Design, Implementation, and Analysis of a Multimedia Indexing and Delivery Server

Leonidas Kontothanassis, Chris Joerg,
Michael J. Swain, Brian Eberman,
Robert A. Iannucci

Cambridge Research Laboratory

Cambridge Research Laboratory
Technical Report Series

CRL 99/5
August 1999

COMPAQ

Cambridge Research Laboratory

The Cambridge Research Laboratory was founded in 1987 to advance the state of the art in both core computing and human-computer interaction, and to use the knowledge so gained to support the Company's corporate objectives. We believe this is best accomplished through interconnected pursuits in technology creation, advanced systems engineering, and business development. We are actively investigating scalable computing; mobile computing; vision-based human and scene sensing; speech interaction; computer-animated synthetic persona; intelligent information appliances; and the capture, coding, storage, indexing, retrieval, decoding, and rendering of multimedia data. We recognize and embrace a technology creation model which is characterized by three major phases:

Freedom: The life blood of the Laboratory comes from the observations and imaginations of our research staff. It is here that challenging research problems are uncovered (through discussions with customers, through interactions with others in the Corporation, through other professional interactions, through reading, and the like) or that new ideas are born. For any such problem or idea, this phase culminates in the nucleation of a project team around a well articulated central research question and the outlining of a research plan.

Focus: Once a team is formed, we aggressively pursue the creation of new technology based on the plan. This may involve direct collaboration with other technical professionals inside and outside the Corporation. This phase culminates in the demonstrable creation of new technology which may take any of a number of forms - a journal article, a technical talk, a working prototype, a patent application, or some combination of these. The research team is typically augmented with other resident professionals—engineering and business development—who work as integral members of the core team to prepare preliminary plans for how best to leverage this new knowledge, either through internal transfer of technology or through other means.

Follow-through: We actively pursue taking the best technologies to the marketplace. For those opportunities which are not immediately transferred internally and where the team has identified a significant opportunity, the business development and engineering staff will lead early-stage commercial development, often in conjunction with members of the research staff. While the value to the Corporation of taking these new ideas to the market is clear, it also has a significant positive impact on our future research work by providing the means to understand intimately the problems and opportunities in the market and to more fully exercise our ideas and concepts in real-world settings.

Throughout this process, communicating our understanding is a critical part of what we do, and participating in the larger technical community—through the publication of refereed journal articles and the presentation of our ideas at conferences–is essential. Our technical report series supports and facilitates broad and early dissemination of our work. We welcome your feedback on its effectiveness.

Robert A. Iannucci, Ph.D.
Director

Design, Implementation, and Analysis of a Multimedia Indexing and Delivery Server

Leonidas Kontothanassis, Chris Joerg,
Michael J. Swain, Brian Eberman,
Robert A. Iannucci

August 1999

Abstract

While text indexing on the Web and corporate intranets has received considerable attention, indexing of multimedia documents in the same environments is still in its nascent stages. We have developed a system that allows us to index and serve multimedia documents over the Internet. The system has been in continuous operation since March 1998 within the company and has been tested extensively under heavy load conditions. In this paper we describe the design and implementation of the system and we present a user-model study indicating the usage patterns that we encountered during the system deployment. The main part of the paper presents the results of a detailed study that sheds light into the performance characteristics of the system components, identifies bottlenecks, and provides information to systems designers interested in delivering systems with good performance for this important class of applications.

We find that the limiting factor for the three major subcomponents that comprise the indexing service is the memory system performance, while the component responsible for delivering the video/audio bits to the users is only limited by the bandwidth of our Internet connection. Surprisingly, I/O is not a bottleneck on our system despite the large amount of disk space consumed by the multimedia data.

CRL Technical reports are available on the CRL's web page at
http://www.crl.research.digital.com.

Compaq Computer Corporation
Cambridge Research Laboratory
One Kendall Square, Building 700
Cambridge, Massachusetts 02139 USA

1 Introduction

Digital multimedia content has increased dramatically over the last few years with the advent of cheap storage and networking technologies. Inexpensive storage makes it possible to store and back-up multimedia documents in digital formats with costs comparable to that of analog formats; and inexpensive networks allow the transmission of multimedia documents to clients in corporate intranets and through the Web.

While it is possible that a user may want to view or work with an entire document, it is more likely that only a snippet out of a long document is needed. The capability to index multimedia and service only the pieces of a multimedia document that a user needs, can significantly enhance the usability of a multimedia content server.

We have built a system called CRL Media Search [5] (termed Media Search from now on) that indexes both video and audio documents and allows for delivery of only the relevant segments in response to a user query. Our work is based on earlier design concepts by De Vries *et al* [14] but we have significantly modified our design goals to address the World Wide Web and the extremely large amounts of content and large numbers of users associated with it. The service we have implemented bears significant resemblance to the service provided by search engines on the World Wide Web. Like search engines we allow the user to perform a text search against a database of multimedia documents and receive a response containing a list of relevant documents.

However, there are two important components that differentiate our work with that performed by search engines:

- Unlike traditional search engines, Media Search returns multiple hits within a document. The main reason for returning multiple internal hits is that unlike text documents, multimedia documents do not provide the user with an easy way to search inside a document. Therefore we provide the service of identifying and summarizing all the relevant locations within a document.
- In addition to providing indexing services, Media Search also delivers the relevant content to the user. Content delivery is largely orthogonal to indexing but we believe it is likely, especially in the case of intranets, that the two services will be combined.

We completed an initial implementation of the system in 1998 and have been running it nearly continuously over Compaq's corporate intranet, with modifications since then. We have added a significant amount of content, including broadcast material from the Web, technical talks, and captured video content. The extended use of the system has allowed us to derive a usage model and to implement artificial clients that mimic the behavior of our human clients. In this way we were able to study the performance of the system under controlled load conditions.

In our study we have tried to understand what parts of the hardware used to run our system are most responsible for any performance bottlenecks. Our system is composed of a relational database (Oracle), an external text index based on the $\mathrm{AltaVista}^{\mathrm{TM}}$ index, and a collection of Perl scripts that manage the presentation of the results to the user. Even though previous performance studies of these applications are already available in the literature [3, 10], our workload differs significantly from the workloads

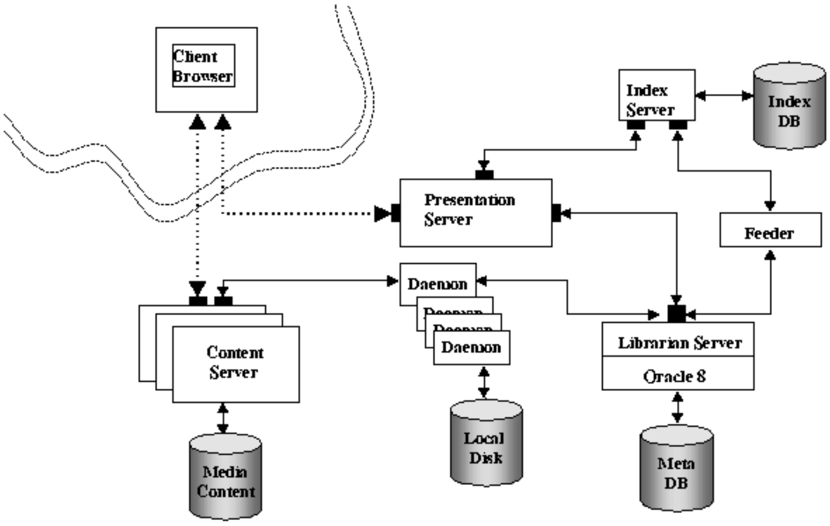

Figure 1: CRL Media Search Architecture

used in existing studies. It differs from transaction processing in that there are no updates to the data, and from decision support in that we present a very large number of small simple queries to the system rather than a small number of large complex ones.

The remainder of this paper is organized as follows. Section 2 describes the implementation of the system and the functionality it present to the user. We present our experimental environment and describe the usage model we have derived based on the internal release on section 3. Our experimental results and the performance characterization of the system are presented in section 4. In section 5 we present related work. Finally, we summarize and present directions for future work in section 6.

2 System Description

In this section we present a detailed description of the CRL MediaSearch system.

The system (Figure 1) is broken into three major sections: 1) one or more Content Servers, 2) a collection of autonomous periodic processing engines or daemons, and 3) the indexing and query response services. This paper focuses on the performance study of the indexing and query response services but we discuss the remaining two pieces for the sake of completeness.

The general flow of a user query through the system is as follows: Initially the query is presented to the Presentation subsystem which contacts the index server for all locations of all documents that match the query. Based on the index server's answer, the presentation will then ask for metadata information (i.e. title, author, etc.) for each returned document from the meta database. In addition to the general metadata, context information for each location will also be retrieved from the metadatabase. Such information includes the text contained in the portion of the document that matches the query, an image that is representative of that portion of the multimedia document, and a link to the document itself so that the user can start playing it from that point onward. The flow of a query is presented in more detail at the end of this section where we also discuss the different types of queries supported by the system.

The functionality of the system component is as follows. The content server stores and serves up the actual content and provides a uniform interface to all forms of the media. In addition, it handles storage and access control. We use it to store MPEG files, RealVideo files, and our own internal low bit rate video format. The Content Server provides a simple URL based protocol for accessing these different forms of the same content.

The collection of daemons performs a multitude of services. They convert media from one format to another (*e.g.* there exists a daemon that converts mpeg representations to RealVideo), and they provide automated annotations to the multimedia document. Such automatic annotations include but are not limited to the segmentation of the document into interesting and logically cohesive subcomponents, and the time-alignment of a textual transcript to the video. These annotations are in fact the searchable component of our media. When an annotation matches a user query we can retrieve the time field of the annotation and present to the user the part of the multimedia file that corresponds to this time. Execution of the daemons is coordinated by a workflow system implemented on top of a relational database. The workflow system requires daemons to specify the types of inputs that they can handle and the types of outputs they produce. Based on this information the workflow system guarantees that daemons working on a particular media object will execute in the correct order by providing them with work only when all the required inputs are available.

By far the most demanding component of our system is the indexing and query response service and that component is the one to which we have devoted most of our effort. It can be broken down into three major sub-components: 1) a librarian server, which is built on top of a standard relational database and maintains metadata information for the documents, 2) an index server, which is a text indexing system based on a modified version of the AltaVista engine, and 3) a presentation server, which communicates with the other subsystems to construct an HTML response to a user query.

The main purpose of the librarian, excluding the workflow management mentioned above, is to store meta information and annotations about multimedia documents. Such meta information includes the title, author, and source of a particular media document. Annotations form the searchable part of our system and can have multiple forms. Currently there exist two common forms of annotations: a) segmentation information about a multimedia document and b) alignment information about the text that is spoken within a multimedia document.

The segmentation annotations consist of a start and end time for each segment, the text that is present in that segment, and a relevant picture—often referred to as a keyframe—associated with the segment. Document segments (also known as frames) are the parts of documents that the system returns in response to user queries. This is based on the assumptions that users are interested only in a portion of the document rather than the whole thing. Our system makes it possible to navigate through neighboring segments to accommodate users who may wish to view a larger portion of the document.

The alignment annotations consist of the start and end time of each word in the transcript of a document, the word itself, and the segment to which the word belongs[1]. A confidence estimate on the correctness of the annotation is also included. It is possible for the time intervals of annotations to overlap either because the annotations are of different types (*e.g.* segmentation and alignment annotations), or because multiple versions of a single annotation, possibly with different confidence estimates, have been inserted into the database.

The second major sub-component of the indexing and query response service is the indexing server. The AltaVista index provides an extremely efficient mechanism for searching text and serves as an external index to our meta database. The indexer accepts a query containing a set of words connected by a boolean expression and returns the document ID and an offset within the document that satisfies the query. We had to modify the index to return all the offsets within a document rather than just one. This a departure from the standard AltaVista model but is necessary due to the nature of the documents. The standard AltaVista engine simply returns the ID of the document that matches a query and leaves the task of locating the position that matches the query to the user. This is possible since it only handles text files that can be further searched using any editing tool. Returning an entire multimedia document that matches a query provides significantly reduced functionality as there are no tools for searching within a multimedia document. We have therefore extended our system to provide users with all relevant locations within a document.

The librarian and external index provide overlapping functionalities and one could envision a system design that included only one of the two components. For example, metadata information can be stored in the AltaVista index as well as the database while indexing of text can be accomplished inside of the relational database as well as in the external index we have used. We have decided to use both components since they each have decided advantages in how well they can handle certain tasks. The librarian provides a very easy interface to modify and update metadata and to add non-text based annotations. Furthermore it makes it easy to extract context information for document locations by indexing on the location ID. On the other hand, the AltaVista index is optimized for text indexing and is much more efficient in searching large amounts of text than a text index inside the relational database. In addition to the fast search capabilities, the external index allows for more complex searches (*e.g.* phrases, words near one another, etc.), and provides a simple, concise query language that is easier to use than SQL.

The last major sub-component of the indexing and query response service is the presentation server. This component is an HTTP server containing a collection of CGI

[1]The segment value for an alignment annotation can be computed, based on the start and end time information of the annotation. We store it explicitly only for efficiency reasons.

Figure 2: A sample response page by the Media Search System to an *all-documents* request.

scripts, written in Perl5, which get invoked to allow the user to navigate the information space we have indexed. In order to avoid the high cost associated with Perl startup, we have used Open Market's FastCGI^{TM} product which allows for a CGI script to become a server process. Requests to a script are then relayed to the corresponding process through a TCP socket. The scripts communicate with the various services offered by the librarian, index, and content servers to allow the user to perform searches, see responses, and view parts of the multimedia stream.

There are four main scripts in the presentation component: *all-documents query*, *single-document query*, *browse*, and *view*. Users begin by entering one or more words that they want to search for. This invokes the *all-documents query* script which returns a response page containing a collection of documents (typically the first 10) that match the query. In order to create this response page the *all-documents query* script first queries the index server with the user's words and gets back a list of documents and locations within documents that match the query. Using this information it can then query the librarian about the objects and the first location within each object in order to get back the relevant metadata (title, author, etc.) and context (surrounding text and image) information. It can then format the results and present them to the user. For each document we return the number of locations within the document that match the query, and we return context information about the first of these locations. We display

Figure 3: A sample response page by the Media Search System to a *browse* request.

the text that was spoken during the frame associated with the first location and, in the case of video, a representative picture from that portion of the video. Links to the *single-document query*, *browse* and *view* scripts are also embedded in the response for each hit.

Clicking on the *single-document query* link associated with a document runs the *single-document query* script which returns a page displaying the matches within that document. Again, for each match, we display the text and image associated with that match. The *browse* script allows the user to retrieve more context for a particular match. *Browse* displays the text and images associated with the frames that surround a frame selected by the user. Finally the *view* script instantiates the multimedia player and starts playing the multimedia document at the location that matches the user's query. It also provides some control over the player and in some cases choice over the format of multimedia that the system will deliver should more than one format exist. Examples of response pages to an *all-documents* and *browse* queries are shown in figures 2 and 3.

3 Methodology

For a system like CRL MediaSearch performance measurements only make sense in relation to their usage patterns. For example, the demands on our system are different depending on whether a user executes a general query, executes a more specific query within a document, or views a multimedia document. We released the system internally within the company and collected logs over a period of one week on how people used our system. Our system at the time contained approximately 200 hours of audio and video from a number of different sources including internal company videos, NPR shows, the state of the union address, and others.

After several tens of thousands of requests we were able to derive usage patterns and then create artificial clients that mimicked the behavior of the real users. This allowed us to examine the performance behavior of the system in a controlled environment under arbitrary amounts of load.

What we have found is that users follow a very standard usage pattern for our system:

- First the user performs a query specifying one or more words to search for. By far the most common case is single word queries but two or three word queries are also found. Then the the user spends an average of 25.1 seconds perusing the results.

- With probability 90% the user selects the *single-document query* page for a particular document and spends and average of 24.8 seconds perusing the search results within a document.

- Independent of the *single-document query* decision the user will select a *browse* page with probability 84% and will spend an average of 24.1 seconds on these results.

- Finally, independent of the earlier decisions the user will select a *view* page with probability 70% and will spend an average of 25.4 seconds playing the audio or video file.

We have created script clients that mimic the behavior described above. Running these script clients on workstations and PCs allows us to induce load on our servers and measure the performance and limitations of our system. One pass through the above 4 steps is called a *user session*. In our experiments each client script repeatedly performs user sessions as defined by the usage pattern shown above.

Our experimental setting consists of the server machines, which are two Digital AlphaServers 4100 each with four processors and 2Gbytes of local memory, and a large number of client machines. The server machines have Alpha 21164 processors with 8Kbyte Instruction and 8Kbyte Data on-chip first-level cache, a 96Kbyte 3-way set associative on-chip second-level cache, and a 4Mbyte board-level cache. We have divided the components of Media Search on the two servers by placing the presentation server on one of the machines and the librarian, indexing server, and content server on the other. The machine with the presentation server had 400MHz processors, while the machine with the librarian, indexer, and content server had 466MHz processors. Both machines have a system bus capable of delivering 1.2 Gbytes/sec.

The clients run on a collection of Unix and NT workstations. All machines are connected by a 100Mbit/sec FDDI network. We have focused our study on the CPU and memory system performance of the application. We also provide an estimate on the bandwidth requirements for serving query responses and content to a large number of clients.

While our study is by necessity tied to the particular hardware configuration we have used, we believe that our server configuration is representative of the state of the art and our results are applicable on a variety of other architectures with similar memory system characteristics. It is also possible that different types of content would result in different usage patterns. However we should be able to extrapolate the performance of a system under any usage model given our current results.

4 Performance Characterization

As we saw in section 2, CRL Media Search is a collection of components that cooperate to provide answers to users. In this section we analyze the performance of Media Search by providing details on the performance of each individual component. We start by looking at the contribution of each component on the total CPU budget of the system and we proceed to explain where the CPU cycles are spent within each component. In addition to CPU requirements we also look at the network bandwidth requirements of the system.

We applied simulated loads to our system in order to determine how many simultaneous users it could support while maintaining a reasonable response time. The Media Search system was able to serve 1250 concurrent users doing queries based on the usage model of section 3 with an average query response time of 0.44 seconds. Further increases in user load resulted in rapid degradation of response time and indicate that our AlphaServers were reaching saturation. To provide a more concrete number of the system capabilities the page view rate was 75 pages per second or 2.7 million pages viewed per working day[2].

While Media Search supports both query and multimedia delivery functionality, we have focused this performance study on the query side alone. We did study multimedia delivery as a standalone application and we discovered that its CPU and memory requirements are smaller than those of the query subsystem by an order of magnitude. For example, servicing queries for 1250 concurrent users keeps almost all of the eight processors on our two AlphaServers busy. These users will require just over 300 concurrent video feeds. Our tests have shown that we can service 500 concurrent $\mathrm{RealVideo}^{TM}$ feeds while keeping a single CPU only half busy[3] and therefore content delivery is only a small fraction of the CPU requirements for a system like Media Search.

Figure 4 shows the percentage of CPU time spent in each one of the system components in an experiment with 1250 clients each executing 14 users sessions. As can

[2]A working day on the World Wide Web is considered to last approximately 10 hours

[3]This was the largest test that our server license allowed us to run.

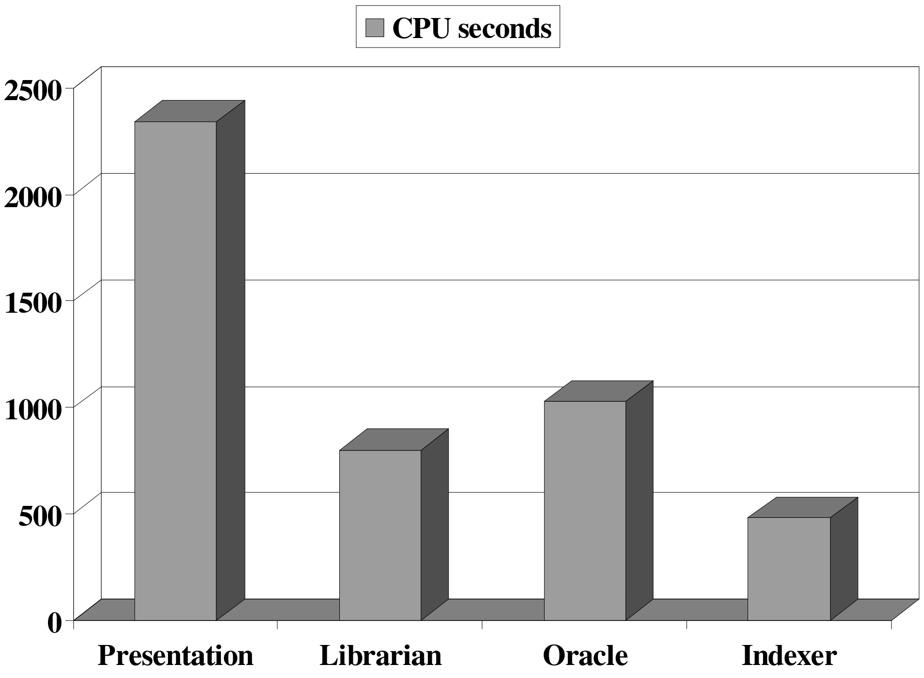

Figure 4: CPU time consumed by different system components.

be seen, the presentation layer is the largest consumer of CPU cycles contributing a little over 50% of the total CPU budget for all of Media Search. The second largest consumer of cycles is the librarian with the Oracle database and the libriarian front end combining for 40% of the CPU cycle while the indexer uses the remaining 10% of the total CPU budget.

We also profiled the execution of each of the components using Compaq's *Continuous Profiling Infrastructure* (CPI) tool [2] in order to determine how each of the components was spending its time. We will discuss those findings in the remainder of this section.

4.1 The presentation subsystem

The presentation layer is by far the largest consumer of CPU cycles. It is responsible for querying the other servers and formatting the results. Since it is the layer that needs to change most frequently in our system in response to user feedback, it has been written in an interpreted language (Perl). This allows for easy modification and customization but results in significantly higher overhead. As we mentioned in section 2 we have already avoided the most significant cost of using CGI scripts and Perl by moving to FastCGI. By far the highest cost of a CGI script is the startup of the Perl interpreter.

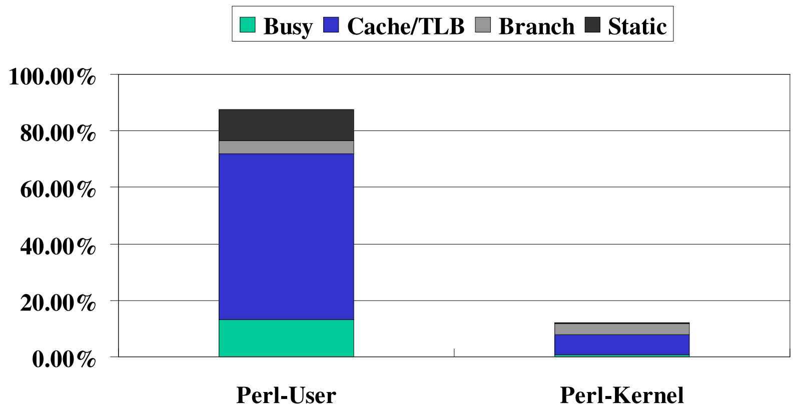

Figure 5: Breakdown of cycles in Perl.

FastCGI allows us to start a number of Perl processes upfront and converts calls to CGI programs that would normally start a new process into messages to one of the already existing processes.

However using an interpreted language still results in substantial overhead. Figure 5 shows how the Perl cycles are distributed between user and kernel mode. Furthermore both the kernel and user bars are divided, from top to bottom, into: a) cycles spent waiting for some internal processor resource (*e.g.* waiting for a value produced by a previous instruction or waiting for an execution unit to become available) (*Static*), b) cycles spent recovering from mispredicted branches (*Branch*), c) cycles spent waiting for memory (*Cache/TLB*), and d) useful or busy cycles (*Busy*). As can be seen from the figure very little time is spent in the kernel. Most of the kernel time is related to lock acquisitions, memory copying, and thread scheduling.

The user-level portion of the application spends most of its time waiting for the memory subsystem. We believe that our heavy usage of associative arrays is partially responsible for this behavior. Associative arrays are hash-table data structures that present to Perl coders the illusion of arrays where indices can be arbitrary strings. Our Perl scripts need to allocate and de-allocate a large number of associative arrays to hold the responses provided by the indexer and the librarian. Previous work [12] has shown that associative arrays are the most inefficient type of data structure in Perl and benefit

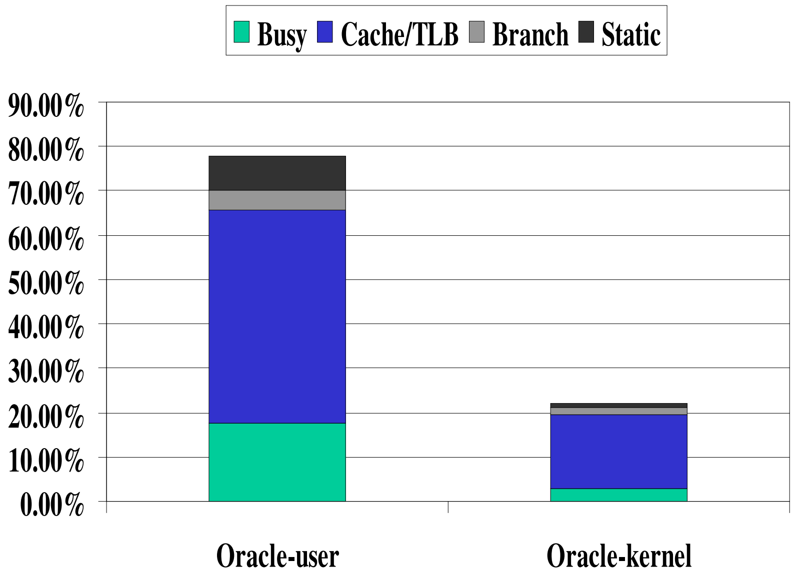

Figure 6: Breakdown of cycles in Oracle.

very little from the existence of the Perl precompilation phase.

While it can be argued that a real production system would not have used Perl as the scripting language for its presentation layer we still believe the Perl results are interesting. First they are independent of the behavior exhibited by the remaining components since the presentation layer was running on a separate machine, and second they showcase the performance costs associated with the flexibility that is often desired by experimental systems like ours that need to change frequently.

4.2 The librarian subsystem

The second major consumer of resources in our system is the librarian which consists of a front-end interface application and the Oracle back-end. The librarian is a critical part of the system because it is the hardest one to scale. The remaining components can be scaled simply by replication and distribution of queries to one of the replicas. The librarian can not be easily replicated due to consistency issues that arise when adding content to the system. For example, adding a document to multiple librarian instances would require that the addition succeed or fail atomically across all of the replicas. This problem does not arise with any of the other subsystems. The presentation layer does not maintain any state and thus failure of a presentation server would only reduce

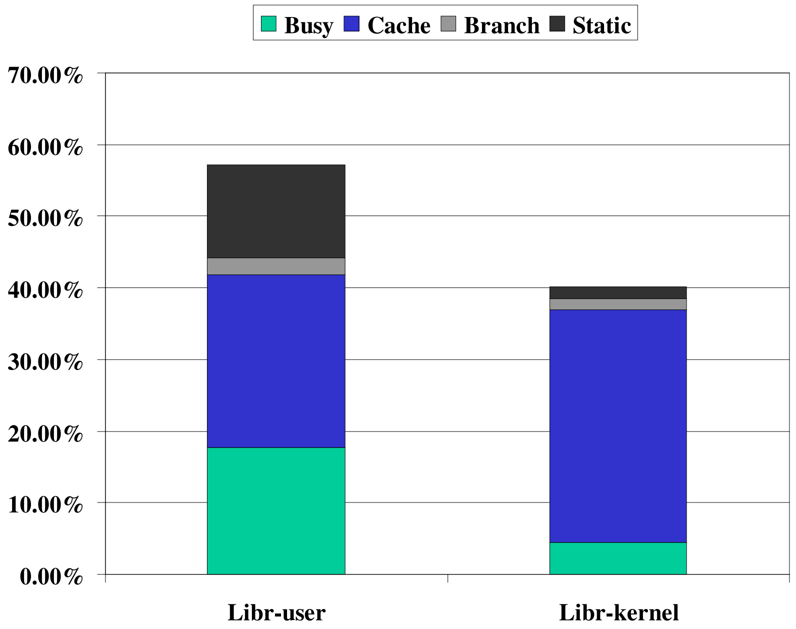

Figure 7: Breakdown of cycles in librarian front-end.

performance. The index server does maintain state but it builds this state based on the contents of the librarian. Should an index server fail it can be taken offline, rebuilt and then brought back online. Content integrity is never compromised because of an index failure.

We have used CPI to analyze the performance of the librarian. As can be seen in figure 6 the Oracle back-end spends 22% of the time in kernel-level code. Most of the kernel time is spent acquiring and releasing locks, copying data, and scheduling the Oracle processes on available processors. The kernel component of Oracle is quite inefficient as only 15% of the kernel cycles are actually spent executing useful instructions. The user-level component of the Oracle back-end exhibits somewhat better behavior with 23% of its cycles spent executing useful instructions. Still waiting for the memory system accounts for 61% of the user-level cycles, while a non negligible component (10%) is spent waiting for internal processor resources.

The librarian front-end provides connectivity to Oracle and packages Oracle responses into a format understood by the presentation layer. Our performance analysis tools indicate that it spends almost half of its time in kernel mode. Careful examination of the profile data reveals that almost all of the kernel time is due to scheduling events, lock acquisition and releases, and network accesses. This profile is consistent with our expectations from the application. The librarian front-end uses a large number of longlived threads and distributes queries between them in a round-robin fashion. Thread scheduling is handled by the kernel, with each thread receiving a request and eventually sending an answer back to the presentation layer using TCP/IP sockets. The kernel component of the application is very inefficient with only 11% of the kernel cycles spent executing useful instructions.

The user-level component of the application spends time communicating (through shared memory) with the Oracle database, formatting results, and managing memory buffers[4]. Even though it has been optimized heavily for better cache performance the nature of its task is to copy data from one buffer to another and thus it still spends a significant number of its cycles (42%) waiting on the memory hierarchy.

Oracle is known to be a rather heavy weight component, perhaps too heavy weight for an application like ours. However most of Oracle's complexity stems from transaction semantics when data is updated in the database. For a query-only workload like ours Oracle actually offers excellent scalability as evidenced by our experiments.

4.3 The indexing subsystem

The indexer is the most efficient of our system components. It is based on the AltaVista index which has been written with the Alpha processor architecture in mind and some of the inner loops of the application are hand-tuned to exhibit good performance. However even this highly efficient component spends a significant amount of time in the memory hierarchy. A combined 57.1% of the time is spent waiting for memory while only 25.9% of the time is spent executing useful instructions. Careful examination however reveals that most of the waiting time is not part of the application itself but stems from the *pthreads* and *C* libraries with which it is linked. The application also exhibits a substantial amount of kernel time. While some of this is justified due to thread scheduling issues, a significant portion can be attributed to the application building a cache of the indices in memory. Our profiles indicate that approximately one third of the kernel time involves data copying and file management. Our monitored experiments run for approximately 20 minutes and always start with a clean index cache. For longer-running experiment the cost of building the cache will be amortized over many queries and will be much less of a contributing factor to the indexer's performance.

4.4 Network Bandwidth Requirements

Internet bandwidth can be the most expensive resource consumed by an application like Media Search. Our systems consumes network bandwidth both when delivering response pages to user queries and when delivering the multimedia documents themselves. However it is possible to envision an indexing system where responses to user queries are handled by our server while the requests for the multimedia documents are redirected to a separate server on the Internet where the multimedia document originated. We have therefore tried to characterize the network bandwidth requirements

[4]For performance reasons the front-end performs its own memory management rather than relying on the system malloc routines.

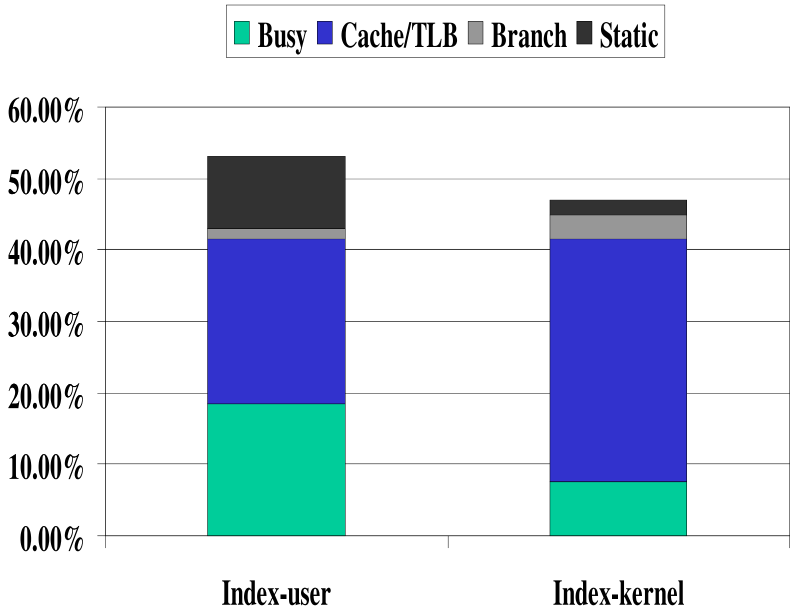

Figure 8: Breakdown of cycles in indexer.

of these two disparate services: responses to user queries and delivery of multimedia streams.

We have measured the capacity of our system to be 1250 simultaneous users, each following the usage model we described in section 3. This works out to 75 page views per second where a page view is in reponse to either an *all-documents query*, *single-document query*, or *browse* request. Another way of looking at this is that our system can service approximately 27 new user sessions per second which in turn will result in 19 multimedia document deliveries. Using the same usage model, we know that users will play approximately 25 seconds out of each delivered document. The bit rate for low-bit video delivery currently is 56Kbits/sec, and a quick calculation shows that the bandwidth requirement for media document delivery is therefore 26.8Mbits/sec.

Bandwidth is also consumed when delivering responses to user queries. Response pages must send a certain amount of HTML and a number of pictures that are representative of the document locations returned by the search. We have instrumented our Web server to measure the number of bytes delivered by each response. The HTML and associated JavaScript of each page we deliver measures anywhere between 9Kbytes and 33Kbytes with an average of 15Kbytes per page. Furthermore, keyframe images are anywhere between 1Kbytes and 4Kbytes with an average of a little over 3Kbytes. Each *all-documents query* and *single-document query* page have a maximum of 10 images and each browse page has a maximum of 5 images. Given their relative frequencies and a maximum response rate of 75 pages per second, we can estimate that our system delivers a maximum of 635 images per second. Thus we can compute the bandwidth requirement for response pages as $15Kbytes/sec * 75 + 635images/sec * 3Kbytes/image = 24.6Mbits/sec$. The most significant component of this cost is picture delivery. One could choose to omit images from a response page thus cutting the bandwidth requirements for the query server by more than 60%.

5 Related Work

Indexing of Web data has been an important topic for the past few years. A large number of search engines index text on the World Wide Web with varying amounts of success on how much of the Web they can cover, how fast they can provide answers, and how relevant those answers are to the users' queries. Multimedia indexing is a relative newcomer in the field of indexing.

One of the early systems that attempted to provide multimedia indexing is the News-on-Demand project, part of the Informedia project at CMU [7]. In this project CNN and other live feeds were digitized, then image analysis was performed to cut the video into shots. Shots were then grouped into scenes using multiple cues. The speech in text format for each scene was then indexed using a standard text indexing system. Users could then send queries to retrieve scenes. More recent work has focused on video summarization [13] and indexing with speech recognition [6, 9, 1, 8]. A commercial product, the Virage Video Cataloguer, performs real-time digitization, shot cut detection, speech recognition, and closed-caption extraction. Off-line, the system provides a keyframe summary of the video over a Web browser; by selecting time periods represented by pairs of keyframes, the user can add annotations to the index. Finally, Maybury's [11] system at MITRE focuses on adding advanced summarization and browsing capabilities to a system that is very similar to Informedia.

The systems cited above, and others, employ the *shot* – an uninterrupted sequence of video – as the basic atomic unit for all additional processing. While shots are clearly important, this type of structuring of the information is not always appropriate. Davenport *et al* [4] introduced the idea of a stream-based representation of video from which multiple segmentation can be generated. In a stream-based representation, the video is left intact, and multi-layered annotations with precise beginning and ending times are stored as associated metadata with the video. Annotations can be a textual representation of the speech, the name of a person, objects in the scene, statistical summaries of a sequence of video, or any other type of data. A stream-based annotation system provides a more flexible framework and can always be reduced to a shot/scene representation by projecting the time intervals of the other annotations against the shot annotations.

A significant advantage of the stream-based approach is that queries are matched against the whole document rather than a subset of it. It is possible for complex queries to not match any one sub-piece of a document, even though the document as a whole may satisfy them. Using the stream-based approach we can find the appropriate document for such complex cases and return a subset of the document that spans multiple document shots. We have used the stream-based approach in Media Search to better capture these complex queries that would not be satisfied by a shot-based approach to the problem. In addition to better service for complex queries, the stream-based approach provides us with the flexibility to define shots differently for different uses of the system even though the underlying mechanisms used to search the documents remain the same. For example, if the system has primarily indexed conversations or speech, as ours has, then what is of interest to the user is the structure of the textual representation of the speech. In this case a single segment per paragraph could be more appropriate than one determined from image analysis. Another example is content reuse and repurposing. Content companies are very interested in reusing old content for new productions. In this case, the semantic content of the story is not of interest. Instead the basic objects, people, and settings are of value. Annotations should mark their appearance and disappearance of these objects from the video. As a final example, consider indexing a symphony video based on instruments and scores. In this case a visually-based temporal segmentation is not appropriate, but one based on the musical structure is. Our system, paired with a Web-based annotation tool we built, can support all these differing styles of segmentation.

6 Conclusion and Future Work

In this paper we have presented our design of a system that can index multimedia documents and can provide access to the indexed documents to large number of clients through a Web interface. Words spoken in the document themselves are used as annotations. These words are extracted from the documents, their exact time is identified and are inserted into a relational database which is then used to build an external text index. Users search the annotations and get in return the document, the location of the particular annotation within the document, and additional annotations that provide context.

By deploying the system within the company we have also produced a usage model based on the observed usage patterns by a large number of clients. We have then used this usage model to conduct a detailed performance analysis of the different system components and we identified memory system performance as the most important performance bottleneck on the indexing side. Essentially we find that caching has only limited success in bridging the gap between CPU speeds and memory speeds for this class of applications and that CPUs spend most of their time waiting for data to arrive from main memory. Surprisingly enough, even with a large amount of content and users, I/O does not appear to be a bottleneck in our system and all disk latency is completely hidden from the user.

We have also shown that the multimedia indexing problem can be relatively successfully reduced to a text indexing problem with some provisions for searching and finding multiple text matches within a document. Even though this approach can be successful there is still need for more complex indexing support. For example it could be useful to search for videos where certain images appear, or audios with certain sounds. These search problems can not be reduced to text searching and the relevant data is often represented as high-dimensional vectors. Efficient indexing technology for high dimensional data is still lacking and presents a formidable challenge for the continued success of multimedia indexing.

Acknowledgements

The CRL Media Search project has benefited from the insights, support, and work of many people, including: Blair Fidler, David Kovalcin, Pedro Moreno, Jean-Manuel Van Tong, Mike Sokolov, Dick Greeley, Chris Weikart, Gabe Mahoney, Bob Supnik, Greg McCane, Katrina Maffey, Peter Dettori, Matthew Moores, Andrew Shepherd, Alan Nemeth, Suresh Masand, Andre Bellotti, S. R. Bangad, Yong Cho, Pat Hickey, Ebrahim Younies, Mike Burrows, Arjen De Vries, and Salim Yusufali.

References

[1] J. Allan, J. Callan, M. Sanderson, J. Xu, and S. Webmann. INQUERY and TREC-7. In *Proceedings of the Seventh Text REtrieval Conference (TREC 7)*, pages 201–216, 1998.

[2] J. M. Anderson, L. M. Berc, J. Dean, S. Ghemawat, M. R. Henzinger, S. T. Leung, R. L. Sites, M. T. Vandervoorde, C. A. Waldspurger, and W. E. Weihl. Continuous Profiling: Where have all the cycles gone? In *Proceedings of the sixteenth ACM Symposium on Operating Systems Principles*, Saint Malo, France, October 1997. 1-14.

[3] L. A. Barroso, K. Gharachorloo, and E. Bugnion. Memory System Characterization of Commercial Workloads. In *Proceedings of the twenty-fifth International Symposium on Computer Architecture*, Barcelona, Spain, June 1998.

[4] G. Davenport, T. Aguierre-Smith, and N. Pincever. Cinematic Primitives for Multimedia. *IEEE Computer Graphics and Applications*, 11(4):67–75, July 1991.

[5] B. Eberman, B. Fidler, R. Iannucci, C. Joerg, L. Kontothanassis, D. Kovalcin, P. Moreno, M. Swain, and J. M. V. Thong. Indexing Multimedia for the Internet. In *Proceedings of the Third International Conference on Visual Information Systems*, Amsterdam, Netherlands, June 1999.

[6] J. Garofolo, E. Voorhees, C. Auzanne, V. Stanford, and B. Lund. 1998 TREC-7 Spoken Document Retrieval Track Overview and Results. In *Proceedings of the Seventh Text REtrieval Conference (TREC 7)*, pages 79–90, 1998.

[7] A. Hauptmann, M. Witbrock, and M. Christel. News-on-demand - an application of informedia technology. In *D-LIB Magazine*, September 1995.

[8] A. Hauptmann, R. Jones, K. Seymore, S. Slattery, M. Witbrock, and M. Siegler. Experiments in Information Retrieval from Spoken Documents. In *Proceedings of the DARPA Broadcast News Transcription and Understanding Workshop*, Lansdowne, VA, February, 1998.

[9] S. E. Johnson, P. Jourlin, G. L. Moore, K. S. Jones, and P. C. Woodland. Spoken Document Retrieval For TREC-7 At Cambridge University. In *Proceedings of the Seventh Text REtrieval Conference (TREC 7)*, pages 191–200, 1998.

[10] K. Keeton, D. Patterson, Y. He, R. Raphael, and W. Baker. Performance Characterization of a Quad Pentium Pro SMP using OLTLP Workloads. In *Proceedings of the twenty-fifth International Symposium on Computer Architecture*, Barcelona, Spain, June 1998.

[11] I. Mani, D. House, and M. Maybury. *Intelligent Multimedia Information Retrieval*, pages chapter 12: Towards Content–Based Browsing of Broadcast News Video. MIT Press, 1997.

[12] T. H. Romer, D. Lee, G. M. Voelker, A. Wolman, W. A. Wong, J. Baer, B. N. Bershad, and H. M. Levy. The Structure and Performance of Interpreters. In *Proceedings of seventh International Conference on Architectural Support for Programming Languages and Operating Systems*, Cambridge, MA, October 1996.

[13] M. Smith and T. Kanade. Video Skimming and Characterization through the Combination of Image and Language Understanding Techniques. In *Proceedings of IEEE Computer Vision and Pattern Recognition*, pages 775–781, San Juan, Puerto Rico, June 1997.

[14] A. P. D. Vries, B. Eberman, and D. Kovalcin. The Design and Implementation of an Infrastructure for Multimedia Digital Libraries. In *In International Database Engineering Applications Symposium*, July 1998.

CRL 99/5

August 1999

Design, Implementation, and Analysis of a Multimedia Indexing and Delivery Server

Leonidas Kontothanassis, Chris Joerg, Michael J. Swain, Brian Eberman, Robert A. Iannucci