US 20110083144A1

(19) United States

(12) Patent Application Publication

Bocharov et al.

(10) Pub. No.: US 2011/0083144 A1

(43) Pub. Date: Apr. 7, 2011

(54) **INTEGRATING CONTINUOUS AND SPARSE STREAMING DATA**

(76) Inventors: **John A. Bocharov**, Seattle, WA (US); **Geqiang (Sam) Zhang**, Redmond, WA (US); **Krishna Prakash Duggaraju**, Renton, WA (US); **Lin Liu**, Sammamish, WA (US); **Anirban Roy**, Kirkland, WA (US); **Jack E. Freelander**, Monroe, WA (US); **Vishal Sood**, Bothell, WA (US)

(21) Appl. No.: **12/612,685**

(22) Filed: **Nov. 5, 2009**

Related U.S. Application Data

(60) Provisional application No. 61/249,254, filed on Oct. 6, 2009.

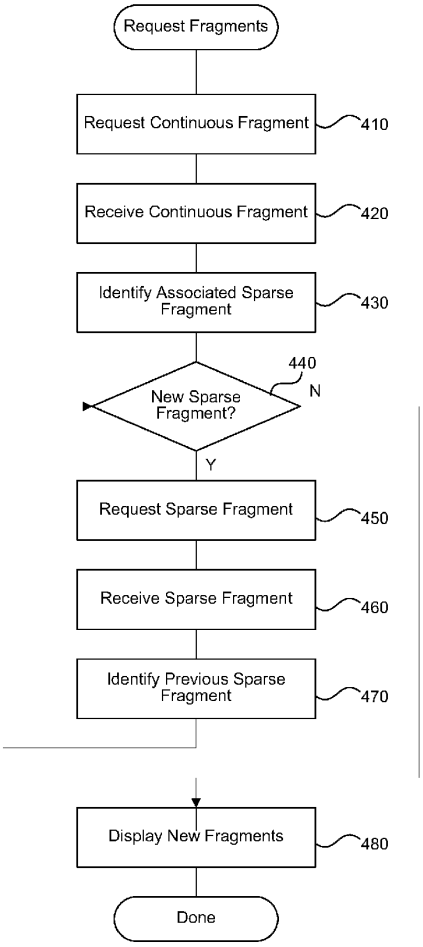
Publication Classification

(51) **Int. Cl.**
H04N 7/025 (2006.01)
H04N 7/173 (2006.01)

(52) **U.S. Cl.** **725/32**; 725/93; 725/92; 725/109

(57) **ABSTRACT**

A sparse streaming system provides a first-class means for sparse metadata to be added to streaming media presentations and to be delivered using an integrated data channel that is cacheable using readily available HTTP-based Internet caching infrastructure for increased scalability. The sparse streaming system stores a reference to a sparse track within a continuous track. If a continuous fragment arrives at the client that refers to a sparse fragment that the client has not yet retrieved, then the client requests the sparse fragment. In addition, each sparse fragment may include a backwards reference to the sparse fragment created immediately prior. The references in the continuous fragments make the client aware of new sparse track fragments, and the backwards references in the sparse track fragments ensure that the client has not missed any intervening sparse track fragments.



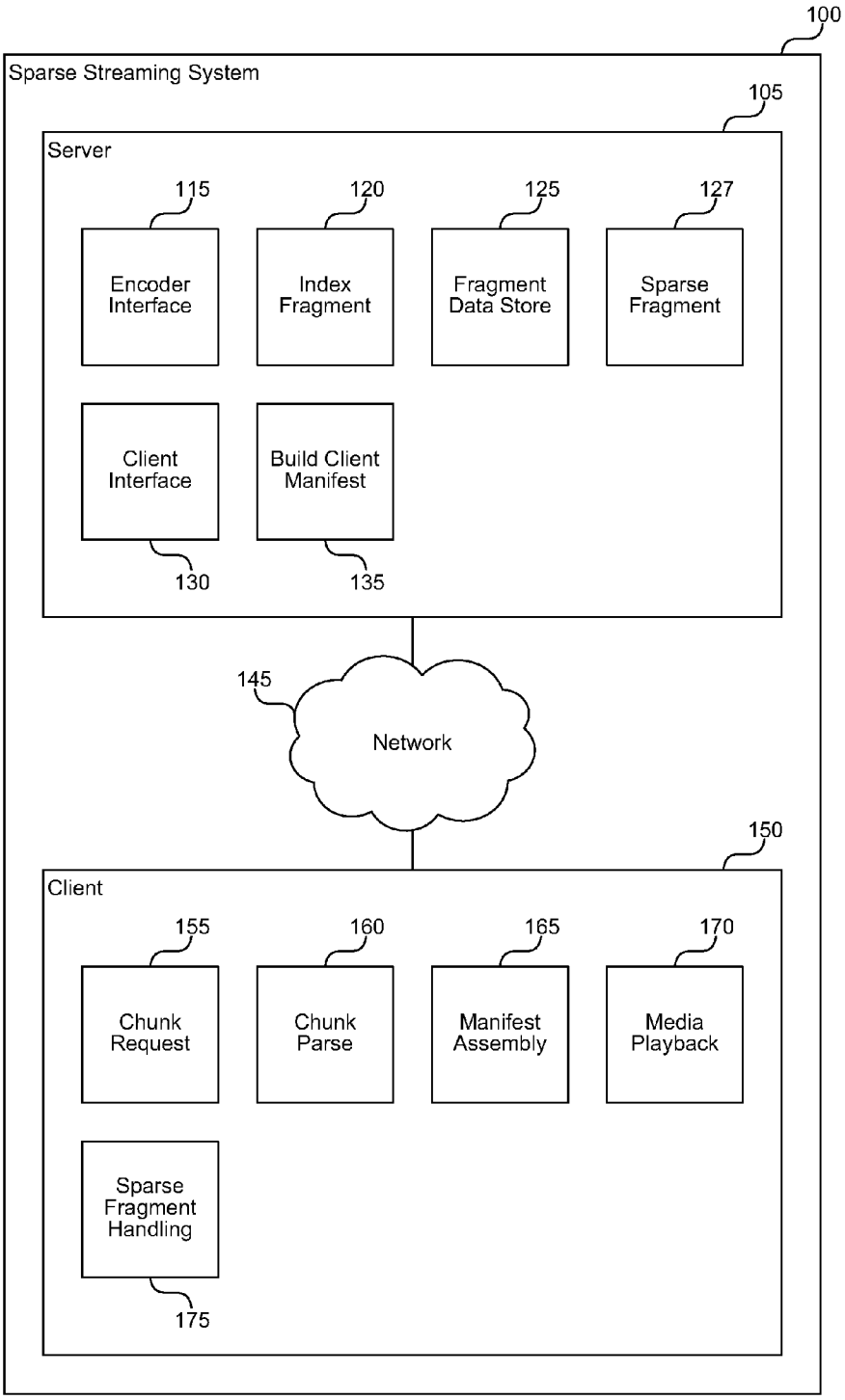

100
Sparse Streaming System
105
Server
115
Encoder Interface
120
Index Fragment
125
Fragment Data Store
127
Sparse Fragment
Client Interface
130
Build Client Manifest
135
145
Network
150
Client
155
Chunk Request
160
Chunk Parse
165
Manifest Assembly
170
Media Playback
Sparse Fragment Handling
175

FIG. 1

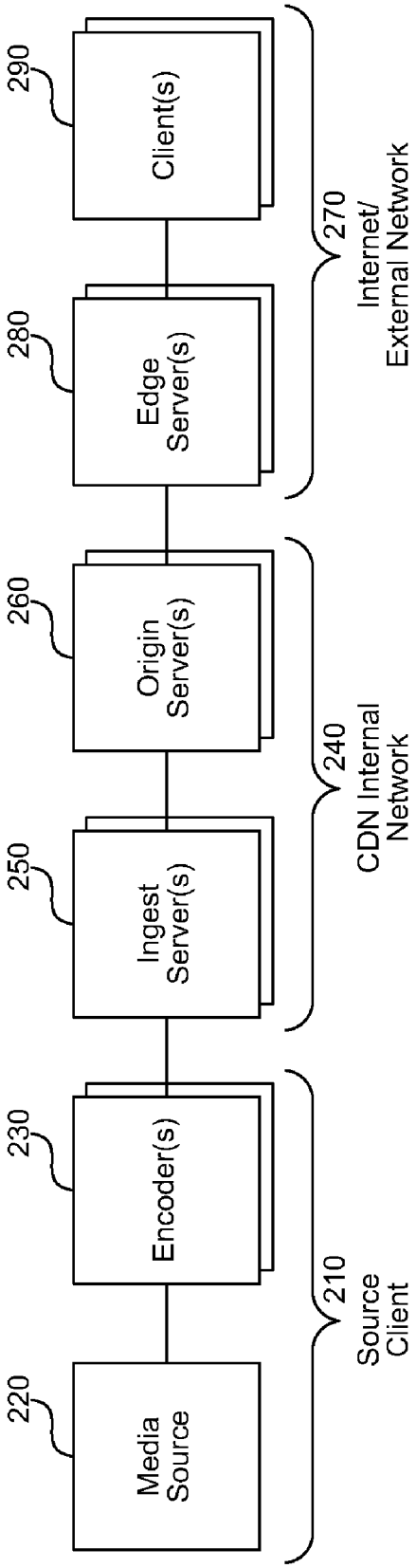

220
Media Source
230
Encoder(s)
250
Ingest Server(s)
260
Origin Server(s)
280
Edge Server(s)
290
Client(s)
210
Source Client
240
CDN Internal Network
270
Internet/ External Network

FIG. 2

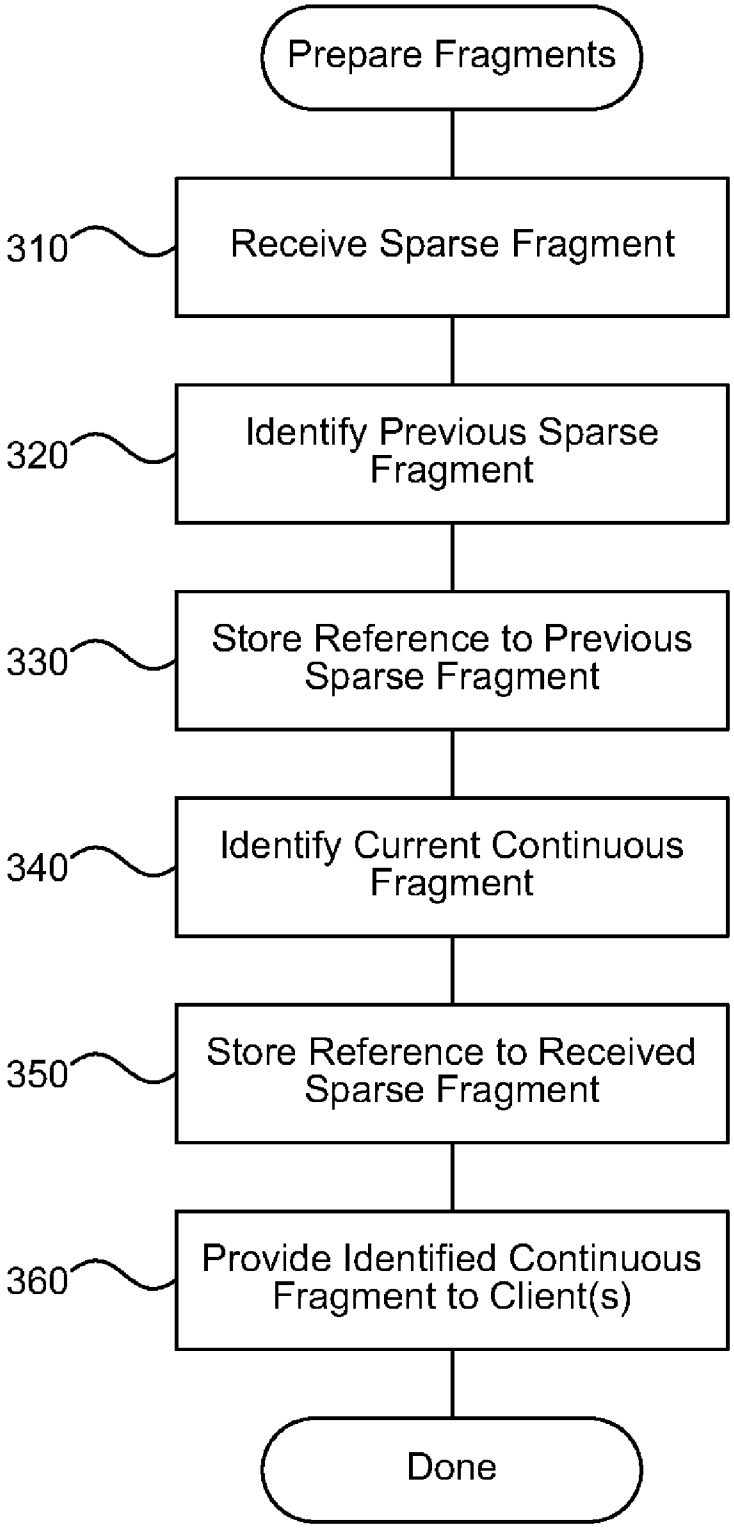

Prepare Fragments
310
Receive Sparse Fragment
320
Identify Previous Sparse Fragment
330
Store Reference to Previous Sparse Fragment
340
Identify Current Continuous Fragment
350
Store Reference to Received Sparse Fragment
360
Provide Identified Continuous Fragment to Client(s)
Done

FIG. 3

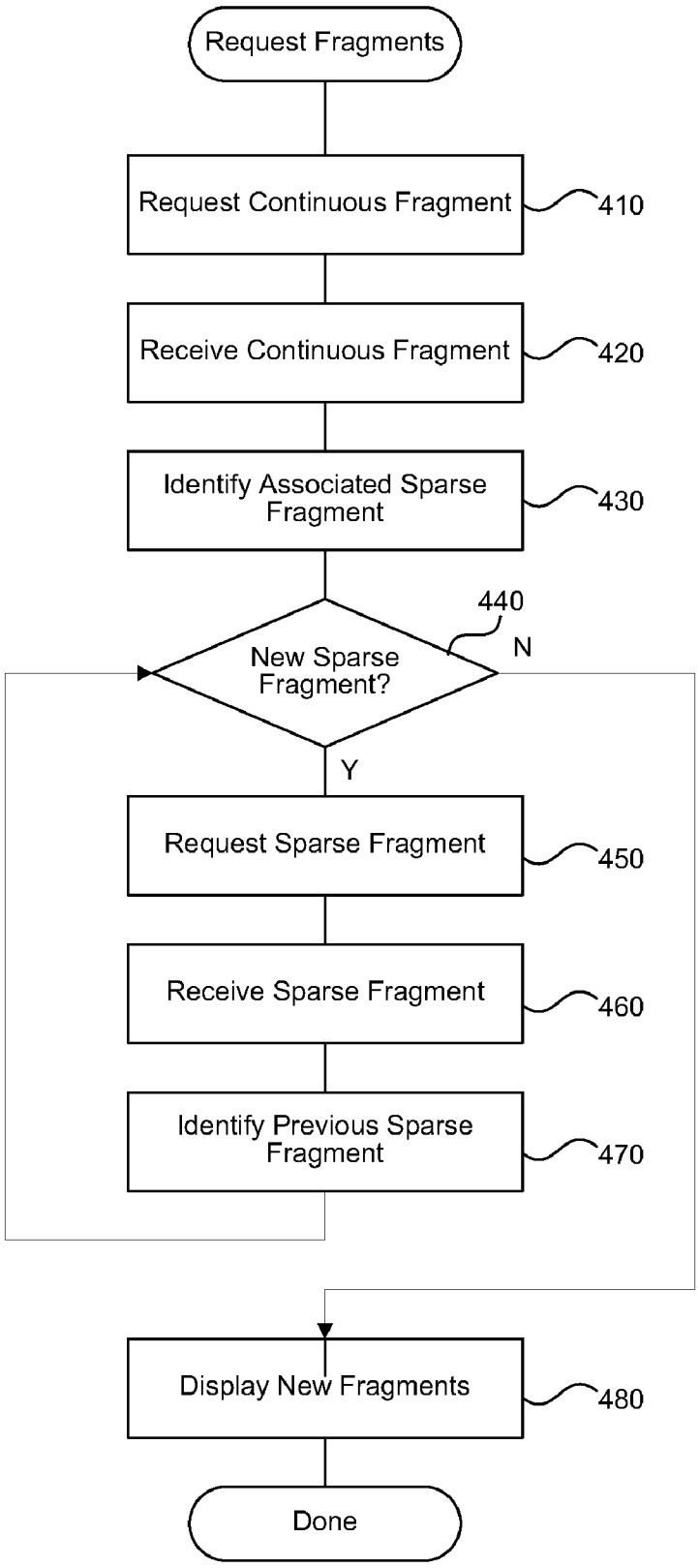

Request Fragments
Request Continuous Fragment
410
Receive Continuous Fragment
420
Identify Associated Sparse Fragment
430
440
New Sparse Fragment?
N
Y
Request Sparse Fragment
450
Receive Sparse Fragment
460
Identify Previous Sparse Fragment
470
Display New Fragments
480
Done

FIG. 4

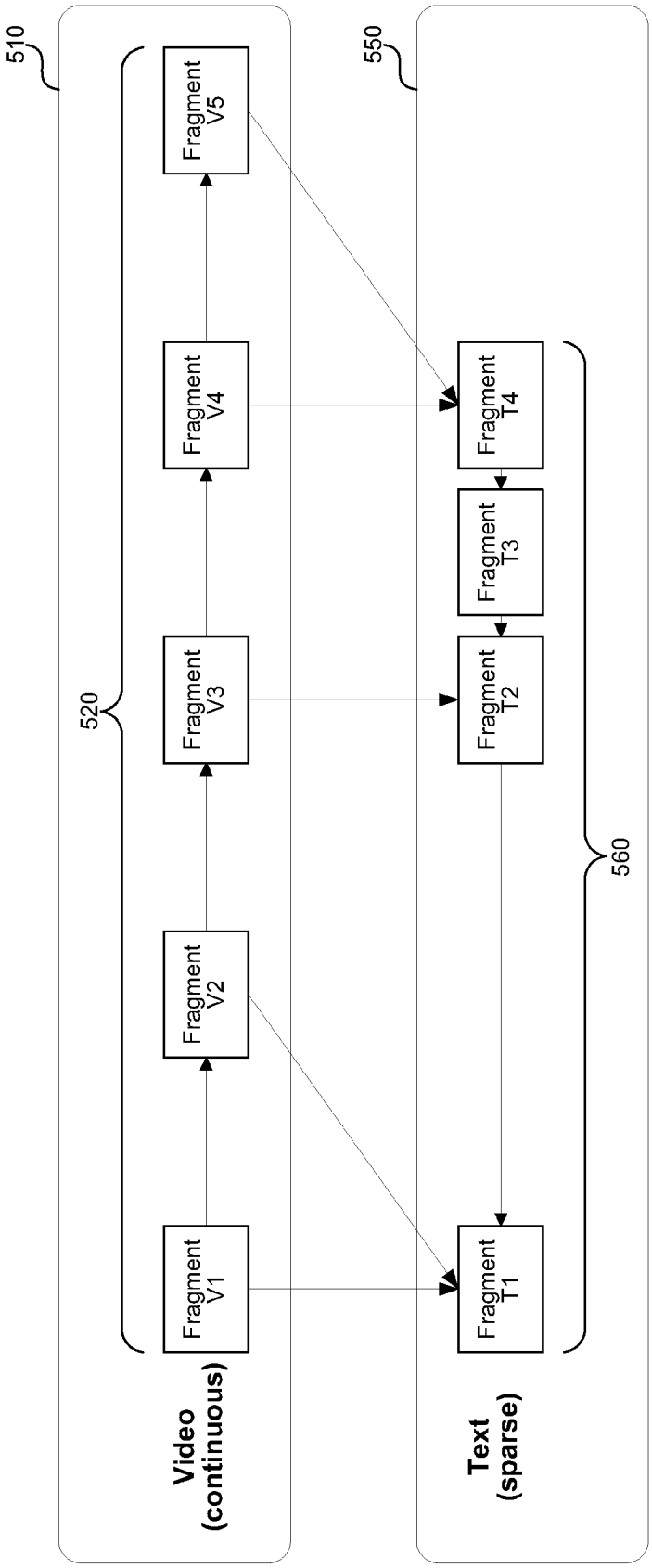

510
520
Video
(continuous)
Fragment
V1
Fragment
V2
Fragment
V3
Fragment
V4
Fragment
V5
550
560
Text
(sparse)
Fragment
T1
Fragment
T2
Fragment
T3
Fragment
T4

FIG. 5

INTEGRATING CONTINUOUS AND SPARSE STREAMING DATA

CROSS-REFERENCE TO RELATED APPLICATIONS

[0001] The present application claims priority to U.S. Provisional Patent Application No. 61/249,254 (Attorney Docket No. 327928.01) entitled "INTEGRATING CONTINUOUS AND SPARSE STREAMING DATA," and filed on Oct. 6, 2009, which is hereby incorporated by reference.

BACKGROUND

[0002] Streaming media is multimedia that is constantly received by, and normally presented to, an end-user (using a client) while it is being delivered by a streaming provider (using a server). One problem with existing media streaming architectures is the tight coupling between server and client. A stateful connection between client and server creates additional server overhead, because the server tracks the current state of each client. This also limits the scalability of the server. In addition, the client cannot quickly react to changing conditions, such as increased packet loss, reduced bandwidth, user requests for different content or to modify the existing content (e.g., speed up or rewind), and so forth, without first communicating with the server and waiting for the server to adapt and respond. Often, when a client reports a lower available bandwidth, the server does not adapt quickly enough, causing breaks in the media to be noticed by the user on the client as packets that exceed the available bandwidth are not received and new lower bit rate packets are not sent from the server in time. To avoid these problems, clients often buffer data, but buffering introduces latency, which for live events may be unacceptable.

[0003] In addition, the Internet contains many types of downloadable media content items, including audio, video, documents, and so forth. These content items are often very large, such as video in the hundreds of megabytes. Users often retrieve documents over the Internet using Hypertext Transfer Protocol (HTTP) through a web browser. The Internet has built up a large infrastructure of routers and proxies that are effective at caching data for HTTP. Servers can provide cached data to clients with less delay and by using fewer resources than re-requesting the content from the original source. For example, a user in New York may download a content item served from a host in Japan, and receive the content item through a router in California. If a user in New Jersey requests the same file, the router in California may be able to provide the content item without again requesting the data from the host in Japan. This reduces the network traffic over possibly strained routes, and allows the user in New Jersey to receive the content item with less latency.

[0004] Unfortunately, live media often cannot be cached using existing protocols, and each client requests the media from the same server or set of servers. In addition, when streaming media can be cached, it is often cached by specialized cache hardware, not existing and readily available HTTP-based Internet caching infrastructure. The lack of caching limits the number of parallel viewers and requests that the servers can handle, and limits the attendance of a live event. The world is increasingly using the Internet to consume up to the minute live information, such as the record number of users that watched live events such as the opening of the 2008 Olympics via the Internet. The limitations of current technology are slowing adoption of the Internet as a medium for consuming this type of media content.

[0005] Many video solutions on the Internet involve the ability to deliver text and data that is synchronized to the video's timeline. For example, the data may include closed captions, captions, advertising insertion, and play-by-play captioning. Existing solutions to this problem have used provisions in media formats that allow custom data to be embedded in the headers, and workflows that use a different mechanism for delivery of the data out-of-band. These solutions do not work well for pull content, where a client regularly requests data from one or more servers. With pull content, the client has to be informed of what content to request, which can be particularly difficult for sparse or sporadic content that is only occasionally included in a stream. Unlike video or audio data, which the client can simply request on a routine schedule (e.g., every two seconds), sparse data such as an advertisement may be available irregularly and at much broader intervals (e.g., ever 15 minutes).

SUMMARY

[0006] A sparse streaming system is described herein that provides a first-class means for sparse metadata to be added to streaming media presentations and to be delivered using an integrated data channel that is cacheable using readily available HTTP-based Internet caching infrastructure for increased scalability. The sparse streaming system typically includes at least one continuous streaming track and at least one sparse streaming track. The sparse streaming system stores a reference to the sparse track within the continuous track. If a continuous fragment arrives at the client that refers to a sparse fragment that the client has not yet retrieved, then the client requests the sparse fragment. In addition, each sparse fragment may include a backwards reference to the sparse fragment created immediately prior. The client can use the reference to follow a chain of backwards references until the client reaches a sparse fragment that the client has already retrieved. In this way, the references in the continuous fragments make the client aware of new sparse track fragments, and the backwards references in the sparse track fragments ensure that the client has not missed any intervening sparse track fragments. Thus, the sparse streaming system provides a way for the client to receive all of the fragments of a sparse track, even when sparse track fragments are not created with any regularity.

[0007] This Summary is provided to introduce a selection of concepts in a simplified form that are further described below in the Detailed Description. This Summary is not intended to identify key features or essential features of the claimed subject matter, nor is it intended to be used to limit the scope of the claimed subject matter.

BRIEF DESCRIPTION OF THE DRAWINGS

[0008] FIG. **1** is a block diagram that illustrates components of the sparse streaming system, in one embodiment.

[0009] FIG. **2** is a block diagram that illustrates an operating environment of the sparse streaming system using Microsoft Windows and Microsoft Internet Information Server (IIS), in one embodiment.

[0010] FIG. **3** is a flow diagram that illustrates processing of the sparse fragment component of a server, in one embodiment.

[0011] FIG. **4** is a flow diagram that illustrates processing of the sparse fragment handling component of a client, in one embodiment.

[0012] FIG. **5** is a diagram that illustrates one or more data structures used by the sparse streaming system to associate continuous and sparse fragments, in one embodiment.

DETAILED DESCRIPTION

[0013] A sparse streaming system is described herein that provides a first-class means for sparse metadata to be added to streaming media presentations and to be delivered using an integrated data channel that is cacheable using readily available HTTP-based Internet caching infrastructure for increased scalability and viewing by many clients. The system described is appropriate for live events or on-demand events that have already concluded. The sparse streaming system typically includes at least one continuous streaming track and at least one sparse streaming track. For example, the continuous streaming track may include audio, video, or other data that a client requests fragments (or chunks) of from a server with a regular cadence (e.g., every two seconds). The sparse track may include sporadic data, such as closed captions, commentary, or advertisements that the client requests irregularly.

[0014] The sparse streaming system stores information about the sparse track within the continuous track. For example, the system may include a reference within each fragment of the continuous streaming track to the last known fragment of the sparse streaming track. The client can use the reference to determine if the client has requested the latest sparse streaming fragment. If a continuous fragment arrives at the client that refers to a sparse fragment that the client has not yet retrieved, then the client requests the sparse fragment. In addition, each sparse fragment may include a backwards reference to the sparse fragment created immediately prior. The client can use the reference to follow a chain of backwards references until the client reaches a sparse fragment that the client has already retrieved. In this way, the references in the continuous fragments make the client aware of new sparse track fragments, and the backwards references in the sparse track fragments ensure that the client has not missed any intervening sparse track fragments. Thus, the sparse streaming system provides a way for the client to receive all of the fragments of a sparse track, even when sparse track fragments are not created with any regularity. In addition, because each fragment (whether of the continuous data stream or the sparse data stream) is available at a well known Uniform Resource Locator (URL) or other identifier, each fragment is cacheable by existing Internet caching infrastructure to reduce the burden on one or more servers and thereby allow more clients to view the streaming presentation.

[0015] In some embodiments, the sparse streaming system uses a particular data transmission format between the server and client. The client requests fragments of media from a server that include a portion of the media. For example, for a 10-minute file, the client may request 2-second or smaller fragments. Note that unlike typical streaming where the server pushes data to the client, in this case the client pulls media fragments from the server. In the case of a live stream, the server may be creating the media on the fly and producing fragments to respond to client requests. Thus, the client may only be several fragments behind the server in terms of how fast the server creates fragments and how fast the client requests fragments.

[0016] Each fragment contains metadata and media content. The metadata may describe useful information about the media content, such as the bit rate at which the media content was encoded, where the media content fits into a larger media element (e.g., this fragment represents offset 1:10 in a 10 minute video clip), the codec used to encode the media content, and so forth. The client uses this information to place the fragment into a storyboard of the larger media element and to properly decode and playback the media content.

[0017] FIG. **1** is a block diagram that illustrates components of the sparse streaming system, in one embodiment. The sparse streaming system **100** includes at least one server **105** and at least one client **150** connected by a network **145**, such as the Internet. The server **105** includes an encoder interface component **115**, an index fragment component **120**, a fragment data store **125**, a sparse fragment component **127**, a client interface component **130**, and a build client manifest component **135**. Each of these components is described in further detail herein.

[0018] The encoder interface component **115** provides an interface between the system **100** and one or more encoders that provide encoded media data. The encoders may push data to the system **100** using common network protocols. For example, the encoders may use an HTTP POST request to provide encoded media data to the system **100**. The encoders may each use a distinguished URL that specifies the encoder that is the source of the encoded media data, which the server may match to the information received by the register event component **110** when the media event was registered.

[0019] The encoder interface component **115** may specify a particular format for received encoded media data, such as an MP4 or other media container (e.g., MKV). The MP4 container format allows multiple types of data to be associated in a single file. The individual data that makes up an MP4 container is called a box, and each box typically has a label that identifies the type of data stored in the box. Encoders may place metadata information in the boxes such as the type of encoding used to encode the encoded media data, as well as the encoded media data itself.

[0020] The system **100** may configure encoders to lower overall latency produced by the system. For example, the system **100** may configure encoders not to include B-frames in streams so that the decoder can more quickly decode frames without waiting for subsequent frames to which B-frames would refer. In addition, the system **100** may instruct encoders to produce fragments that include less than a whole GOP, so that the encoder can push fragments to the server **105** more quickly.

[0021] The index fragment component **120** creates and maintains an index table of fragments received from various encoders. Because the system **100** is receiving media fragments on an on-going basis during an event from potentially many encoders, the system **100** uses the index table to keep track of what media fragments have been received and from which encoders (or in which formats). Each encoder may use a common method for identifying media fragments (e.g., a time stamp using a synchronized clock) so that the index fragment component **120** can correlate fragments from different encoders that represent the same period in a live event. In this way, the system **100** can detect when media fragments are missing and can provide clients with manifest information about available media fragments.

[0022] The fragment data store **125** stores received media fragments and the created index table of fragments to provide

to clients in response to received client requests. The fragment data store may include a database, disk drive, or other form of data storage (e.g., a Storage Area Network (SAN) or a cloud-based storage service).

[0023] The sparse fragment component **127** receives sparse fragments from one or more sources and prepares the fragments for retrieval by clients. The sparse fragment component **127** may receive sparse data from a variety of sources and aggregate the data or make each set of data available as a separate stream. For example, announcers viewing a live sporting event may push a button or send data in some other manner to a server that indicates when a player has scored a goal or another significant game event has occurred. The announcers may also provide commentary associated with the sporting event presentation. As another example, the announcers or another party may mark points within the sporting event that are appropriate breaks for advertising (e.g., commercials). The sparse fragment component **127** collects these events and makes them available in association with the continuous video and/or audio presentation. In addition, the sparse fragment component **127** may prepare the fragments for retrieval by associating each sparse fragment with at least one continuous fragment and by associating sparse fragments with one another as described further herein.

[0024] The client interface component **130** receives client requests for media fragments and provides manifest data and media fragments to clients. When a client initially connects to the system **100**, the client may send a request for a client manifest. The client interface component **130** invokes the build client manifest component **135** to create a manifest that includes information about the encodings available from the system **100**, and fragments stored by the system **100** up to the current time based on the index table. The client can use this information either to begin requesting ongoing live fragments, or to skip backwards in time to earlier portions of a presentation. The client can use this technique, for example, if the client joins a live event that is already in progress and wants to catch up with the previous portions of the event.

[0025] The build client manifest component **135** builds a manifest to satisfy a client request that includes information about each of the encodings available from the system **100** and fragments stored by the system **100** up to the current time. The build client manifest component **135** also provides a manifest to include with each media fragment that provides information to the client about the current media fragment as well as potentially subsequent fragments. By combining the initially received manifest with subsequent manifests provided with each media fragment, the client can build an up to date manifest that includes complete information about the media event from the start up until the current time. When the media event completes, the client has a complete storyboard of the media event that the client can use for on-demand viewing of the media event.

[0026] In some embodiments, the client interface component **130** responds to client requests for available fragments without waiting for subsequent fragments from which to include information with the current fragment. The client can request subsequent fragments by referencing the current fragment. For example, if the client last requested a fragment at time 1000, and wants the subsequent fragment, the client may send a request to get the fragment following the fragment at time 1000. In this way, the server can send fragments without introducing additional latency by waiting for subsequent fragments before sending a fragment.

[0027] The client **150** includes a chunk request component **155**, a chunk parsing component **160**, a manifest assembly component **165**, a media playback component **170**, and a sparse fragment handling component **175**. Each of these components is described in further detail herein.

[0028] The chunk request component **155** makes requests from the client for individual media chunks from the server. As shown in FIG. **2**, the client's request may pass first to an edge server (e.g., an Internet cache), then to an origin server, and then to an ingest server. At each stage, if the requested data is found, then the request does not go to the next level. For example, if the edge server has the requested data, then the client receives the data from the edge server and the origin server does not receive the request. Each chunk may have a URL that individually identifies the chunk. Internet cache servers are good at caching server responses to specific URL requests (e.g., HTTP GET). Thus, when the first client calls through to the server **105** to get a chunk, the edge servers cache that chunk and subsequent clients that request the same chunk may receive the chunk from the edge server (based on the cache lifetime and server time to live (TTL) settings). The chunk request component **155** receives the chunk and passes it to the chunk parsing component **160** for interpretation. The chunk request component 1550 may request chunks by referencing a previous chunk (e.g., give me the chunk after chunk N).

[0029] The chunk parsing component **160** interprets the format of a media chunk received by the chunk request component **155** and separates the chunk into its component parts. Typically, the chunk includes a header portion containing metadata, and a data portion containing media content. The chunk parsing component provides the metadata to the manifest assembly component **165** and the media content to the media playback component **170**. The media content may include mixed media types, such as video and audio data related to a presentation.

[0030] The manifest assembly component **165** builds a manifest that describes the media element to which received media content belongs. Large media files that clients download as a whole (i.e., not streamed) often include a manifest describing the whole file, the codecs and bit rates used to encode various portions of the file, markers about meaningful positions with the file, and so forth. During streaming, particularly live content, a server **105** cannot provide a complete manifest because the event is still ongoing. Thus, the server **105** provides as much of the manifest as it can through the metadata in the media chunks. The server **105** may also provide an application-programming interface (API), such as a predefined URL, for the client to request the manifest up to the current point in the media stream. This can be useful when the client **150** joins a live, streamed event after the event is already in progress. The manifest allows the client **150** to request previously streamed portions of the media element (e.g., by rewinding), and the client **150** continues to receive new portions of the manifest through the metadata of the streamed media chunks.

[0031] The manifest assembly component **165** builds a manifest similar to that available for a complete media file. Thus, as the event proceeds if the user wants to skip backwards in the media (e.g., rewind or jump to a particular position), then skip forward again, the user can do so and the client **150** uses the assembled manifest to find the appropriate

chunk or chunks to playback to the user. When the user pauses, the system **100** may continue to receive media chunks (or only the metadata portion of chunks based on a distinguished request URL), so that the manifest assembly component **165** can continue to build the manifest and be ready for any user requests (e.g., skip to the current live position or play from the pause point) after the user is done pausing. The client-side assembled manifest allows the client **150** to play the media event back as on-demand content as soon as the event is over, and to skip around within the media event as it is going on.

[0032] The media playback component **170** plays back received media content using the client hardware. The media playback component **170** may invoke one or more codecs to interpret the container within which the media content is transported and to decompress or otherwise decode the media content from a compressed format to a raw format (e.g., YV12, RGBA, or PCM audio samples) ready for playback. The media playback component **170** may then provide the raw format media content to an operating system API (e.g., Microsoft DirectX) for playback on local computer system sound and video hardware, such as a display and speakers. The server **105** can influence latency on the client **150** by the encoding used. For example, encodings that include B-frames may cause the client to buffer received data longer before playing it back. By encoding video data without B-frames, the server can cause the client to playback video with less latency.

[0033] The sparse fragment handling component **175** identifies one or more sparse tracks associated with a particular media presentation and retrieves sparse fragments of the sparse track. Each continuous fragment may include a reference to the most recently created sparse fragment. For example, the sparse fragments may be identified by a time stamp and each continuous fragment may contain the timestamp of the most recent sparse fragment. If the client **150** receives a continuous fragment that references a sparse fragment that the client **150** has not previously retrieved, then the client **150** requests the sparse fragment from the server **105**. The sparse fragment may reference other previous sparse fragments that the client **150** has not received, and the client **150** may request those fragments as well, as described further herein.

[0034] The computing device on which the sparse streaming system is implemented may include a central processing unit, memory, input devices (e.g., keyboard and pointing devices), output devices (e.g., display devices), and storage devices (e.g., disk drives or other non-volatile storage media). The memory and storage devices are computer-readable storage media that may be encoded with computer-executable instructions (e.g., software) that implement or enable the system. In addition, the data structures and message structures may be stored or transmitted via a data transmission medium, such as a signal on a communication link. Various communication links may be used, such as the Internet, a local area network, a wide area network, a point-to-point dial-up connection, a cell phone network, and so on.

[0035] Embodiments of the system may be implemented in various operating environments that include personal computers, server computers, handheld or laptop devices, multiprocessor systems, microprocessor-based systems, programmable consumer electronics, digital cameras, network PCs, minicomputers, mainframe computers, distributed computing environments that include any of the above systems or devices, and so on. The computer systems may be cell phones, personal digital assistants, smart phones, personal computers, programmable consumer electronics, digital cameras, and so on.

[0036] The system may be described in the general context of computer-executable instructions, such as program modules, executed by one or more computers or other devices. Generally, program modules include routines, programs, objects, components, data structures, and so on that perform particular tasks or implement particular abstract data types. Typically, the functionality of the program modules may be combined or distributed as desired in various embodiments.

[0037] FIG. **2** is a block diagram that illustrates an operating environment of the sparse streaming system using Microsoft Windows and Microsoft Internet Information Server (IIS), in one embodiment. The environment typically includes a source client **210**, a content delivery network **240**, and an external network **270**. The source client is the source of the media or live event. The source client includes a media source **220** and one or more encoders **230**. The media source **220** may include cameras each providing multiple camera angles, microphones capture audio, slide presentations, text (such as from a closed captioning service), images, and other types of media. The encoders **230** encode the data from the media source **220** in one or more encoding formats in parallel. For example, the encoders **230** may produce encoded media in a variety of bit rates.

[0038] The content delivery network **240**, where the sparse streaming system operates, includes one or more ingest servers **250** and one or more origin servers **260**. The ingest servers **250** receive encoded media in each of the encoding formats from the encoders **230** and create a manifest describing the encoded media. The ingest servers **250** may create and store the media fragments described herein or may create the fragments on the fly as they are requested. The ingest servers **250** can receive pushed data, such as via an HTTP POST, from the encoders **230**, or via pull by requesting data from the encoders **230**. The encoders **230** and ingest servers **250** may be connected in a variety of redundant configurations. For example, each encoder may send encoded media data to each of the ingest servers **250**, or only to one ingest server until a failure occurs. The origin servers **260** are the servers that respond to client requests for media fragments. The origin servers **260** may also be configured in a variety of redundant configurations.

[0039] In some embodiments, the ingest servers **250** comprise one or more servers dedicated to ingesting encoder media streams. An administrator or content author may create a publishing point that defines a URL at which clients of the ingest servers **250** can find a particular media element (e.g., a live event). For example, using IIS, the administrator may publish a URL "http://ingserver/pubpoint.isml." The publishing point is used by the encoders **230** to provide new media data to the ingest servers **250** and by the origin servers **260** to request media data from the ingest servers **250**. Each encoder may use a distinguished URL to connect to the ingest servers **250** so that the ingest servers **250** can detect different encodings of the same data. For example, based on the URL in the previous example, an encoder may send an HTTP POST to provide media data to the ingest server using the URL "http://ingserver/pubpoint.isml/Streams(stream1)." The ingest servers **250** store the received data for later retrieval by clients of the ingest servers **250** (e.g., the origin servers **260**). The POST may contain various types of media formats, such as an MP4

container. An MP4 container contains various types of information, called boxes, that are typically labeled with a four-letter code, such as "ftyp" to describe the type of encoding used and "moov" to contain audiovisual data. Whether using MP4 or other container formats, the encoder may add additional boxes or information to the stream, such as a "ManifestBox" that contains a manifest describing the media element.

[0040] When the ingest servers **250** receive a request for data, the ingest servers **250** provide the data stored earlier. The ingest servers **250** may support several types of requests, including a request for encoder stream manifest that identifies the available encoder streams and a request for data from a specific stream (including portions of the stream data). The type of request may be identified by the URL of the request. For example, when the ingest servers **250** receive the URL "http://ingserver/pubpoint.isml/StreamManifest," the ingest servers **250** return an encoder manifest that contains identifiers for each available encoder. When the ingest servers **250** receives the URL "http://ingserver/pubpoint.isml/Streams (stream1)," the ingest servers **250** send a corresponding media stream for the encoder associated with the identifier "Encoder1" in response. The response may include the MP4 data, such as the cached "ftyp," "ManifestBox," and "moov" boxes described above followed by media fragments in a FIFO buffer. The ingest servers **250** may also receive partial data requests (e.g., during failover scenarios) of the form "http://ingserver/pubpoint.isml/Streams(stream1)/StartTime (12345678)," that cause the ingest servers **250** to skip sending "ftyp," "ManifestBox," and "moov" boxes and to try to start from the media fragment that is closest to the specified timestamp.

[0041] The origin servers **260** receive requests for media streams from media clients and retrieve requested media streams from one or more ingest servers **250**. Like the ingest servers **250**, an administrator or content author registers a publishing point on the origin server, and then associates the ingest servers **250** and/or encoder URLs with the publishing point. The origin servers **260** may first request (e.g., using an HTTP GET request) a manifest from the ingest servers **250** that describes the available streams. The origin server then submits separate requests for each encoder stream to the ingest server, and the ingest server responds with the requested media stream as received from the encoder.

[0042] The origin servers **260** may separately receive manifest information about the media stream and media fragments that represent parts of a larger media element being provided by the media stream. The origin servers **260** build an index of each fragment received from each stream based on a timestamp or other identifier provided by each encoder that allows the origin servers **260** to correlate data from each encoder. The origin servers **260** may build their own MP4 container or other storage format from the received data from which to respond to media client requests. By building a file of a known format from a live event, the origin server may be able to provide a unified download of a media file quickly after the event.

[0043] When the origin servers **260** receive a media client request, the origin servers **260** generate a client manifest by appending the index that the server has built to the static stream information received from the encoder manifest(s). If there are multiple streams, then the origin servers **260** merge the stream manifests into a comprehensive client manifest. This allows the client to be selective in which encoding type the client requests without obtaining further information from the origin servers **260**. The server provides the manifest to the client using a standard response type that can be cached by existing Internet infrastructure, such as an HTTP response. Because the manifest data may change over time, the server may set a short cache timeout value (e.g., time to live (TTL)) on the manifest response.

[0044] The external network **270** includes edge servers **280** and other Internet (or other network) infrastructure and clients **290**. When a client makes a request for a media fragment, the client addresses the request to the origin servers **260**. Because of the design of network caching, if one of the edge servers **280** contains the data, then that edge server may respond to the client without passing along the request. However, if the data is not available at the edge server, then the edge server forwards the request to one of the origin servers **260**. Likewise, if one of the origin servers **260** receives a request for data that is not available, the origin server may request the data from one of the ingest servers **250**.

[0045] FIG. **3** is a flow diagram that illustrates processing of the sparse fragment component of a server, in one embodiment. The component is invoked to process sparse track data made available to clients from one or more server. Beginning in block **310**, the component receives a current sparse track fragment. For example, the component may receive the fragment from an encoder or other data source, such as a client from which a sports announcer connects to the server. Continuing in block **320**, the component identifies a previous sparse fragment from the same track. For example, the data source may have sporadically provided sparse fragments and the component identifies the fragment that preceded the current sparse fragment. Continuing in block **330**, the component stores a reference to the identified previous sparse track fragment in the received current sparse track fragment. The references in each sparse fragment form a reverse list that a client can use to identify previous sparse fragments from a current sparse fragment. The reference may include a timestamp, URL, or other identifier for locating the fragment to which the reference refers.

[0046] Continuing in block **340**, the component identifies a current continuous track fragment associated with the sparse track. For example, the component may identify a video stream track associated with a sparse text data stream. Continuing in block **350**, the component stores a reference to the received current sparse track fragment in the identified current continuous track fragment. Each continuous track fragment points to the most recently received sparse track fragment. In this way, receiving any continuous track fragment is sufficient to allow the client to follow the reverse list of sparse track fragments. Continuing in block **360**, the component provides the identified current continuous track fragment including the stored reference to the received current sparse track fragment to at least one client. For example, the client may request continuous track fragments using periodic HTTP GET requests to a server. After block **360**, these steps conclude.

[0047] FIG. **4** is a flow diagram that illustrates processing of the sparse fragment handling component of a client, in one embodiment. The component is invoked as a client processes a streaming media presentation to request fragments from a server. Beginning in block **410**, the component requests a continuous track fragment associated with the streaming media presentation. For example, the client may regularly request continuous track fragments according to a specified

time interval, quality of service (QoS) data, or other parameters. Continuing in block **420**, the component receives the requested continuous track fragment. Each fragment may be provided at a given URL for which the server provides a response that can be cached by Internet HTTP cache servers.

[0048] Continuing in block **430**, the component identifies a reference to a sparse track fragment associated with the continuous track fragment. For example, an HTTP "Pragma" header associated with the continuous track fragment response may reference any sparse data tracks associated with the continuous data track. Continuing in decision block **440**, if the identified sparse track fragment has previously been received by the client, then the component jumps to block **480**, else the component continues at block **450**. For example, the client may receive an identifier for each track, such as a track name and timestamp, that allows the client to compare sparse track fragments to identify previously received fragments. Continuing in block **450**, the component requests the referenced sparse track fragment. For example, the component may issue an HTTP request specifying a URL associated with the sparse track fragment.

[0049] Continuing in block **460**, the component receives the requested sparse track fragment. For example, a server or cache may provide the fragment via an HTTP response. Continuing in block **470**, the component identifies a reference to a previous sparse track fragment associated with the received sparse track fragment. For example, the HTTP response associated with the fragment may include an HTTP "Pragma" header containing a reference to the immediately previous sparse track fragment. After block **470**, the component loops to block **440** to determine whether the client has previously received the referenced sparse track fragment.

[0050] After the client has retrieved the continuous fragment and each new sparse track fragment, the component continues in block **480**. In block **480**, the component displays any newly received sparse and continuous track fragments. For example, the client may display a video track fragment and superimpose associated text caption data from sparse track fragments. After block **480**, these steps conclude.

[0051] FIG. **5** is a diagram that illustrates one or more data structures used by the sparse streaming system to associate continuous and sparse fragments, in one embodiment. In the diagram, fragments arrive from left to right over time, including a continuous video data stream **510** and a sparse text data stream **550**. The video data stream **510** includes fragments **520** that arrive on a regular schedule over time. The text data stream **550** includes fragments **560** that arrive irregularly. Each of the video fragments **520** points to the most recent text data fragment. For example, video fragments V**1** and V**2** point to text fragment T**1**, video fragment V**3** points to text fragment T**2**, and video fragments V**4** and V**5** point to text fragment T**4**. Each of the text fragments **560** also includes a reverse pointer to the previous text fragment. For example, text fragment T**4** points to T**3**, T**3** to T**2**, and T**2** to T**1**. Forward pointers in between video track fragments are specified using a custom MP4 box in the response. Pointers to text track fragments in the video track responses are indicated in the HTTP "Pragma" header. Reverse pointers in the text track responses are also indicated using the HTTP "Pragma" header.

[0052] The diagram illustrates two situations that can occur between the video data stream **510** and the text data stream **550**. First, multiple video fragments **520** may arrive between text fragments **560**. This is illustrated by video fragments V**1** and V**2** that both point to text fragment T**1**. The client detects that V**2** points to the same fragment as V**1**, and if it has already retrieved T**1** after receiving V**1**, does not perform any additional action for the text data stream when V**2** arrives. When V**3** arrives, the client recognizes that V**3** points to a new text fragment T**2** and retrieves T**2**. The second situation is that multiple text data fragments may arrive between video fragments, as illustrated by T**3**, which arrives between V**3** and V**4**. The reverse pointers allow the client to recognize that it needs to retrieve T**3** by following the path from V**4** to T**4** to T**3**. When the client follows the path from T**3** to T**2**, the client recognizes that T**2** has already been retrieved and stops following the chain. The client can then display T**3** and T**4** based on any instructions or protocol associated with the text data stream **550**.

[0053] The client can also perform error recovery on the text data stream **550** even if some frames of the video data stream **510** are not received. For example, if the client fails to receive V**3** and V**4** but receives V**5**, the client can follow the reference from V**5** to T**4**, then from T**4** to T**3** and T**3** to T**2**. Thus, if the text data stream includes information that the client is under contract to retrieve, but the video stream can be skipped, then the client can retrieve the missing text data stream fragments without going back and retrieving the missing video stream fragments.

[0054] In some embodiments, the sparse streaming system allows clients to view previously live content on-demand after the live event has concluded. The same continuous and sparse fragments that clients retrieved during the live event can also be retrieved by clients connecting after the event has occurred. These fragments may also be available from Internet cache servers without a particular request reaching a content network.

[0055] In some embodiments, the sparse streaming system provides multiple versions of sparse data tracks. For example, the system may provide localized versions of a track containing closed captions so that viewers having different native languages can each select a track in their own native language. Previous systems that embedded this information within a media presentation often asked clients to download an entirely different presentation to switch languages. However, the sparse streaming system allows clients to switch sparse tracks "on the fly" without affecting the video presentation. For example, while only one set of video fragments may be available in caches throughout the network, a client can request multiple different fragments for a sparse track. In addition, although a client may have been requesting sparse track fragments in English during the beginning of the presentation, the client can switch to Spanish sparse fragments at any point during the presentation without interruption.

[0056] In some embodiments, the sparse streaming system supports multiple sparse tracks associated with a particular continuous track. Each sparse track may have a different purpose, such as a highlight track (for significant events in a presentation), a closed caption track, and so forth. Each fragment in the continuous track can include a list of sparse tracks associated with the continuous track. The sparse streaming system performs the steps described further herein to retrieve sparse fragments for each of the associated sparse tracks. In some embodiments, the system labels each sparse fragment using a combination of a track name, track discriminating attributes, and a time position to allow the fragment to be reliably located by clients when failover events occur.

[0057] A highlight may include any interesting segment of media, such as a point during a sporting event during which a player scored a goal. The client can play a highlight reel after an event has concluded by playing those media fragments of the media with associated with highlight markers. If the client did not receive the live event, the client can request the manifest for the media and then request only those media fragments corresponding to the highlights. If a user wants to see more of the media before and after the highlight (e.g., as indicated by the user fast-forwarding or rewinding), then the client can request additional media fragments to play the requested portions of the media.

[0058] In some embodiments, the sparse streaming system allows a client connecting after a live event has started to catch up with sparse data previously created in association with a sparse data track. When a provider declares that a track is a "control track," the system cumulatively aggregates the track's data into the manifest downloaded by the client when playback is initiated. The content provider may have contracts or other agreements with third parties that specify that each client view particular information (e.g., an advertisement) before seeing the main presentation. The system can provide a late connecting client with information about each of the sparse fragments associated with a presentation without the client having to separately request each sparse fragment. This increases network efficiency and allows third party business logic to make decisions about whether the client retrieves each sparse fragment. In addition, clients can display a timeline of the entire event (e.g., with skip markers highlighting interesting events) after having downloaded the manifest without requesting each fragment of the presentation.

[0059] In some embodiments, the sparse streaming system allows third parties to extend the system by including custom data in sparse fragments. Note that the sparse streaming system is not particularly concerned with what data is contained in each of the sparse fragments. Thus, a particular third party can extend the system by including information that allows the third party to make particular decisions at the client. For example, a third party may want to show advertisements to clients at particular points in a presentation, or to come out of an advertisement if a particularly significant event occurs (e.g., a goal scored in a sporting event) during the presentation.

[0060] In some embodiments, the sparse streaming system receives sparse data from data sources over stateless connections. For example, a data source may connect to a content server, upload one or more fragments of sparse data, then disconnect. In previous systems, sources of data typically maintain a connection with the server for the duration of a multimedia event, consuming valuable server resources. Using the sparse streaming system, data sources can provide sparse data as sparse data is available without holding a constant connection open between the data source and the server.

[0061] In some embodiments, the sparse streaming system may include spatial information in association with sparse fragments. For example, a sparse fragment containing an advertisement may include a location where the content provider requests that the client overlay the advertisement on the main presentation (e.g., upper right quadrant). In addition, the sparse fragments may indicate at what point the sparse data will be displayed, so that the client can receive sparse data in advance of the time it is to be consumed. For example, a client may receive an advertisement three minutes into a presentation that contains instructions to play the advertisement four minutes into the presentation. The extra time may allow the client to retrieve a particular advertisement from a third party advertising and tracking provider.

[0062] In some embodiments, the sparse streaming system includes an HTTP "Pragma" header with responses for continuous tracks to specify the time position of the last known fragment for any associated sparse tracks. The system may also use the HTTP "Pragma" header to specify the reverse pointers between sparse fragments described further herein.

[0063] In some embodiments, the sparse streaming system supports inline advertising. For a live event, it may be unknown at the start of the event when commercial breaks will occur. An event coordinator may press a button during production when it is time for a commercial, causing the system to insert an advertising marker in the media stream metadata or as a sparse track. When the client receives the advertising marker, the client may request and receive media fragments associated with a previously identified advertisement. For example, the system may provide a list of potential advertisements in an initial manifest. The advertisement may be provided in media fragments similar to other media, and may not be stored at the same server that provides the live event. Upon encountering an advertisement marker, the client pauses playback of the main stream, retrieves and displays the advertisement, and then resumes playback of the main stream. Unlike previous systems that used techniques such as polling an XML file to identify new advertisements to display, advertisements become a dynamic part of a content provider's workflow using the sparse streaming system.

[0064] In some embodiments, the sparse streaming system determines which sparse tracks to provide based on a subscription or other payment model. For example, a content provider may charge more for commentary related to a live event. In this case, the system may enable or disable switching to particular sparse tracks based on whether the conditions of the payment model have been met (e.g., the user's account is current). This information may be included in the manifest provided to the client. The content provider may offer some streams free, such as a low bit rate or highlight only media, while charging for others.

[0065] The sparse streaming system may request and receive media content in a variety of encodings. In some embodiments, the sparse streaming system uses custom MP4 boxes. The Motion Picture Experts Group (MPEG) version 4 standard provides for boxes within the format that can contain custom data. The MP4 extension is the file format commonly associated with this version of content. The system may leverage boxes to include the custom metadata and media content chunks. Other media formats provide similar customization of content within a container and may be used by the system.

[0066] In some embodiments, the sparse streaming system conforms to the guidelines of the Representational State Transfer (REST) style of software architecture for distributed hypermedia systems. One concept in REST is that an application can interact with a resource by knowing only the identifier of the resource (e.g., a URI) and the action requested (e.g., retrieval), and without knowing whether there are caches, proxies, gateways, firewalls, tunnels, or anything else between the application and the server actually holding the information. Following REST guidelines allows the system to use existing Internet infrastructure and pre-existing resource-conserving techniques such as caching. Some example REST-based principles that the system implements

in some embodiments include: each URI identifies exactly one response, each URI points to a server resource that is stateless and cacheable, and each URI is intuitive and uses nouns (verbs are HTTP verbs). In particular, the system may avoid making requests using query strings and may use substantially unique keys for start times that are requested via URLs.

[0067] From the foregoing, it will be appreciated that specific embodiments of the sparse streaming system have been described herein for purposes of illustration, but that various modifications may be made without deviating from the spirit and scope of the invention. For example, although certain types of sparse data have been used as examples, those of ordinary skill in the art will appreciate that the system can be used with many, diverse types of data. Accordingly, the invention is not limited except as by the appended claims.

I/We claim:

1. A computer-implemented method for preparing sparse data for retrieval by a client, the method comprising:
- receiving a current sparse track fragment;
- identifying a previous sparse track fragment received prior to the current sparse track fragment;
- storing a reference to the identified previous sparse track fragment in association with the received current sparse track fragment;
- identifying a current continuous track fragment associated with the current sparse track fragment;
- storing a reference to the received current sparse track fragment in association with the identified current continuous track fragment; and
- providing the identified current continuous track fragment including the stored reference to the received current sparse track fragment to at least one client,
- wherein the preceding steps are performed by at least one processor.

2. The method of claim **1** wherein receiving the current sparse track fragment includes aggregating sparse track fragments from multiple sources.

3. The method of claim **1** wherein providing the identified current continuous track fragment includes providing an HTTP-cacheable delivery of textual metadata associated with a video presentation.

4. The method of claim **1** wherein providing the identified current continuous track fragment includes providing an on-demand version of a previous live media presentation.

5. The method of claim **1** wherein receiving the current sparse track fragment includes receiving multiple localized versions of a sparse data track.

6. The method of claim **1** wherein storing the reference to the received current sparse track fragment in association with the identified current continuous track fragment comprises storing the reference in a way that allows multiple sparse tracks to be associated with a single continuous track fragment.

7. The method of claim **1** further comprising providing the received current sparse track fragment in response to a client request.

8. The method of claim **1** further comprising providing a manifest to the client that allows the client to retrieve a subset of the sparse track fragments available to the client.

9. The method of claim **1** further comprising providing a manifest to the client that informs the client of previously missed sparse track fragments.

10. The method of claim **1** wherein the current sparse track fragment includes information that can trigger a client to perform one or more actions.

11. The method of claim **1** wherein the current sparse track fragment includes spatial information to inform a client of advertisement insertion points in a continuous video track.

12. A computer system for integrating sparse and continuous streaming data, the system comprising:
- a processor and memory configured to execute software instructions;
- an encoder interface component configured to provide an interface between the system and one or more encoders that provide encoded media data as continuous media fragments;
- an index fragment component configured to create and maintain an index table of media fragments received from encoders;
- a fragment data store configured to store received media fragments and the created index table of fragments to provide to clients based on received client requests;
- a sparse fragment component configured to receive sparse fragments from one or more data sources and prepare the fragments for retrieval by clients; and
- a client interface component configured to receive client requests for continuous and sparse fragments and provide manifest data and requested fragments to clients.

13. The system of claim **12** wherein the sparse fragment component is further configured to receive sparse data from a variety of sources and aggregate the data.

14. The system of claim **12** wherein the sparse fragment component is further configured to receive data associated with an announcer commenting on a live sporting event.

15. The system of claim **12** wherein the sparse fragment component is further configured to prepare the sparse fragments for retrieval by associating each sparse fragment with at least one continuous fragment and by associating each sparse fragment with at least one previous sparse fragment.

16. The system of claim **12** wherein the sparse fragment component identifies sparse fragments by at least a track name and a timestamp.

17. A computer-readable storage medium comprising instructions for controlling a computer system to request sparse track fragments associated with a continuous data track, wherein the instructions, when executed, cause a processor to perform actions comprising:
- requesting a continuous track fragment associated with the continuous data track of a media presentation;
- receiving the requested continuous track fragment;
- identifying a reference to a sparse track fragment associated with the continuous track fragment;
- if the sparse track fragment has not previously been received, requesting the identified sparse track fragment; receiving the requested sparse track fragment; and
- displaying the received continuous track fragment and any newly received sparse track fragment.

18. The medium of claim **17** further comprising, after receiving the requested sparse track fragment, identifying a reference to a previous sparse track fragment associated with the received sparse track fragment and if the previous sparse track fragment has not previous been received, retrieving the previous sparse track fragment.

19. The medium of claim **17** wherein receiving the requested continuous track fragment comprises receiving

error recovery information that allows discovering missed sparse track fragments by following a stored reference between a current sparse track fragment and a previous sparse track fragment without retrieving intervening continuous track fragments.

20. The medium of claim **17** wherein receiving the requested continuous track fragment comprises receiving the continuous track fragment from an edge server cache.

* * * * *