Une solution client/serveur pour l'analyse de corpus audiovisuels.

Arnaud Dupuis, Cédric Dufouil

IRISA / INRIA Rennes équipe de recherche TEXMEX,
Campus universitaire de Beaulieu, 35042 Rennes Cedex
arnaud.dupuis@irisa.fr
http://www.irisa.fr/texmex/people/dupuis/index_fr.htm

Abstract

Le traitement de grands corpus audiovisuels nous impose de concevoir une infrastructure informatique adaptée aux fortes contraintes techniques associées à ce type de données. Tout d'abord, les traitements vidéos sont généralement de gros consommateurs de puissance de calcul, de plus la gestion et le stockage des importants volumes d'informations peut rapidement s'avérer être complexe à mettre en oeuvre (notamment pour des problèmes de droits à la copie). Nous avons donc développé une plate-forme multimédia simple d'utilisation, accessible quel que soit le système d'exploitation de la machine cliente et gérant les droits d'accès aux corpus audiovisuels. Dans cet article, nous présentons notre solution client/serveur conçue pour rendre les opérations de gestion et de décodage des vidéos totalement transparentes pour les applications clientes. Enfin, nous présentons quelques résultats expérimentaux illustrant les performances de notre serveur avec n clients.

Mots-clés *: MPEG, analyse de vidéos, décodage de vidéos, référence temporelle, plate-forme multimédia.*

Abstract

Processing huge corpuses of audiovisual content enforces the need to create an adapted infrastructure. There is three manly technical constraints: First, data managment and storage aspects represents a crucial point. Secondly, video analysis tools consume a large part of the computer processing power. And third, we consider that such a platform must be easily accessible, independently of the operating system used by the client. In This paper, we detail the client/server solution we have built to facilitate our developments of processing and indexing video algorithms. With this software solution, video storage and decodage are totally transparent for the client application. Finally, we present in this paper some experimental results of our server performances with n clients.

Key-words*: MPEG, video analysis, video decodage, time reference, multimedia platform.*

1 INTRODUCTION

Nos activités de recherche dans le domaine de l'indexation multimédia nous contraignent à manipuler d'importants volumes de données nécessitant d'importantes capacités de stockage (une journée de télévision occupe environ 40Go en MPEG-2 4Mb/s, auxquels il faut ajouter les données intermédiaires de calcul et les résultats d'analyse). À ces problèmes de stockage s'ajoute la protection des droits d'auteurs sur les données audiovisuelles, qui interdisent souvent la diffusion ou duplication sans accord préalable. Sur un plan plus technique, le traitement de documents multimédias pose de nombreux problèmes de précision lors de l'accès aux informations de natures différentes (image, son, texte). Typiquement, l'accès aléatoire (rechercher une image précise sans lire les images précédentes) au sein d'un flux vidéo MPEG, pourtant primordiale dans un processus de développement d'algorithmes de traitement et d'analyse automatique de vidéos, peut rapidement s'avérer être complexe à mettre en oeuvre.

Pour répondre à ces problématiques, nous avons conçu et développé un serveur de documents multimédias, destiné à héberger nos corpus audiovisuels, et accessibles aux utilisateurs par le biais d'un login et un mot de passe. Nous avons également défini une norme d'accès aux données, indépendante des fréquences d'échantillonnage du son et des images, et offrant la possibilité d'accéder simplement et aléatoirement aux différentes séquences composant un corpus audiovisuel réparti sur un ou plusieurs fichiers.

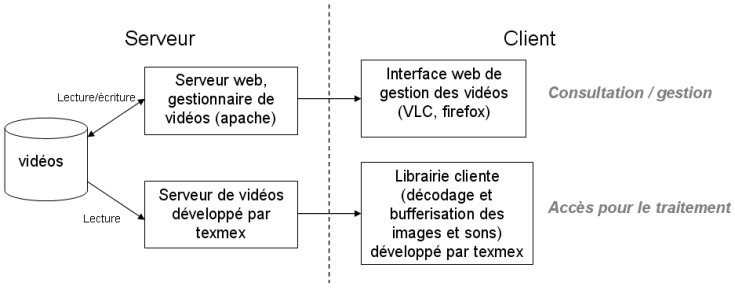

Figure 1: Schéma global du système.

Dans cet article technique, nous présentons notre solution d'accès à des vidéos stockées sur un serveur distant. Cette solution (figure 1) est composée d'un gestionnaire de vidéos présenté dans la section suivante et d'un couple client-serveur dédié au traitement, que nous présentons dans le paragraphe 3. Des résultats expérimentaux sont proposés dans la section 4, enfin nous conclurons et présenterons nos perspectives.

2 GESTIONNAIRE DE VIDÉOS.

L'hébergement de vidéos sur un serveur distant impose de développer une interface sécurisée de gestion des corpus vidéos. Notre interface propose aux utilisateurs (identifiés par mot de passe), d'accéder simplement aux vidéos, d'enregistrer des programmes télévisés, de visualiser les vidéos hébergées sur le serveur, ou encore de naviguer au sein de ces dernières par le biais de fichiers d'annotation TV-anytime [5]. Trois contraintes techniques fortes ont conditionné le développement de cette interface de gestion:

1. La première contrainte technique était de pouvoir gérer la base de vidéos par le réseau, et ce, quel que soit le système d'exploitation de la station de travail « cliente » (Windows, Linux ou Mac OS X). La solution que nous avons retenue consistait à développer une interface web de gestion, à accès restreint et disponible uniquement en interne.

2. La copie des vidéos sur une machine cliente étant exclue, une deuxième contrainte technique forte était de pouvoir prévisualiser les vidéos en *streaming*. De nombreuses solutions propriétaires existent, toutefois, elles imposent généralement un réencodage des flux à des formats spécifiques (ram, wmv...), peu satisfaisants dans notre cas d'étude. En effet, les vidéos dont nous disposons sont encodées dans un format MPEG (1 ou 2), aux normes parfaitement connues, les réencoder nous imposerait d'utiliser un format propriétaire dont nous ne maîtrisons pas le décodage et provoquerait probablement une dégradation sensible de la qualité des images.
   La solution que nous préconisons consiste à diffuser nos vidéos MPEG en utilisant un simple protocole http géré par VLC[1] (logiciel libre) et un serveur de fichiers Apache. L'utilisation conjointe sur la machine cliente des logiciels firefox et VLC, nous permet d'éviter l'utilisation du couple internet explorer / windows média player et nous offre par conséquent la possibilité d'utiliser ce site sous linux, windows et mac, et ce, quel que soit le format de la vidéo (VLC gérant de nombreux codecs).

3. Enfin, la dernière contrainte technique nous imposait de pouvoir enregistrer plusieurs semaines de flux télévisés en continu. Il est à noter que l'utilisation de cartes d'acquisition vidéo grand public, initialement prévues pour enregistrer uniquement quelques heures consécutives, impose de réaliser une gestion mémoire adaptée à de longs enregistrements.

[1] VLC est un logiciel libre initialement développé à l'école centrale de Paris. http://www.videolan.org/vlc/

3 ACCÈS ALÉATOIRE AUX DONNÉES VIDÉO.

3.1 Accès aux données et synchronisation.

Une des principales difficultés rencontrées lors du développement de systèmes de gestion et de traitement de données multimédia, réside dans la synchronisation temporelle pour un accès aléatoire aux données [3]. Dans cette section, nous présentons quelques définitions que nous utilisons pour synchroniser nos données de contenu (vidéo, audio, textuelles...) ou de description (résultats expérimentaux).

3.1.1 Notions d'intervalle et de segment temporel.

Nous considérons qu'un intervalle représente une séquence continue extraite d'une vidéo, et qu'un segment temporel est composé de plusieurs intervalles consécutifs ou non.

Selon Moënne-Loccoz dans [4], un intervalle temporel continu extrait d'un flux multimédia S, peut être défini par $I_a^b(S) = [a, b[, \forall a, b$ avec $1 \leq a < b \leq T_S$ et T_S correspondant à la longueur totale du flux exploitable S. Un segment temporel peut alors être défini par une composition d'intervalles temporels tel que:

$$I_{(a_k)}^{(b_k)}(S) = \bigcup_k I_{a_k}^{b_k}, \forall k = 1, \ldots, n \tag{1}$$

avec n correspondant au nombre d'intervalles temporels.

Les intervalles et segments temporels ainsi définis permettent de fixer le cadre théorique d'accès aux données, toutefois la nature hétérogène des éléments composant un document multimédia nécessite de définir un espace commun autorisant des accès synchronisés.

3.1.2 Utilisation de références temporelles (timeref).

Les problèmes de synchronisation liés aux fréquences d'échantillonnage des images et du son, nous contraignent à utiliser une référence commune permettant un accès aléatoire à un document multimédia. Une approche consiste à estimer que cette référence peut être définie par le plus petit commun multiple des fréquences d'échantillonnage des différents médias. Ainsi, nous posons la valeur référence $valref = 14112000$ représentant une seconde et qui intègre notamment les fréquences les plus utilisées de 25Hz, 30Hz, 44.1kHz et 48kHz. Cette valeur permet notamment de simplifier les accès aux données en utilisant une simple valeur entière, quelle que soit la nature des données à traiter (son, image, texte...).

Ainsi, supposons l'exemple fictif d'une vidéo codée en MPEG-2 à 25 images par secondes et une fréquence d'échantillonnage du son de 44.1kHz. Supposons à présent que l'on désire lire la vidéo de l'image 100 à l'image

500, il suffira alors d'effectuer une requête identique pour le son et pour l'image dans l'espace timeref $[56448000, 282240000[$. Toutefois, si un timeref requête ne tombe pas exactement au début d'une information (exemple lorsque la requête n'est pas synchronisée au son ou à l'image), la valeur retenue correspondra alors à la valeur de timeref inférieure. Par exemple, si le temps d'exposition d'une image est inclus dans l'espace $[trA, trB[$ et que le timeref requête trR est compris entre $trA < trR < trB$, alors le timeref requête réel sera corrigé tel que $trR \leftarrow trA$.

La notion de segment temporel associée à des timeref de synchronisation, offre des perspectives intéressantes pour les algorithmes de traitement d'importants corpus audiovisuels, composés d'images de sons ou encore de documents textuels. Toutefois, cette représentation impose d'introduire un formalisme de requête proposant un accès simple aux informations.

3.1.3 Accès aux données par le bias d'URN (*Uniform Resource Names*)

Les paragraphes précédents soulignent qu'une des grandes problématiques associées à la conception d'un serveur de contenus (images, sons, textes,...) ou de descriptions (résultats de traitements) réside dans la synchronisation des données. Selon nous, toutes les données associées à une séquence (images, sons, résultats des traitements) doivent être accessibles en utilisant une logique commune et simple à utiliser. Une option intéressante consiste à baser l'accès aux informations sur un formalisme de type urn. Le principe des urn, couplé à l'utilisation de références temporelles, offre l'avantage non négligeable de décomposer précisément une séquence vidéo et d'accéder à une (ou plusieurs) partie(s) de son contenu indépendamment des fréquences d'échantillonnage du son et de l'image. Nous avons donc développé notre format d'urn dont la structure est présentée ci-dessous.

```
urn:x-tmxirisa:<urnfamily>:<urntype>:<idtype>:<id>
               (:<trstart>:<trstop>)
```

Actuellement ce format d'urn débute toujours par les informations `urn:x-tmxirisa` et se décompose en huit identifiants, séparés par le caractère ':'. Les six derniers identifiants sont variables et se décomposent comme suit:

1. `<urnfamily>`: identifie la famille des données représentées par l'urn. Cette famille peut prendre les valeurs `content` ou `description`. `content` indique que les données à traiter seront d'un type contenu (ex: des vidéos, des images ou encore du son...). `description` indique que les données à traiter seront des informations de description d'un contenu (ex: mouvement de caméra, détection de visages...).

2. `<urntype>`: correspond au type des données représentées par de l'urn d'identification. Pour une famille de type `content`, trois types de contenus sont disponibles: `video`, `audio` et `image`. Pour une

famille de type `description`, plusieurs types de contenus sont possibles (ex: `motion`, `facedetect`, `facetrack`, ...)

3. `<idtype>`: représente le type d'identificateur utilisé pour retrouver les informations sur le serveur. Nous avons développé deux types d'identifiants. `idfile` indique que l'identifiant suivant (`<id>`) permet d'accéder directement à une partie d'un fichier (ex: cibler le traitement sur journal télévisé de la mi-journée dans un fichier de 24 heures). `idcollection` indique que l'identifiant suivant (`<id>`) offre la possibilité de traiter une collection de documents [2] (exemple: traitement de tous les journaux télévisés répartis sur sept fichiers de 24 heures chacun).

4. `<id>`: représente l'identifiant du document ou de la collection de documents à traiter. Précisons qu'il est indispensable que cet identifiant soit unique (fichiers et collections).

5. `<trstart>` (optionnel): correspond à la référence temporelle indiquant le début du document à traiter. Cette variable n'est pas utilisée pour le traitement de collections.

6. `<trstop>` (optionnel): correspond à la référence temporelle indiquant le fin du document à traiter. Cette variable n'est pas utilisée pour le traitement de collections.

Ce formalisme d'accès offre la possibilité de traiter des séquences vidéos réparties sur plusieurs fichiers sans que l'utilisateur ne sache réellement à quel fichier il accède. L'accès aux informations est donc formalisé en termes de séquences (intervalles temporels) ou de collections (segments temporels) et non de fichiers. Par exemple, supposons un algorithme de traitement de l'actualité sur une période de plusieurs semaines de télévision. La définition d'une collection `journaux-du-020106-au-290106` via notre interface web (voir paragraphe 2) offre alors la possibilité de traiter l'ensemble des journaux en une seule urn (`urn:x-tmxirisa:content:video:idcollection:journaux-du-020106-au-290106`), et ce, quelles que soient les positions des séquences au sein des fichiers vidéos.

Précisons enfin que les urn sont décodées et vérifiées par l'application serveur (figure 2).

3.2 Accès aux images d'un flux MPEG-1 ou MPEG-2

Nous avons choisi de coder nos vidéos aux normes MPEG-1 et 2 qui offrent l'avantage d'être bien connues et décrites dans de nombreux documents. Plusieurs librairies (C/C++) proposent de décoder des flux MPEG, toutefois

[2] Partie d'un fichier ou fichier complet.

elles sont souvent loin de satisfaire aux exigences d'une plateforme vidéo telle que celle présentée dans cet article. Dans les paragraphes précédents nous avons posé un cadre théorique d'accès synchronisé aux données. Dans cette partie, nous proposons d'appliquer ce cadre aux spécificités du codage MPEG.

3.2.1 Relation "timeref / time stamp" .

A partir de la définition du timeref présentée dans le paragraphe 3.1.2 et la définition des références temporelles PTS et DTS de la norme MPEG [1] [2] [6], nous avons établi la relation suivante entre la référence temporelle (timeref) d'une image X (trX) et sa référence temporelle de décodage (DTS_X):

$$trX = DTS_X * timebase * valref - trVS \quad (2)$$

Avec $timebase$: référence temporelle de la vidéo (information contenue dans le flux MPEG), $trVS$: référence temporelle du début de la vidéo et $valref$=14112000.

Afin de simplifier la lecture aléatoire d'une vidéo et de compenser les imperfections temporelles générées par certains encodeurs hard, nous calculons au préalable toutes les références temporelles de toutes les images I de la vidéo MPEG, que nous stockons dans un fichier côté serveur. Précisons que cette opération est totalement transparente pour l'utilisateur.

3.2.2 Transfert et décodage des paquets vidéo.

Fonctionnement général du couple client/serveur.

Le fonctionnement général du couple client/serveur est représenté sous une forme simplifiée figure 2 (exemple d'accès aux images). Lorsqu'une connexion avec un serveur est établie à l'initiative d'un client, ce dernier peut alors effectuer une demande d'ouverture d'un contenu vidéo, sous la forme d'une requête urn. L'urn est alors vérifiée par le serveur, le contenu vidéo est ouvert et des informations telles que la taille des images ou encore la durée totale de la vidéo sont retournées au client. Le client peut alors demander une image de la séquence à traiter, le serveur vérifie alors que l'image requête se trouve bien dans l'urn et retourne les paquets nécessaires à son décodage. Le client décode l'image, la stocke dans un buffer (RGB ou YUV), la traite et boucle en demandant l'image suivante. Dès que la séquence est terminée, le contenu et les connexions sont fermés.

Décodage sur le client ou le serveur ?

Au sein d'une architecture telle que celle que nous proposons ici, le décodage d'un flux MPEG peut être effectué soit sur le serveur, soit sur la machine cliente. Notre solution logicielle a été conçue pour supporter ces deux

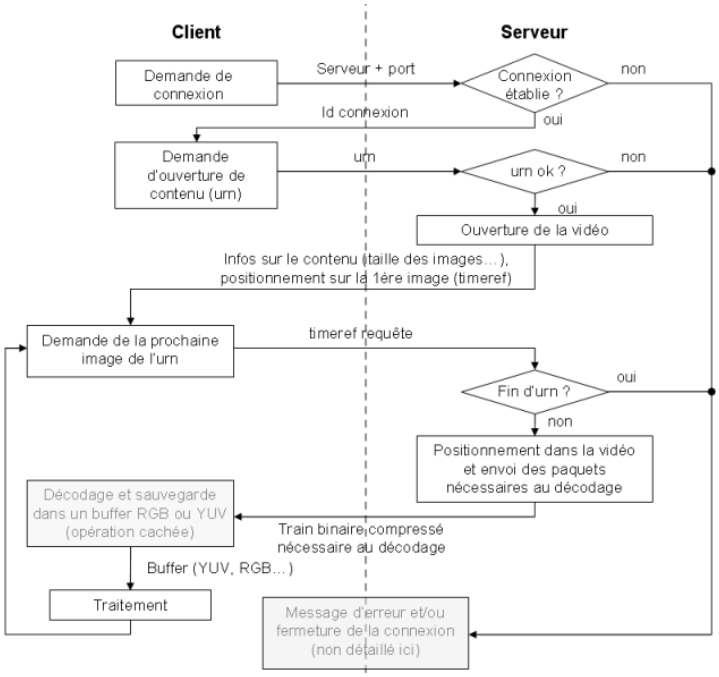

Figure 2: Schéma simplifié de fonctionnement général du couple client/serveur (accès aux images).

cas, toutefois, nous préconisons un décodage sur la machine cliente (figure 2). En effet, un décodage sur le serveur impliquerait d'une part une forte utilisation du processeur du serveur et d'autre part une charge inutile du réseau. *A contrario*, le décodage sur la machine cliente ne surcharge pas le processeur du serveur et n'encombre pas le réseau (voir expérimentations, paragraphe 4), mais nécessite l'installation de librairies de décodage. Nous avons donc développé une librairie cliente légère (quelques fichiers) et simple d'utilisation, qui intègre le décodage des flux vidéos. Précisons que ce décodage est réalisé de manière totalement transparente pour l'utilisateur qui ne manipule qu'un simple buffer image décompressé.

Choix d'une librairie de décodage.

Parmi l'ensemble des librairies de décodage MPEG libres de droits, notre choix s'est focalisé sur la librairie FFMPEG (licence LGPL). Cette librairie (utilisée par VLC) dispose de nombreux codecs et bénéficie d'une certaine pérennité de part sa forte utilisation. De plus, elle permet de décoder à la fois l'image et le son, et supporte l'ouverture de fichiers de longue durée (24 heures). Notons que certaines librairies (ex: libmpeg3) se révèlent parfois

inappropriées au décodage de vidéos de longue durée. Ces limitations sont généralement dues à des dépassements de capacités de certaines variables.

3.2.3 Identification de la première image d'une séquence.

L'identification de la première image d'un fichier MPEG-2 est un problème crucial, souvent rencontré lors du développement d'algorithmes d'analyse de flux vidéo. En effet, comme l'illustre la figure 3, les premières images d'un flux MPEG peuvent parfois être incomplètes. Ce cas, relativement rare, peut apparaître lors de l'enregistrement d'un flux brut provenant de la TNT ou encore en utilisant un encodeur MPEG de mauvaise qualité. Ainsi, la figure 3 présente un exemple de séquence où les trois premiers paquets sont insuffisant pour reconstituer les premières images et où la première image valide apparaît uniquement à partir du début de premier GOP, à savoir, la première image I. Afin de limiter les conséquences d'un tel cas (relativement rare), nous considérons que la première image fiable est représentée par la première image I d'un fichier. Toutefois, ne disposant pas des informations nécessaires pour certifier si une image est complète ou non, nous n'avons posé aucune restriction lors de la transmission des paquets. Ainsi, le serveur transmet au client toutes les images d'une séquence en précisant si ces dernières sont susceptibles d'être incomplètes, charge ensuite au client de les traiter ou non.

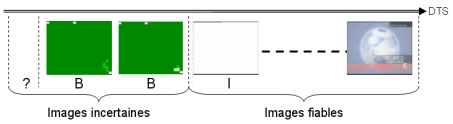

Figure 3: Exemple de flux MPEG où les trois premiers paquets de données ne représentent aucune image valide.

4 RÉSULTATS EXPÉRIMENTAUX.

4.1 Conditions expérimentales.

Les résultats présentés dans ce paragraphe ont été obtenus à partir d'un serveur unique (1 processeur Intel Xeon 3.4GHz 32 bits, 3 Go de RAM, 1 baie RAID Dell/EMC AX100 3To, connexion réseau 1Gb/s, Windows server 2003) et de n machines clientes toutes identiques (l processeur AMD Opteron 246 2.0GHz 64 bits, 2 Go de RAM, linux, ubuntu 64 bits).

L'algorithme de test utilisé réalisait un simple calcul d'histogramme sur les composantes RGB de chaque image. Notons qu'une conversion de l'espace colorimétrique YUV (directement issu du MPEG) en RGB était effectuée

lors du décodage de chaque image (conversion intégrée dans la librairie cliente, totalement transparente pour l'utilisateur).

La vidéo de test utilisée représente une heure consécutive de télévision extraite d'un fichier de 24 heures codé en MPEG-2 4Mb/s de résolution 720*576, soit un total de 90000 images représentant chacune environ 1.2Mo non compressés, soit environ 103 Go de données non compressées. Précisons qu'un fichier de 24 heures de télévision compressé en MPEG-2 (4 Mb/s), représente environ 38Go de données, soit environ 1.16Go par heure.

4.2 Observation des charges client/serveur (client unique).

Nous observons que le calcul d'histogrammes sur une vidéo d'une heure avec un client unique dure approximativement 580 secondes. Nous constatons que durant cette période, le processeur du serveur est utilisé en moyenne à seulement 1.5% de ses capacités, alors que le processeur du client est utilisé à presque 100% de ses capacités. Par ailleurs, l'encombrement réseau est d'en moyenne à 17.5Mb/s, soit un total sur 580 secondes d'environ 1.2Go de données compressées et correspondant approximativement à l'heure de vidéo traitée. Notons enfin que nous avons également observé un comportement très linéaire du système qui permet de traiter un fichier de 24 heures sur une machine cliente en environ 230 minutes (soit une moyenne d'environ 156 images traitées par secondes). Pour une telle configuration matérielle, le facteur limitant semble donc être concentré autour de l'occupation du processeur de la machine cliente. Nous avons donc commencé à étudier des algorithmes de répartition des calculs sur n machines clientes.

4.3 Vers une répartition des calculs (n clients).

Partant du constat que le facteur limitant les performances se trouve côté client, nous avons testé notre algorithme en décomposant la séquence d'une heure sur un ensemble de n machines clientes, chaque machine étant chargée de traiter $3600/n$ secondes consécutives de la vidéo.

Une synthèse des résultats est proposée figure 4. Dans cet exemple, nous constatons qu'un gain de temps réel est obtenu en utilisant jusqu'à environ 15 machines clientes (traitement 13 fois plus rapide qu'avec une simple machine), précisons tout de même que plus le traitement sera complexe, plus le nombre de machines nécessaires à l'optimisation des performances sera élevé. Nous constatons également qu'au delà de 15 machines clientes, les performances du système restent relativement stables. Cela indique d'une part que notre solution logicielle répartit équitablement les informations entre les différents clients et d'autre part qu'un facteur limite les performances globales.

En se basant sur ces résultats, nous constatons que le facteur limitant les performances de notre infrastructure est principalement concentré sur les

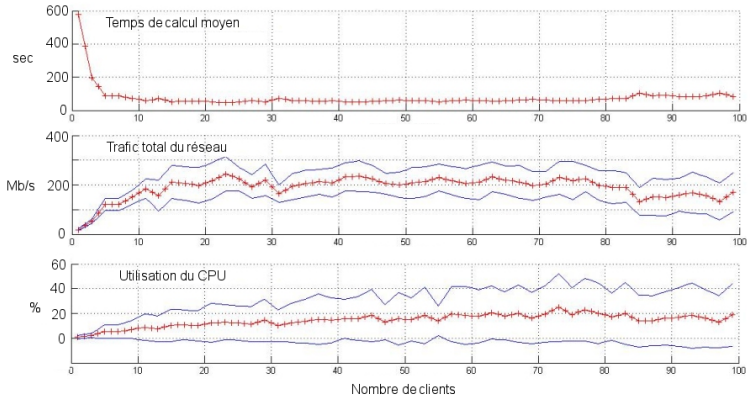

Figure 4: Test de charge du serveur pour une urn représentant 1 heure de vidéo répartie sur n clients identiques (Les courbes bleues représentent l'écart type de chaque signal.)

accès aux disques durs. Actuellement, nous utilisons une baie RAID (niveau 5) d'entrée de gamme (Dell/EMC AX100) que nous saturons de part l'aspect non séquentiel de la lecture des vidéos. Néanmoins, cette infrastructure offre des performances tout à fait satisfaisantes pour le traitement de vidéos d'une durée raisonnable.

5 CONCLUSION.

Dans cet article, nous avons abordé plusieurs aspects techniques liés aux problèmes de stockage et de diffusion pour le traitement de séquences vidéos. Les corpus vidéos nécessaires au développement d'algorithmes de traitements multimédias, et plus particulièrement dans le domaine de l'indexation de séquences télévisées, peuvent rapidement représenter plusieurs To d'informations, souvent protégées par des conventions respectant les droits d'auteurs. La duplication de ces données sur différentes machines apparaît donc inconcevable. Dans cet article, nous avons présenté notre solution logicielle composée d'un gestionnaire de vidéos et d'un couple client/serveur permettant d'accéder à des images ou des échantillons sonores précis d'une vidéo stockée sur un serveur. Notre objectif principal était de libérer les utilisateurs des contraintes techniques liées au décodage ou à la synchronisation des données en fournissant les informations sous forme de buffers non compressés. La conception de cette solution nous a alors conduit à analyser des problématiques telles que la synchronisation de l'accès aux données (image et son), l'identification de la première image de la séquence, le choix de librairies de décodage, ou encore l'optimisation de la charge du

serveur... Nous avons également réalisé des premiers tests de répartition des calculs sur plusieurs machines clientes qui nous ont permis d'évaluer la robustesse de notre serveur. Nous poursuivons nos travaux dans ce sens dans l'objectif de réduire au mieux les temps de calculs, souvent conséquents, afin de traiter rapidement des corpus vidéos représentant plusieurs semaines de télévision.

REFERENCES

[1] Michel Barlaud et Claude Labit. *Compression et codage des images et des vidéos.* IC2 Traitement du signal et de l'image, Hermes Sciences, 2002.

[2] Borko Furht. A survey of multimedia compression techniques and standards. part ii: Video compression. *Real-Time Imaging*, 1(5):319–337, november 1995.

[3] Nicolas Moënne-Loccoz. Oval: an object-based video access library to facilitate the developement of content-based video retrieval systems. Technical report, Viper group - Univertity of Geneva, 2004.

[4] Nicolas Moënne-Loccoz, Bruno Janvier, Stéphane Marchand-Maillet et Eric Bruno. An integrated framework for the management of video collection. In *Workshop on Multimodal Interaction and Related Machine Learning Algorithms (MLMI'04)*, Martigny, Switzerland, June 2004.

[5] Boris Rousseau, Wilfried Jouve et Laure Berti-Équille. Enriching multimedia content description for broadcast environments: From a unified metadata model to a new generation of authoring tool. In *IEEE International Symposium on Multimedia (ISM 2005)*, Irvine, California, december 2005.

[6] Thomas Sikora. Mpeg digital video-coding standards. *IEEE Signal Processing Magazine*, 14(5):82–100, september 1997.