Multimedia Content Analysis and Indexing: Evaluation of a Distributed and Scalable Architecture

JM Van Thong, Scott Blackwell, Chris Weikart, Hasnain A. Mandviwala[1]
Cambridge Research Laboratory
HP Laboratories Cambridge
HPL-2003-182
August 27th , 2003*

multimedia information systems, content-based searching, media indexing, media processing, distributed computing, grid computing, web services

Multimedia search engines facilitate the retrieval of documents from large media content archives now available via intranets and the Internet. Over the past several years, many research projects have focused on algorithms for analyzing and indexing media content efficiently. However, special system architectures are required to process large amounts of content from real-time feeds or existing archives. Possible solutions include dedicated distributed architectures for analyzing content rapidly and for making it searchable. The system architecture we propose implements such an approach: a highly distributed and reconfigurable batch media content analyzer that can process media streams and static media repositories. Our distributed media analysis application handles media acquisition, content processing, and document indexing. This collection of modules is orchestrated by a task flow management component, exploiting data and pipeline parallelism in the application. A scheduler manages load balancing and prioritizes the different tasks. Workers implement application-specific modules that can be deployed on an arbitrary number of nodes running different operating systems. Each application module is exposed as a web service, implemented with industry-standard interoperable middleware components such as Microsoft ASP.NET and Sun J2EE. Our system architecture is the next generation system for the multimedia indexing application demonstrated by www.speechbot.com. It can process large volumes of audio recordings with minimal support and maintenance, while running on low-cost commodity hardware. The system has been evaluated on a server farm running concurrent content analysis processes.

* Internal Accession Date Only Approved for External Publication
[1] Georgia Institute of Technology, College of Computing
To be published in and presented at ITCom 2003, 7-11 September 2003, Orlando, Florida

1 Introduction

Increases in data storage and network bandwidth capacity have made massive amounts of audio, video, and images available on intranets and the Internet. However, digital media documents, though rich in content, generally lack structured and descriptive metadata that would allow indexing, random access, and cross-linking. Consequently, content processing algorithms, such as segmentation, summarization, speech recognition, etc. are clearly needed for indexing and searching of digital multimedia. Moreover, many system level challenges remain to be solved to enable efficient indexing and searching of large multimedia document collections[17].

First, multimedia indexing systems need to be designed to process vast amounts of content (e.g. millions of hours of audio/video) and to retrieve relevant information interactively with rapid response time. Thousands of hours of media are now available online, but a limited number of text-based search engines exist to retrieve these documents. Our content-based media search engine, SpeechBot [1], indexes over 17,000 hours of content, a mere fraction of what is archived on the Internet. SingingFish [4] uses attached production metadata to index over 9 million multimedia URL's. Radio and TV stations that possess large archives, in some cases over a million hours, in analog or digital form, face a similar content indexing and retrieval problem.

Second, media indexing systems need to be extensible, allowing additional content processing or search algorithms to either improve existing functionality or add new features. Also, it is likely that several different content analyzers and search methods will have to be combined to achieve better results. Integrating human generated metadata can improve performance even further.

Finally, media search systems need to have good precision and accuracy. Although digitizing and storing large quantities of media files are challenging problems in themselves, browsing and searching these files accurately is an even more complex problem, studied by many research groups in the recent past [10, 5, 22, 20, 7]. Given the level of noise and the lack of structure in media documents, content analysis and indexing represent a challenging signal processing, machine learning and information retrieval problem [11].

In the following sections, we first describe the system challenges of large multimedia content analysis systems. Then we propose a system design that addresses the problems of system scalability and extensibility for multimedia content analysis and indexing. We conclude with a preliminary performance evaluation, and future work.

2 Multimedia searching and indexing systems

2.1 Searching low-level perceptual metadata

Since media documents cannot be searched in their native form, they must be processed to extract either perceptual or semantic information which will allow indexing and searching. Visual and acoustic data are low level representations and do not enable high level semantic queries, such as finding individuals in a picture or words spoken in an audio recording. Image perceptual representations may entail Color, Shape, and Texture (CST) vectors. Likewise, acoustic representations may include pitch, rhythm or phoneme transcription. These metadata are usually

represented as feature vectors, like CST vectors or a lattice of acoustic units (e.g. a phoneme lattice). Such metadata often require minimal computational resources to be generated. Hence, simple streaming pipeline architectures can be used, allowing the production of feature streams while the media is recorded and digitized.

Low-level features can be used to search content by a similarity match. For example, a feature vector is computed from a query image and compared to feature vectors for each image in the repository; images with the highest similarity scores are returned [21]. Similarly, from an audio recording, a lattice of phonemes can be computed with a speech recognition system. A spoken query or a text-based query term may then be converted into a sequence of phonemes (i.e. a phonemic pronunciation from a dictionary) that can be scanned through the phoneme lattice in order to return the most likely acoustic matches [8].

These approaches can be fairly effective for querying multimedia, but have several significant drawbacks. First, they perform queries at the perceptual level, e.g. finding images that look alike or occurrences of spoken words that sound like a given word. Although simple to use, queries are limited to perceptual clues and users still must choose the right query example to get relevant results. Moreover, these algorithms often generate a number of false positives that increase with the size of the searched database. Finally, they do not scale well since the entire repository must be traversed for every query. However, significant research effort has been expended in the design of similarity matching systems that can scale up on demand. In this approach, the most significant challenge is scanning the whole repository with acceptable performance (retrieval accuracy and response time).

2.2 Indexing high-level semantic metadata

In contrast, semantic queries can be performed when higher level metadata representations have been computed. Semantic metadata representations may include, for instance, the word or topic transcriptions of spoken audio content, or the description of the subject in a picture. Such metadata can then be indexed in its native form (usually text), allowing queries for exact matches in near constant time. However, this approach is not without its disadvantages.

First, the semantic analysis of the media content utilizes statistical models which need to be trained beforehand on a predefined set of objects, e.g. a word dictionary for speech recognition. The analysis is then performed with this fixed set of models, limiting the output to the trained objects only. Second, metadata must be computed up front, and often a sequence of properly orchestrated complex algorithms must be executed before the information is available for searching.

In summary, there is a tension between two possible approaches for media content processing and indexing. One scheme involves processing the data into low-level features at a low computational cost, and then searching the metadata at query time. Another, our approach, comprises computing higher level semantic representations at a higher initial cost, but enhances query performance by indexing metadata. Both approaches require building distributed systems with significant computing power and intelligent middleware. Ultimately, systems combining low level metadata search and semantic indexing will likely achieve the best results.

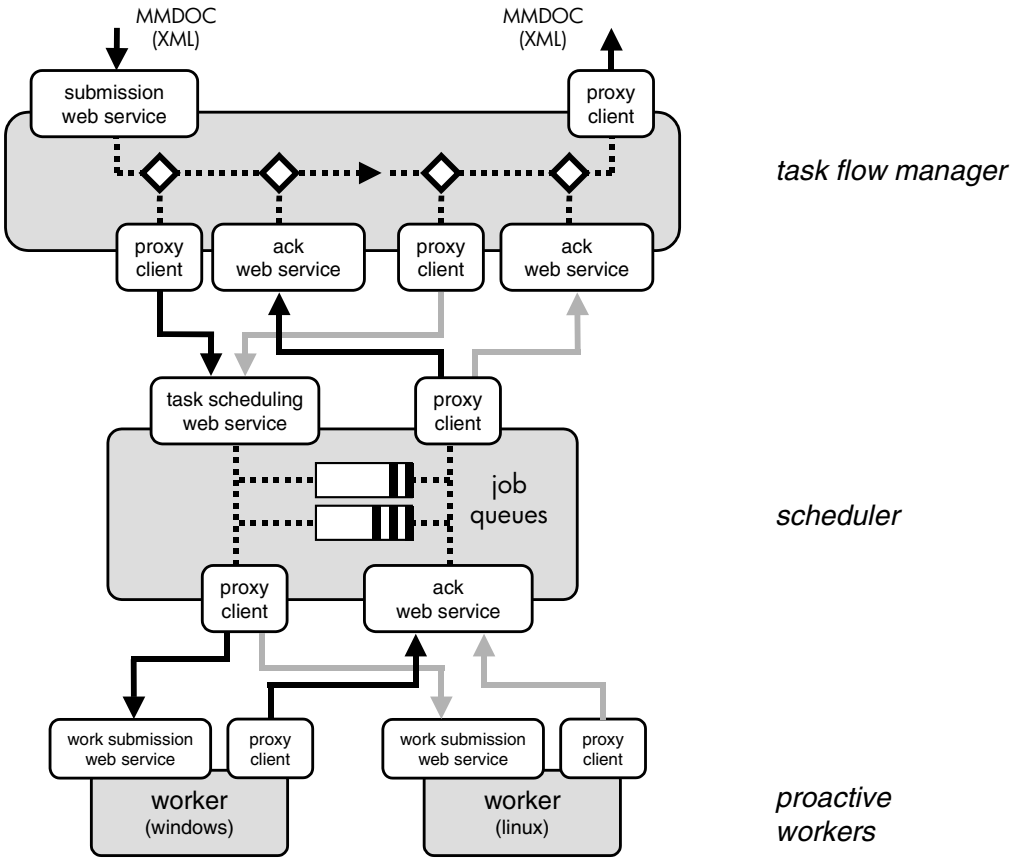

Figure 1: System overview.

3 System Architecture

3.1 Design Requirements

The sheer abundance of multimedia content and the expensive computations needed to analyze each document require a system architecture that is highly scalable. Demands for scalability are ever-increasing and every design has its theoretical limit; however, there is a tradeoff between complexity, efficiency and scalability in any given design that achieves the right balance for a given problem. A good system architecture should not have crippling limitations that restrict a *reasonable* expansion of the system.

For the problem of providing scalable resources, two approaches have traditionally been used. Using tightly coupled clusters such as *Beowulf* [19] is one solution, yielding a high performance support system with very low overhead. However, these systems have a disadvantage in that they are dependent on a homogeneous underlying hardware and software platform. The second approach is to use a loosely distributed architecture. Here, the system design permits the use of multiple hardware and software platforms, but carries the disadvantages of costly messaging

overhead, bookkeeping, failure recovery, etc. Each architecture has its usefulness in a given problem domain. We have adopted the distributed architecture, allowing for scattered resources on multiple platforms that can be added and removed quickly. This design scales well with some overhead.

Another core challenge in a distributed system solution is workflow management. Parallel applications, such as the one currently being addressed, require a sequence of tasks to be executed in the proper order[16]. Such workflows may be seen as directed acyclic graphs (DAGs) having pipelined components within each graph that link the output of one component to the input of the next, as in a producer-consumer model. Moreover, as content analysis algorithms are constantly refined, they drive the need for a more flexible workflow management sub-system so that changes in workflow may be easily assimilated.

Workflow designs may not be *static* as described above, but instead *dynamic* requiring a different flow graph for every given input document. Our requirements, on the other hand, are met by a semi-static workflow with a manager that allows for compile-time modifications and not necessarily dynamic run-time changes.

With the need for workflow management and the obvious requirement of distributed computational nodes, workflow management clearly may be dissociated from task scheduling. A scheduler helps prioritize tasks and maximize available resource usage, required for an efficient distributed, parallel application. Task prioritization in particular is a novel feature which may be exploited when, for example, breaking news coverage must be analyzed before the system can finish its existing batch of work. Scheduling can also help manage heterogeneous resources and capabilities by becoming aware of them (such a specific face detector service) and matching task requirements with resource availability.

Heterogeneity is a reality of multimedia analysis applications as many software components developed on various architectures must be plugged together. This requires standards that promote interoperability between computational components. The *web services* model is an emerging solution for such problems, and our architecture is built using web services to implement an asynchronous messaging layer.

A distributed architecture creates the need for reliability in certain critical components so that failures are tolerated. Mechanisms such as checkpoint-restart are useful, and allow for recovery to a consistent state in case of fatal crashes of several components.

3.2 Related Work

A recent trend has seen grid-based systems emerging [12], implementing systems for different application domains such as medical imaging [14], computer vision, simulation, data visualization, graphical interactive sessions[15], etc. Grid systems allow data- and CPU-intensive processes to run on clusters of heterogeneous computers, potentially located at different sites. Many features of our infrastructure mirror those found in grid systems, but differ in capability and simplicity.

Other systems built for streaming multimedia content analysis, such as *Stampede* [18], allow for interactive and real-time application pipelines by dropping data when necessary. However, real-time processing is not a requirement of our system, given requirements addressed by batch-

processing applications. Bounded latency similar to real-time processing is a desirable feature for content analysis but may only be achieved by providing sufficient resources for processing the input data stream – dropping data is not acceptable for a media indexing application.

Work has also been done in workflow representations, such as derived or manually-built grammar representations [23], defining dependencies between analytic components. Other frameworks represent dependencies as an object-oriented hierarchy, linking data objects and algorithms in a semantic graph [13]. Our approach is different as it uses a high-level *graphical* description of the process components. The input/output data to/from media content analyzers are encapsulated using XML documents. This is similar to an earlier version of our system [9].

3.3 System components

Our system architecture is divided into three components with each component assigned a unique role in performing media content analysis. These components are the (1) *workflow manager*, (2) the *proactive workers*, and (3) the *scheduler*. This architecture is presented in greater detail in our companion paper [16].

The *workflow manager* controls the order of media processing. The workflow manager collects feature metadata by enriching a metadata XML document format called *MMDOC* [16]. As features of the multimedia document are computed, they are appended to this document, producing a richly structured high-level feature set that is indexed for access via user queries. The workflow manager is able to handle hundreds of simultaneous workflows from a set of pre-defined workflow graphs. It performs checkpointing and can restart at the last consistent state.

A unique workflow graph wf_i, where i is the MMDOC workflow graph, is formally described by a set of analytical components $AC_i = \{ac_{i,j}\}$, where j is the analytic component ID-string. Each $ac_{i,j}$ is described by PR_r, where PR_r is the set of physical resources required to execute that analytic component. The knowledge of required physical resources is critical since it is needed to optimize scheduling. Resource requirements can be estimated by simulation [6], learned from previous executions, or deduced from heuristics. However, for our initial implementation we simply assign values based on empirical observation.

The next component in the architecture is the remote *proactive worker*. Workers provide computational capability for multimedia content analysis. These workers are aware of their available resources, including the set of analytic components $AC_i = \{ac_{i,j}\}$ they can execute. Therefore, workers can advertise a tuple rc_k, defined as the resource capability of the worker $k = (PR_a$, the set of available physical resources; and $AC_i = \{ac_{i,j}\}$, the set of available analytical components $)$ to the scheduler when requesting work instances $w_{i,j}$. This design allows workers to be heterogeneous as long as they describe their capabilities appropriately. The workers are *proactive* in seeking work, eliminating the burden of a join/leave subsystem to track them. Each worker simply submits an rc_k to get a work instance $w_{i,j}$.

The *scheduler* manages the distribution of tasks to worker nodes and therefore optimizes the execution of tasks by minimizing idle time. The scheduler is aware of PR_r, the physical resources required by an analytic component $ac_{i,j}$. The scheduler has priority-ready queues that are named on the $(PR_r, ac_{i,j})$ tuple, which enable efficient task scheduling when proactive workers submit their published $rc_k = (PR, AC)$ requests. All tasks submitted to workers are transferred to

another queue, where they are kept until results are received by the scheduler. To handle recovery from failed workers, each enqueued operation is accompanied with a fixed expiration time. When a task expires, it is reinserted in the appropriate queue for rescheduling. This design frees the scheduler from tracking tasks submitted to unreliable worker nodes.

3.4 Implementation Details

We have implemented an asynchronous message-passing protocol between the components described in the previous section. In this protocol, an acknowledgement is a separate message, instead of a synchronous response to an initial request (figure 1). This approach avoids blocking connections for relatively long-lived tasks, and is appropriate for loosely-coupled, possibly wide-area distributed systems. Messages are transmitted RPC-style via web services - *Simple Object Access Protocol* (SOAP) over HTTP. It is often argued that XML-based wire formats are inefficient and overkill for message passing in a heterogeneous environment. But for the problem at hand, where content processing tasks are computationally much more expensive than XML messaging overhead, the advantages of XML make it an acceptable choice.
To support the architectural design, all tasks must carry some meta-information. This consists of a workflow instance identifier, a task priority, the physical resource requirements PR_r of the task, and an asynchronous acknowledgement location. These are all encapsulated within a SOAP document header as shown in figure 2.

```
<soap-env:Header>
   <types:SBHeader>
      <workflowID xsi:type="xsd:string">
         {BE524120-51D7-4F29-A448-94B1C7BF055F}
      </workFlowID>
      <resReq xsi:type="xsd:string">cpu=4, network=4</resReq>
      <taskPriority xsi:type="xsd:int">0</taskPriority>
      <ackURL xsi:type="xsd:string">
         http://<hostname>:<port>/speechbot-scheduler/SchedulerServlet
      </ackURL>
   </types:SBHeader>
</soap-env:Header>
```

Figure 2: A SOAP header included with every SOAP message exchanged in the system.

3.5 Platform Specification

The described system was implemented using Java and .NET C# development tools on Linux and Microsoft Windows operating systems. Java web services were implemented using Java 2 Enterprise Edition (J2EE) and the Java Web Services Development Pack (JWSDP) API collections. Java web services were built on Linux, as .NET is unavailable for Linux and other Unix variants. .NET, instead of Java, was used on Windows to demonstrate platform independence and interoperability within the architecture.

The workflow manager was implemented using the commercial package *Microsoft BizTalk Server 2002* [2] on Windows 2000 Server. BizTalk incorporates the graphical tool *Visio* [3] to ease the design of workflow graphs. Support for reliable, checkpointed workflows and document logging were additional desirable features of BizTalk. Scalability in BizTalk is achieved by *dehydrating* long running workflows to avoid blocking system resources. Dehydration is the process by which a workflow instance awaiting a message is written to storage. Once the expected message arrives, the dehydrated workflow is *hydrated* to continue processing. The workflow manager communicates with external resources (Java and .NET web services) via proxies implemented using the .NET Framework and registered as COM+ components. Asynchronous results are returned to the workflow manager using .NET web services which place acknowledgments in message queues.
The Scheduler was designed using the Java APIs while workers were implemented using .NET and Java. Next, we describe the performance of our system implementation.

4 Performance Evaluation

The system described above is implemented by a functionally complete prototype. While not yet suitable for production deployment, we are able to use this implementation to process multiple, concurrent jobs to completion.
The prototype implementation is currently configured to produce text transcriptions of audio from Real Media streams. It orchestrates the services of four proactive workers to accomplish this goal.

GetReal downstreams and decodes Real Media URL to raw audio (WAV) and metadata,

Mfcc produces preliminary feature vectors (Melcep Frequency Cepstral Coefficients, or MFCCs) for speech recognition from audio,

Transcribe produces transcriptions from overlapping audio chunks and MFCCs,

Merge merges the results of multiple Transcribers into a single transcription.

The two most time-consuming workers in the system are GetReal and Transcribe. Because GetReal is aimed at external streaming archives, accessible via proprietary streaming protocols, it cannot process streams faster than the end-user playback rate dictated by the built-in Real Player ^{tm} components. Therefore, GetReal is network and playback-rate-bound. On our current hardware platform, we run out of CPU at 30 GetReal processes, but can scale out with multiple boxes. We then saturate our currently allotted 4 Mb/s to the Internet at 88 GetReal processes. This provides more than enough scalability for our purposes. But as long as we process at playback rates, we cannot do better than 1x real-time. Alternatively, an FTP worker can download an audio file made available on an FTP server, in a mere fraction of the file's duration.
Transcribe is based on a CRL custom version of the Sphinx 3 large vocabulary speech recognizer from CMU (Carnegie Mellon University), and processes audio between 5x and 10x real-time.

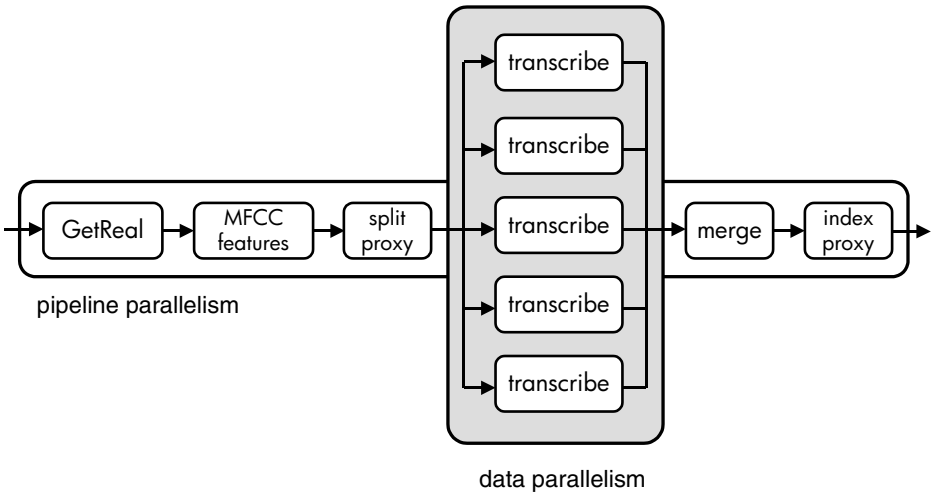

Figure 3: Application pipeline.

This would represent a low throughput performance, were it not for our support of data parallelism. The input audio is split into overlapping chunks, and fed to multiple Transcribe workers which process the chunks in parallel (figure 3). The results are then merged into the final transcript. This approach can scale to a throughput equal to the input rate of media documents at the cost of some latency, given enough Transcribe workers are provided.

Distributed event logging was implemented that enabled us to capture the arrival, scheduling, submission, and result acknowledgement of each work component. Even though all hosts were NTP-synchronized, there was considerable clock-jitter between hosts. We circumvent the problem by comparing readings taken on the same host during log analysis.

In order to evaluate the system, we deployed up to five workers with Transcribe, Mfcc and Merge services along with two workers providing the GetReal service. The singleton scheduler and BizTalk workflow manager were deployed on dedicated machines. Our hardware resources include dual Intel P4 Xeon 2.4GHz machines with 1GB of RAM running Mandrake Linux 9.1, along with several Intel PIII SMP machines running Microsoft Windows 2000 and XP.

Work Component	**Computing time avg (min)**	**STD**	**Scheduling overhead avg (min)**	**STD**	**Total latency avg (min)**	**STD**	**Message overhead avg (min)**
GetReal[1]	N/A	N/A	00:03.94	00:02.04	N/A	N/A	N/A
Mfcc	02:23.11	00:56.75	00:01.78	00:00.60	02:25.03	00:57.28	00:00.14
Transcribe	12:19.65	02:32.89	00:02.26	00:01.16	12:22.05	02:32.14	00:01.14
Merge	00:00.60	00:00.24	00:01.15	00:00.90	00:02.24	00:00.74	00:00.50

Table 1: A breakdown of the computation time and scheduling overhead for each worker observed in the system.

In table 1 we show the performance of the target application. The scheduling overhead on average is only 2.29 seconds per task request, which is largely due to the 5 second polling interval preset on all proactive workers. Each Transcribe task injected into the workflow consists of 5 audio segments of 35 seconds each. Adjacent segments overlap by 5 seconds; thus each Transcribe job represents an audio document of 155 seconds (2.5 minutes) in length.
Messaging overhead is negligible compared to the computation times of each task, except for the Merge task where the message sizes are considerably larger. However, for this application domain, the use of XML to leverage interoperability between various platforms is justified. We have not tested the scalability limit of the system because of time constraints.

5 Conclusions and future work

Our system implements a scalable distributed architecture for multimedia content processing. The different functions of the system (workflow management, scheduling, and analytic tasks) are implemented as distinct and disjoint components. Each component communicates with the other using standard protocols, thus achieving interoperability. The system scales easily by adding new worker resources, distributed across the network.
In our design we exploit data parallelism and pipeline parallelism to distribute work load. Media streams fit this model particularly well, as they can be split along time into independent units of work. Moreover, the system is designed such that every component can be replicated as needed, including the workflow manager and the scheduler. This is possible because each component is independent of each others.
However, several problems remain to be addressed. Performance can be optimized further by improving data locality between the remotely distributed components. We also do not address security issues, such as the problem of rogue workers stealing tasks away. Fortunately, we can take advantage of emerging security standards for web services.
Future work will focus on alternate implementations of the workflow management component. Although we can interactively design new workflows using Visio's graphical interface, the result is a static XML document that cannot be modified at run time. Dynamic workflows are desirable when the topology of the workflow depends on the media content. Moreover, we would like to allow users to programmatically create new workflows. We also plan to deploy the system on a larger scale, allowing us to dramatically increase the size of our public index [1].

6 Acknowledgments

We would like to thank Leonidas Kontothanassis and Matt Gaubatz for their comments while reviewing this document. Our fruitful discussions with Leonidas helped us compare our work with contemporary research on distributed systems. Core research on speech recognition, semantic extraction, and information retrieval has been carried out tirelessly and with great talent by Pedro Moreno, Beth Logan, and Dave Goddeau. We would like to acknowledge the

[1]The GetReal module is bounded by the play-back rate and its performance is controlled by factors outside the consideration of this system

early contributions of Digital Equipment Corporation Cambridge Research Lab (now Hewlett-Packard) researchers, including Tom Levergood, Dave Wecker, T. V. Raman, Brian Eberman, and Michael Swain who envisioned media-content-based indexing systems when the Web was still in its infancy. And last, the presented system architecture is the latest version of a system initially designed and developed by our talented colleagues from (gone but not forgotten) Software Engineering Australia, including Blair Fidler, Katrina Maffey, Matthew Moores, Ian Walters and Liam Davies.

References

[1] Speechbot web site, December 1999. http://www.speechbot.com.

[2] Biztalk server, 2002. http://www.microsoft.com/biztalk/.

[3] Visio, 2002. http://www.microsoft.com/office/visio/.

[4] Singingfish web site, July 2003. http://www.singingfish.com.

[5] D. Abberley, G. Cook, S. Renals, and T. Robinson. Retrieval of broadcast news documents with the THISL system. In *Proceedings of the 8th Text Retrieval Conference (TREC-8)*, 1999.

[6] J. Cao, S. A. Jarvis, S. Saini, and G. R. Nudd. Gridflow: Workflow management for grid computing. In *Third IEEE/ACM International Symposium on Cluster Computing and the Grid.*, May 2003.

[7] S-F Chang, Q. Huang, T. Huang, A. Puri, and B. Shahraray. Multimedia search and retrieval. In *Multimedia Systems, Standards, and Networks*, 2000.

[8] M. Clements, P. S. Cardillo, and M. S. Miller. Phonetic searching vs. LVCSR: How to find what you really want in audio archives. In *20th Annual AVIOS Conference*, 2001.

[9] A. P. de Vries, B. Eberman, and D. E. Kovalcin. The design and implementation of an infrastructure for multimedia digital libraries. In *Proceedings of the International Database Engineering and Applications Symposium*, July 1998.

[10] N. Dimitrova, H-J. Zhang, B. Shahraray, I. Sezan, T. Huang, and A. Zakhor. Applications of video-content analysis and retrieval. *IEEE Multimedia Journal*, 9(3), 2002.

[11] C. Djeraba. Content-based multimedia indexing and retrieval. *IEEE Multimedia*, 9(2), April 2002.

[12] I. Foster, C. Kesselman, J.M. Nick, and S. Tuecke. The physiology of the grid: An open grid services architecture for distributed systems integration. In *http://www.globus.org/ogsa*, 2002.

[13] J. Griffioen, R. Mehrotra, R. Yavatkar, and R. Adams. Moods: A multimedia information modeling system. In *Workshop on Database Systems, ACM Multimedia*, 1994.

[14] S. Hastings, T. Kurc, S. Langella, U. Catalyurek, T. Pan, and J. Saltz. Image processing for the grid: A toolkit for building grid-enabled image processing applications. In *CCGRID2003, Third IEEE/ACM International Symposium on Cluster Computing and the Grid*, May 2003.

[15] R. Kumar, V. Talwar, and S. Basu. A resource management framework for interactive grids. In *First International Workshop on Middleware for Grid Computing, 2003*, 2003.

[16] H. Mandviwala, S. Blackwell, and JM Van Thong. A scalable multimedia content analysis and indexing system architecture. In *CBMI 2003, Third International Workshop on Content-Based Multimedia Indexing*, September 2003.

[17] M. T. Maybury. *Intelligent Multimedia Information Retrieval*. AAAI Press, and The MIT Press, 1997.

[18] U. Ramachandran, R. S. Nikhil, N. Harel, J. M. Rehg, and K. Knobe. Space-time memory: A parallel programming abstraction for interactive multimedia applications. In *10th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, May 1999.

[19] T. L. Sterling. *Beowulf Cluster Computing with Windows*. MIT Press, 2001.

[20] JM. Van Thong, D. Goddeau, A. Litvinova, B. Logan, P. Moreno, and M. Swain. Speechbot: a speech recognition based audio indexing system for the web. In *Proc. RIAO*, 2000.

[21] N. Vasconcelos and A. Lippman. A probabilistic architecture for content-based image retrieval. In *IEEE Conference Computer Vision and Pattern Recognition 2000*, 2000.

[22] H. D. Wactlar, A. G. Hauptmann, and M. J. Witbrock. Informedia: News-on-demand experiments in speech recognition. In *DARPA Speech Recognition Workshop*, 1996.

[23] M. Windhouwer, A. Schmidt, and M. Kersten. Acoi: A system for indexing multimedia objects. In *International Workshop on Information Integration and Web-based Applications and Services*, Indonesia, November 1999.