University of Amsterdam. His research interest is in cognitive vision, especially front-end color and texture vision and mechanisms of focal attention. Geusebroek received his PhD in computer science from the University of Amsterdam and was awarded a prestigious young talent grant from the Netherlands Organization for Scientific Research.

Dennis Koelma is a senior scientific systems designer in the Intelligent Systems Lab at the University of Amsterdam. Currently, he works on Impala/Horus, which is a software architecture for multimedia content access and analysis. His research interests include multimedia computing, (parallel) software architectures, graphical user interfaces, and visual information systems. Koelma received his PhD in computer science from the University of Amsterdam.

Cees G.M. Snoek is a senior researcher in the Intelligent Systems Lab at the University of Amsterdam. He's also the lead designer of the MediaMill video search engine, which was awarded the best technical demonstration at ACM Multimedia 2005. His research focuses on techniques for concept-based video retrieval. Snoek received his PhD in computer science from the University of Amsterdam.

Marcel Worring is an associate professor in the Intelligent Systems Lab at the University of Amsterdam. His research interests are in multimedia search and systems, and he leads several multidisciplinary projects covering knowledge engineering, pattern recognition, image and video analysis, and information space reduction, conducted in close cooperation with industry. Worring received his PhD in computer science from the University of Amsterdam.

Arnold W.M. Smeulders is a professor in multimedia information analysis in the Intelligent Systems Lab at the University of Amsterdam. He's also the scientific director of The Netherlands' national MultimediaN public–private partnership. He's a member of the steering board of the European Technology Platform Networked and Electronic Media (NEM) and the IEEE International Conference on Multimedia and Expo (ICME). He's also a fellow of the International Association for Pattern Recognition (IAPR) and acts as the associate editor of the *International Journal of Computer Vision* and *IEEE Transactions on Multimedia*.

Readers may contact Frank J. Seinstra at fjseins@cs.vu.nl or fjseins@science.uva.nl.

For further information on this or any other computing topic, please visit our Digital Library at http://www.computer.org/csdl.