StreetTiVo: Using a P2P XML Database System to Manage Multimedia Data in Your Living Room

Ying Zhang[1], Arjen de Vries[1,2], Peter Boncz[1], Djoerd Hiemstra[3], and Roeland Ordelman[3]

[1] Centrum voor Wiskunde en Informatica, Amsterdam, the Netherlands
[2] Delft University of Technology, Delft, The Netherlands
[3] University of Twente, Enschede, The Netherlands
{zhang,arjen,boncz}@cwi.nl, {hiemstra,ordelman}@cs.utwente.nl

Abstract. *StreetTiVo* is a project that aims at bringing research results into the living room; in particular, a mix of current results in the areas of Peer-to-Peer XML Database Management System (P2P XDBMS), advanced multimedia analysis techniques, and advanced information retrieval techniques. The project develops a plug-in application for the so-called Home Theatre PCs, such as set-top boxes with MythTV or Windows Media Center Edition installed, that can be considered as programmable digital video recorders. StreetTiVo distributes compute-intensive multimedia analysis tasks over multiple peers (i.e., StreetTiVo users) that have recorded the same TV program, such that a user can search in the content of a recorded TV program *shortly* after its broadcasting; i.e., it enables *near real-time* availability of the meta-data (e.g., speech recognition) required for searching the recorded content. StreetTiVo relies on our P2P XDBMS technology, which in turn is based on a DHT overlay network, for distributed collaborator discovery, work coordination and meta-data exchange in a volatile WAN environment. The technologies of video analysis and information retrieval are seamlessly integrated into the system as XQuery functions.

1 Introduction

Things are changing in the living room, under the TV set: TV is going digital and consumer electronics gets networked. The so-called "set-top" boxes that are needed for digital television have appeared in the houses of many; and, many of these set-top boxes are rather powerful computers connected to the Internet, running Windows Media Center or its open-source MythTV equivalent. A related trend is the increasing demand for multimedia information access. Presently, "ordinary" people own hundreds of gigabytes of multimedia data, resulting from their digital photo cameras, hard-disk video recorders, etc. However, searching multimedia files is usually restricted to simple look up in the meta-data of files, such as file names and (human-edited) descriptions. More advanced multimedia retrieval requires highly compute-intensive pre-processing of the data (e.g., speech recognition and image processing). As a rough estimation, it takes a moderate computer more than one order of magnitude more time to derive the auxiliary data that would enable better search facilities.

The idea of *StreetTiVo* is to unite the computing power of those Media Center devices (peers) in the living rooms. By distributed and parallel execution of compute-intensive multimedia analysis tasks on multiple peers, near real-time indexing of the content can be provided using just the ordinary hardware available in the network. StreetTiVo divides the Media Center devices into groups and assigns each group a short time slice of a recording (e.g., ten seconds), to run multimedia analysis tools on those time slices only. Thus, the peers form virtual digital streets and are virtual neighbours of each other. Note that StreetTiVo uses the P2P concept in a strictly legal way, as it is not used to distribute the video files themselves. Users can only watch the content that they have recorded themselves. What is exchanged by StreetTiVo are only the results of multimedia analysis of those videos, i.e., generated meta-data.

Summarizing, the goal of the StreetTiVo project is: *unite Media Center devices using P2P technologies to cooperatively run compute intensive multimedia analysis applications just in everybody's living room so that they could produce results in near real-time.*

Imagine for example, if only a tiny fraction of the millions of people recording the Champions League soccer competition would participate in StreetTiVo! Useful media analysis tools would include the transcription of the text spoken by the presenters during the match, but also the cross-media analysis to recognize goals and other exciting moments (e.g., from audio volume and/or camera motion patterns). After each group of media centers has exchanged its partial analysis results with the other groups (that recorded the same match), all StreetTiVo participants obtain the complete set of automatically derived annotations, so that meta-data could be used for direct entry to the most exciting moment(s) of the game, or, the automatic generation of a summary including all highlights.

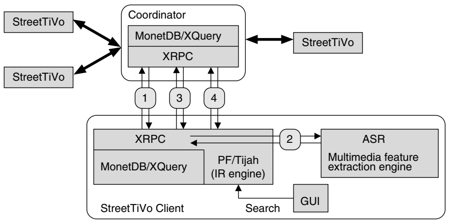
In this paper we describe the P2P data management approach of StreetTiVo in Section 2. The current system architecture is presented in Section 3. We briefly describe the three major components: XRPC, ASR and PF/Tijah. This first version of StreetTiVo was chosen to be simple, so that we could quickly demonstrate the cooperation between XRPC, ASR, PF/Tijah in a non-trivial application setting. A distributed architecture is adopted, in which all StreetTiVo users (i.e., clients) are managed by a central server. In Section 4, we explain the envisioned P2P model and discuss the challenges on our way to make StreetTiVo a truly P2P application. We conclude in Section 5.

2 P2P Data Management

From a network communication perspective, DHT-based overlays have gained much popularity in both research projects and real-world P2P applications [1,2,3,4,5,6,7,8,9,10,11,12]. DHT networks have proven to be efficient and scalable (guarantees $O(logN)$ scalability) in volatile WAN environments [6,13].

From a data management perspective, XML has become the de facto standard for data exchange over the Internet, and XQuery the W3C standard for querying XML data. The infrastructure of StreetTiVo is therefore provided by *MonetDB/XQuery** [14], a P2P XML DBMS that supports distributed evaluation of XQuery[15] queries over DHT networks [1,3,4,6]. Each peer runs an XDBMS, *MonetDB/XQuery* [16], to manage its local data. Communication among peers is done by remote execution of XQuery functions using *XRPC* [17,18], a simple XQuery extension for Remote Procedure Calls (RPC) that enables *efficient* distributed querying of *heterogeneous* XQuery data sources.

The current implementation of StreetTiVo as XQuery expressions applies a Dutch automatic speech recognition system (*ASR*) [19] to the recorded programmes, and provides a full-text retrieval service of the ASR output by employing the MonetDB/XQuery extension *PF/Tijah* [20]. When a TV program is broadcast, all participants (peers that are recording this TV program) jointly extract the (Dutch) spoken text using the ASR component, and exchange local partial ASR results with each other. ASR produces XML documents containing speech texts and some meta-data, for example, start/end timestamp of a sentence in the video file. Each participant stores *all* resulting XML documents of the recording in its local MonetDB/XQuery that can then be queried using PF/Tijah. The meta-data of the retrieved sentences provide sufficient information for the StreetTiVo GUI to display only the desired video fragment.



1. register–recording($channel, $startDT, $duration, $clntAddr): fragment
2. start–asr($progID, $fragID, $start, $end)
3. add–finished–fragment($progID, $fragID, $text, $clntAddr): fragment
4. get–speech–text($progID, $fragIDs)

Fig. 1. StreetTiVo architecture: client-server model

```
⟨recording progID="bbc1_20080425_200000"⟩
  ⟨participants⟩
    ⟨participant host="x.example.org" /⟩
    ⟨participant host="y.example.org" /⟩
  ⟨/participants⟩
  ⟨fragments⟩
    ⟨fragment fragID="1" start="0" end="10"⟩
      ⟨owner host="x.example.org" /⟩⟨/fragment⟩
    ⟨fragment fragID="2" start="10" end="25"⟩
      ⟨assignee host="y.example.org" /⟩⟨/fragment⟩
    ⟨fragment fragID="3" start="25" end="30"⟩
      ⟨assignee host="x.example.org" /⟩⟨/fragment⟩
    ⟨fragment fragID="4" start="30" end="38"/⟩
  ⟨/fragments⟩
⟨/recording⟩
```

Fig. 2. Information maintained for each recording

3 StreetTiVo Architecture

The current version of StreetTiVo uses a simple client-server network model, as shown in Figure 1. In this setup, we assume that the coordinator is a reliable host, while the clients join and leave unpredictably (similar to the early work of [21]). While our next step will be to replace this model with a more sophisticated DHT-based P2P model, we first detail the current implementation.

Each peer runs a MonetDB/XQuery server and communicates with other peers via XRPC, by sending SOAP XRPC request/response messages. The central StreetTiVo coordinator is responsible for registration of recordings, and the generation and distribution of ASR tasks. For each recording, the coordinator maintains a list of participating peers and a list of tasks (called fragments). A recording is divided into short fragments (usually several seconds) to be analyzed parallelly by the participants. For each fragment, the coordinator maintains if it is being processed by a peer (i.e., it has an assignee), or if its speech text is already available (i.e. it has an owner). All meta-data are in XML format and stored in MonetDB/XQuery.

Example 1. The XML snippet in Figure 2 shows the recording element for the TV program bbc1_20080425_200000. The *progID* is determined by channel, date and start time. The TV program is recorded by two peers and is divided into 4 fragments. The attributes start and end of a fragment indicate the relative start/end timestamps of the fragment in the video file. Fragments are assigned to the participants in the order they register, so initially fragments 1 and 2 are assigned to the hosts x.example.org and y.example.org, respectively. So far, x.example.org has finished analyse fragment 1 (indicated as the owner of the fragment) and has been assigned a new job fragment 3. The host y.example.org is still processing fragment 2. Since there are no more participants, fragment 4 is waiting to be assigned.

All interfaces between coordinator, clients and the local ASR engine have been defined as XQuery functions. A small Java program implements the XQuery function (start-asr() in Figure 1) that triggers the ASR engine. The interaction between one StreetTiVo client and the coordinator is shown in details. Collaborative speech recognition works as follows.

Step ①. When a TV program is scheduled for recording, the StreetTiVo client sends an XRPC request to the coordinator to execute the function register-recording(). The coordinator responds with a fragment, which has not been processed by any participants, and inserts the client as an assignee into the fragment element. For reliability reason, each fragment is assigned to multiple clients.

If the request is the first registration for a TV program, the coordinator first needs to generate ASR tasks. An easy way to do this is to divide the whole recording into equal sized fragments. To get high quality ASR result, the ASR segmenter should be used, which is able to filter out the audio that do not contain speech and generates fragments accordingly (see Section 3.2). Information provided by the ASR segmenter ensures that the ASR speech recognizer produces more accurate results. However, the better quality comes at a high cost of speed, because the coordinator must record the TV program itself and run ASR segmenter *afterwards*. So, there is a trade-off between speed and quality.

Step ②. Upon receipt of the response from the coordinator (in *Step* ①), the StreetTiVo client starts its local ASR engine to analyze the fragment specified in the response message, by calling the interface function start-asr().

Step ③. After the ASR engine has finished analyzing a fragment, the StreetTiVo client reports this by calling add-finished-fragment() on the coordinator and passing among others the retrieved text as parameter.

Step ④. After having finished one task, the StreetTiVo client is expected to request a new task (get-job()), until the coordinator responds with an empty task, which might mean that there are sufficient number of assignees for each fragment, or that the coordinator has received the ASR results of all fragments.

Step ⑤. If there are no new tasks, the StreetTiVo client waits for some predefined time to give the other participants the opportunity to finish their ASR tasks, and then attempts to retrieve the speech text of the missing fragments. The StreetTiVo client can directly ask the coordinator for the missing fragments (get-speech-text()), since all ASR results are also stored at the coordinator, but the preferred procedure is to just call the coordinator's get-fragments() function (not shown) to find out what participant owns which ASR result, and retrieve the fragments' texts from those nodes.

Once the ASR results are locally available, StreetTiVo users can search for video fragments by entering keywords in the GUI. The keywords are subsequently translated into Tijah queries. PF/Tijah returns matching sentences ranked by their estimated probability of relevance.Each sentence is tagged with its relative start/end timestamps in the video file, this way, the desired video fragments can be retrieved. In summary, StreetTiVo has three major components: XRPC takes care of communication among peers, ASR provides video analysis functions, and PF/Tijah enables retrieval of video fragments using keywords. In the remainder of this section, we give a brief overview of these techniques.

3.1 XRPC: Distributed XQuery Processing

XQuery 1.0 [15] only provides a *data shipping* model for querying XML documents over the Internet. The built-in function fn:doc() fetches an XML document from a remote peer to the local server, where it subsequently can be queried. The recent W3C Candidate Recommendation *XQuery Update Facility* (XQUF) [22] introduces a built-in function fn:put() for remote storage of XML documents, which again implies data shipping. There have been various proposals to equip XQuery with *function shipping* style distributed querying abilities [23,24,25]. On the syntax level, we consider our XRPC proposal an incremental development of these. XRPC adds RPC to XQuery in the most simple way: adding a destination URI to the XQuery equivalent of a procedure call (i.e. function application).

Remote function applications in XRPC take the XQuery syntax: execute at {Expr}{FunApp (ParamList)}, where *Expr* is an XQuery xs:string expression that specifies the URI of the peer on which the function *FunApp* is to be executed. As a running example, we assume a set of XQuery DBMS (peers) that each store a movie database in an XML document filmDB.xml with contents similar to:

```
⟨films⟩
  ⟨film⟩⟨name⟩The Rock⟨/name⟩⟨actor⟩Sean Connery⟨/actor⟩⟨/film⟩
  ⟨film⟩⟨name⟩Green Card⟨/name⟩⟨actor⟩Gerard Depardieu⟨/actor⟩⟨/film⟩
⟨/films⟩
```

We assume an XQuery module `film.xq` stored at the host `example.org` that defines a function `filmsByActor()`:

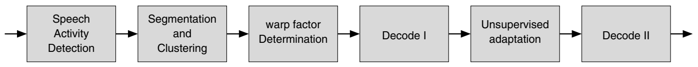

Speech Activity Detection
Segmentation and Clustering
warp factor Determination
Decode I
Unsupervised adaptation
Decode II

Fig. 3. Overview of the ASR decoding system

```
module namespace file="films";
declare function film:filmsByActor($actor as xs:string) as node()*
{ doc("filmDB.xml")//name[../actor=$actor] };
```

With XRPC, we can execute this function on a remote peer, e.g. x.example.org, to get a sequence of films in which Sean Connery plays in the film database stored on the remote peer.

```
import module namespace f="films" at "http://example.org/film.xq";
⟨films⟩ { execute at {"xrpc://x.example.org"} {f:filmsByActor("Sean Connery")} } ⟨/films⟩
```

which yields: ⟨films⟩⟨name⟩The Rock⟨/name⟩⟨/films⟩. We introduce here a new xrpc:// network protocol, accepted in the destination URI of execute at, to indicate a peer's ability of handling XRPC queries. The generic form of such URIs is xrpc://⟨host⟩[:port] [/[path]], where xrpc:// is the network protocol, ⟨host⟩[:port] identifies the remote peer, and [/[path]] is an optional local path at the remote peer.

The SOAP XRPC Protocol. The design goal of XRPC is to create a distributed XQuery mechanism with which *different* XQuery engines at different sites can jointly execute queries. This implies that our proposal also encompasses a *network protocol*, SOAP XRPC, which uses the Simple Object Access Protocol (SOAP) [26] (i.e. XML messages) over HTTP. XML is ideal for distributed environments (think of character encoding hassles, byte ordering), XQuery engines are perfectly equipped to process XML messages, and an XML-based message protocol makes it trivial to support passing values of any type from the XQuery Data Model [27]. The choice for SOAP brings as additional advantages seamless integration of XQuery data sources with web services and Service Oriented Architectures (SOA) as well as AJAX-style GUIs. The complete specification of the SOAP XRPC protocol can be found in [17]. Here we show, as an example, the XRPC request message that should be generated for the query above:

```
⟨?xml version="1.0" encoding="utf-8"?⟩
⟨env:Envelope xmlns:xrpc="http://monetdb.cwi.nl/XQuery"
              xmlns:env="http://www.w3.org/2003/05/soap-envelope"
              xmlns:xs="http://www.w3.org/2001/XMLSchema"
              xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
              xsi:schemaLocation="http://monetdb.cwi.nl/XQuery
                                  http://monetdb.cwi.nl/XQuery/XRPC.xsd"⟩
  ⟨env:Body⟩
    ⟨xrpc:request xrpc:module="films" xrpc:method="filmsByActor" xrpc:arity="1" xrpc:iter-count="1"
                  xrpc:updCall="no" xrpc:location="http://x.example.org/film.xq"⟩
      ⟨xrpc:call⟩
        ⟨xrpc:sequence⟩
          ⟨xrpc:atomic-value xsi:type="xs:string"⟩Sean Connery⟨/xrpc:atomic-value⟩
        ⟨/xrpc:sequence⟩
      ⟨/xrpc:call⟩
    ⟨/xrpc:request⟩
  ⟨/env:Body⟩
⟨/env:Envelope⟩
```

3.2 ASR: I Know What You Said

Automatic Speech Recognition (ASR) supports the conceptual querying of video content and the synchronization to any kind of textual resource that is accessible, including other annotations for audiovisual material such as subtitles. The potential of ASR-based indexing has been demonstrated most successfully in the broadcast news domain. Typically large vocabulary speaker independent continuous speech recognition (LVCSR) is deployed to this end.

The ASR system deployed in StreetTiVo was developed at the University of Twente and is part of the open-source SHoUT speech recognition toolkit[1]. Figure 3 gives an overview of the ASR decoding system. Each step provides the input for the following step. The whole process can be roughly divided into two stages. During the first stage, the *Speech Activity Detection* (SAD) is used to filter out the audio parts that do not contain speech. This step is crucial for the performance of the ASR system to avoid that it tries to recognize non-speech audio that is typically found in recorded TV programs such as music, sound effects or background noise with high volume (traffic, cheering audience, etc). After SAD, the system tries to figure out 'who spoke when', a procedure that is typically referred to as speaker diarization. In this step, the speech fragments are split into segments that only contain speech from one single speaker. Each segment is labelled with its corresponding speaker ID. Next, for each segment the vocal tract length (VTLN) warping factor is determined for vocal tract length normalisation. Variation of vocal tract length between speakers makes it harder to train robust acoustic models. In the *SHoUT* system, normalisation of the feature vectors is obtained by shifting the Mel-scale windows by a certain warping factor during feature extraction for the first decoding step.

After having cleaned up the input sound and gained sufficient meta information, speech recognition can be started in the second stage. Decoding is done using the HMM-based Viterbi decoder. In the first decoding iteration, triphone VTLN acoustic models and trigram language models are used. For each speaker, a first best hypothesis aligned on a phone basis is created for unsupervised acoustic model adaptation. Optionally, for each file a topic specific language model can be generated based on the input of first recognition pass. The second decoding iteration uses the speaker adapted acoustic models and the topic specific language models to create the final first best hypothesis aligned on word basis. Also, for each segment, a word lattices is created. A more detailed description of each step can be found in [19].

3.3 PF/Tijah: XML Text Search

PF/Tijah [20] is another research project run by the University of Twente with the goal to create a flexible environment for setting up search systems by integrating MonetDB/XQuery that uses the Pathfinder compiler [28] with the Tijah

[1] For information on the use of the SHoUT speech recognition toolkit see `http://wwwhome.cs.utwente.nl/~huijbreg/shout/index.html`

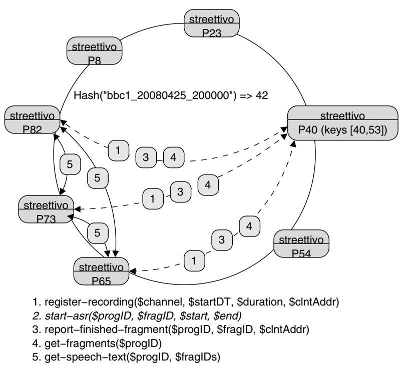
XML Information Retrieval (IR) system [29]. The main features supported by PF/Tijah include the following:

- Retrieving arbitrary parts of textual data, unlike traditional IR systems for which the notion of a document needs to be defined up front by the application developers. For example, if the data consist of scientific journals one can query for complete journals, journal issues, single articles, sections from articles or paragraphs *without* adapting the index or any other part of the system configuration;
- Complex scoring and ranking of the retrieved results by means of so-called *Narrowed Extended XPath* (NEXI) [30] queries. NEXI is a query language similar to XPath that only supports the descendant and the self axis step, but that is extended with a special `about()` function that takes a sequence of nodes and ranks those by their estimated probability of relevance to the query;
- PF/Tijah supports incremental indexing: when new ASR fragments are added to the database, their text will be automatically indexed by PF/Tijah, *without* the need to re-index the entire database from scratch;
- search combined with traditional database querying, including for instance joins on values. As an example, one could search for programmes mentioning "football" that are broadcast on the same channel as programmes mentioning "crime".

StreetTiVo inserts fragments containing the transcripts of ASR whenever they are available. Therefore, they will not be nicely grouped per programme in the database, nor will they be in chronological order. The combination XQuery and NEXI text search enables StreetTiVo to search matching fragments, combine the fragments with the same programme identifier, combine their scores (or take the score of the best matching fragment), rerank programmes by the scores of their fragments, and display the matching programmes along with their best matching fragments: all of this is done in one query.

4 Next Steps

Our next step in the development of StreetTiVo is to replace the client-server model with MonetDB/XQuery* [14], which integrate the P2P data structure Distributed Hash Tables (DHTs) into XQuery (see Figure 4). A DHT [1,3,4,6] provides *(i)* robust connectivity (i.e., it tries to prevent network partitioning), *(ii)* high data availability (i.e., prevent data loss if a peer goes down by automatic replication), and *(iii)* a



1. register–recording($channel, $startDT, $duration, $clntAddr)
2. start–asr($progID, $fragID, $start, $end)
3. report–finished–fragment($progID, $fragID, $clntAddr)
4. get–fragments($progID)
5. get–speech–text($progID, $fragIDs)

Fig. 4. StreetTiVo architecture: P2P model

scalable $(key, value)$ storage mechanism with $O(log(N))$ cost complexity, where N is the number of peers in the network. A number of P2P database prototypes have already used DHTs [5,7,8,10,11].

In a StreetTiVo system using a DHT model, peers are managed by a DHT ring. There is no single StreetTiVo coordinator. All peers are unreliable and each can be both a coordinator and a client, thus, each peer must additionally support the functions provided by the coordinator, as discussed in Section 3. The process to collectively speech extraction using ASR is similar as in the client-server model, except several small changes:

Example 2. Figure 4 shows an example scenario, in which three peers N65, N73 and N82 will record the TV program with *progID*="bbc1_20080425_200000". All participants use the same hash function to calculate the hash of the *progID*, which is 42 here. Since the peer P40 is responsible for keys $[40, 53]$, it is chosen as the coordinator for this TV program. The participating peers register the recording at P40 (Step ①). Generating ASR tasks is still done by the (temporary) coordinator, P40. When a peer finished an ASR task, it reports this at the coordinator (Step ③), but *without* sending the extracted text. All participants will repeat steps ④, ② and ③ to process more fragments, until there are no more ASR tasks. Finally, the participants exchange the ASR results by first finding the owner of each fragments from the coordinator (Step ⑤), and then retrieving the missing ASR results from each other (Step ⑤).

Note that, in the DHT model, the availability of the recording meta-data (i.e., the recording elements maintained by a coordinator as discussed in Section 3) is guaranteed thanks to the automatic replication facility provided by the underlying DHT network, that is, all data on a peer managed by the DHT network are replicated on the peer's predecessors and successors. Thus, if peer P40 would fail in our example, all request with key 42 would be routed by the DHT network to P23 or P54. Also note that, the ASR results are not managed by the DHT network. Basically, all StreetTiVo peers can retrieve these data, but only the peers that have recorded the particular TV program can display the video. The availability of the ASR results is affected by the number of participants (more participants $\Rightarrow$ higher availability). However, this is not a crucial issue, since every StreetTiVo peer is able to run ASR on a missing fragments.

The challenges in integration of XQuery and DHT are:

(i) how a DHT should be exploited by an XQuery processor,
(ii) if and how the DHT functionality should surface in the query language.

In MonetDB/XQuery*, we propose to avoid any additional language extensions, but rather introduce a new dht// network protocol, accepted in the destination URI of fn:doc(), fn:put() and execute at. The generic form of such URIs is dht://dht_id/key, where dht:// is the network protocol, *dht_id* is the ID of the DHT network to be used. Such an ID is useful to allow a P2P XDBMS to participate in multiple (logical)

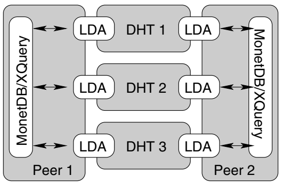


Fig. 5. Tight coupling

DHTs simultaneously (see Figure 4). The *key* is used to store and retrieve values in the DHT.

In the architecture shown in Figure 4, we run the DHT as a separate process called the Local DHT Agent (LDA). Each LDA is is connected to one DHT *dht_id*. We propose a *tight coupling* between the DHT network and the XDBMS [14], in which each DHT peer uses its local XDBMS to store the data (i.e. XML documents) and the local XDBMS uses its underlying DHT network to route XQuery queries to remote peers for execution (i.e. pass XRPC requests to the LDA). A positive side-effect of this tight coupling is that the DBMS gets access to the information internal to the P2P network. This information (e.g. peer resources, connectivity) can be exploited in query optimization. To realize this coupling, we need to extend the DHT API (put() and get()) with one new method: xrpc (key, q, m, $f(ParamList)$) : item()*, where $f(ParamList)$ is the XQuery function that is to be executed on a remote DHT peer determined by *key*. The parameters q and m specify XQuery module, in which the function f_r is defined and the location of the module file. With this method, an XRPC call on a peer p_0 to a dht://dht_x/key_y URI is handled as follows:

1. The XRPC request($q, m, f, ParamList$) is passed to the Local DHT Agent lda_0^x of p_0, which in turn passes the request to the DHT network dht_x.
2. The DHT dht_x routes the request using the normal DHT routing mechanism to the peer p_i responsible for key_y.
3. When the LDA lda_i^x on p_i receives such a request, it performs an XRPC call containing the same request to the MonetDB/XQuery instance on p_i.
4. When lda_i^x receives the response message, it transports the response back via dht_x to the query originator p_0.

Use Cases. Below we show how two main StreetTiVo functions can be implemented as XQuery module functions, which then can be executed using XRPC and the tightly coupled DHT semantics.

(i) Collaborator Discovery. In StreetTiVo, every TV program has a unique identifier *progID*, and for each recorded TV program a recording element is maintained by the peer responsible for the key *hash*(*progID*) with lists of participants and fragments. If a peer is going to record the TV program "bbc1_20080425_200000", it should register the recording at the coordinator of this TV program. This can be done by the following XRPC call:

```
import module namespace stv = "streettivo" at "http://example.org/stv.xq";

let $key := hash("bbc1_20080425_200000"),
    $dst := fn:concat("dht://dhtx/", $key)
return execute at {$dst} {stv:register-recording(bbc1, "2008-04-25T20:00:00", "1H", "x.example.org")}
```

(ii) Distributed Keyword Retrieval. Assume a StreetTiVo user wants to search in today's newscast "bbc1_20080425_20-0000", he/she has recorded, for video fragments that were about the situation in Tibet, but the ASR results are not completely available (yet) on his/her local machine. Then the search request might be sent to other StreetTiVo peers that have recorded the same newscast.

The following pseudo-code first retrieves the list of fragments from the coordinator, and then sends a search request to each peer that owns ($frags//owner/@host) the ASR results of a fragment:

```
import module namespace stv = "streettivo" at "http://example.org/stv.xq";

let $key := hash("bbc1_20080425_200000"),
    $dst := fn:concat("dht://dht_x/", $key)
    $frags := execute at {$dst} {stv:get-fragments(bbc1, "2008-04-25T20:00:00"}
return for $p in $frags//owner/@host return
        execute at {$p} {stv:search("situation in Tibet")}
```

5 Conclusion

In this paper, we have described StreetTiVo, a P2P Information Retrieval system that enables near real-time search in video contents by just using existing hardware in the living rooms to collectively run compute-intensive video analysis video content analysis tools.

Thanks to its implementation in a high-level declarative database language, it is straightforward to extend StreetTiVo with other types of functionality. We plan to complement the current media analysis with image processing techniques to automatically detect celebrities in news broadcasts or goals in soccer matches. Maybe even more interesting is that StreetTiVo users can also easily share their own human made annotations. For example, people usually schedule a recording several minutes before/after the start/end of the to be recorded TV program, to prevent missing part of the program. If just one StreetTiVo user has annotated the exact start/end timestamp of the TV program, the information can be shared in the platform with other StreetTiVo users who have recorded the same program, and the unnecessary parts of their recordings can be removed transparently.

Acknowledgement

The authors would like to thank Marijn Huijbregts for creating the SHoUT ASR toolkits, Erwin de Moel for developing the GUI for Windows Vista Media Center, Henning Rode and Jan Flokstra for their contribution to PF/Tijah, and Niels Nes, Matthijs Mourits and Roberto Cornacchia for their help in organizing the digital video recording. MultimediaN is funded by the Dutch government under contract BSIK 03031.

References

1. Aberer, K.: P-Grid: A Self-Organizing Access Structure for P2P Information Systems. In: CooplS (2001)
2. Rowstron, A.I.T., Druschel, P.: Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In: Guerraoui, R. (ed.) Middleware 2001. LNCS, vol. 2218, p. 329. Springer, Heidelberg (2001)
3. Ratnasamy, S., Francis, P., Handley, M., Karp, R., Schenker, S.: A Scalable Content-Addressable Network. In: SIGCOMM (2001)
4. Stoica, I., et al.: Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications. In: SIGCOMM (2001)

5. Huebsch, R., et al.: Querying the Internet with PIER. In: VLDB (2003)
6. Rhea, S.C., Geels, D., Roscoe, T., Kubiatowicz, J.: Handling Churn in a DHT. In: USENIX Annual Technical Conference, General Track (2004)
7. Rao, W., Song, H., Ma, F.: Querying XML data over DHT system using xPeer. In: Jin, H., Pan, Y., Xiao, N., Sun, J. (eds.) GCC 2004. LNCS, vol. 3251, pp. 559–566. Springer, Heidelberg (2004)
8. Bonifati, A., Chang, E.Q., Lakshmanan, A.V.S., Ho, T., Pottinger, R.: HePToX: marrying XML and heterogeneity in your P2P databases. In: VLDB (2005)
9. Rhea, S., Godfrey, B., Karp, B., et al.: OpenDHT: a public DHT service and its uses. In: SIGCOMM (2005)
10. Huebsch, R., Chun, B.N., et al.: The Architecture of PIER: an Internet-Scale Query Processor. In: CIDR (2005)
11. Karnstedt, M., Sattler, K.U., et al.: UniStore: Querying a DHT-based Universal Storage. Technical report, EPFL (2006)
12. Zhao, B.Y., et al.: Tapestry: A Resilient Global-scale Overlay for Service Deployment. IEEE J-SAC 22(1) (January 2004)
13. Rhea, S., et al.: Fixing the Embarrassing Slowness of OpenDHT on PlanetLab. In: USENIX WORLDS 2005 (2005)
14. Zhang, Y., Boncz, P.: Integrating XQuery and P2P in MonetDB/XQuery*. In: EROW (January 2007)
15. Boag, S., et al.: XQuery 1.0: An XML Query Language W3C Candidate Recommendation, June 8 (2006)
16. Boncz, P., et al.: MonetDB/XQuery: A Fast XQuery Processor Powered by a Relational Engine. In: SIGMOD (June 2006)
17. Zhang, Y., Boncz, P.: XRPC: Interoperable and Efficient Distributed XQuery. In: VLDB (September 2007)
18. Zhang, Y., Boncz, P.: Distributed XQuery and updates processing with heterogeneous XQuery engines. In: SIGMOD (2008)
19. Huijbregts, M., Ordelman, R., de Jong, F.: Annotation of heterogeneous multimedia content using automatic speech recognition. In: Falcidieno, B., Spagnuolo, M., Avrithis, Y., Kompatsiaris, I., Buitelaar, P. (eds.) SAMT 2007. LNCS, vol. 4816, pp. 78–90. Springer, Heidelberg (2007)
20. Hiemstra, D., Rode, H., van Os, R., Flokstra, J.: PFTijah: text search in an XML database system. In: OSIR (August 2006)
21. de Vries, A., Eberman, B., Kovalcin, D.: The design and implementation of an infrastructure for multimedia digital libraries. In: IDEASapos (July 1998)
22. Chamberlin, D., et al.: XQuery Update Facility (W3C Working Draft 11 (July 2006)
23. Onose, N., Siméon, J.: XQuery at your web service. In: WWW (2004)
24. Re, C., et al.: Distributed XQuery. In: IIWeb (September 2004)
25. Thiemann, C., Schlenker, M., Severiens, T.: Proposed Specification of a Distributed XML-Query Network. CoRR cs.DC/0309022 (2003)
26. Mitra, N., Lafon, Y.: SOAP Version 1.2 Part 0: Primer W3C Recommendation, June 24 (2003), `http://www.w3.org/TR/2003/REC-soap12-part0-20030624`
27. Fernández, M., et al.: XQuery 1.0 and XPath 2.0 Data Model (XDM) W3C Recommendation, January 23 (2007), `http://www.w3.org/TR/xpath-datamodel`
28. Grust, T., Sakr, S., Teubner, J.: XQuery on SQL Hosts. In: VLDB (2004)
29. List, J., et al.: Tijah: Embracing information retrieval methods in XML databases. Information Retrieval Journal 8(4), 547–570 (2005)
30. O'Keefe, R.A., Trotman, A.: The simplest query language that could possibly work. In: INEX (2004)