(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
26 November 2015 (26.11.2015)

WIPO | PCT

(10) International Publication Number
WO 2015/176659 A1

(51) International Patent Classification:
G06F 17/30 (2006.01)

(21) International Application Number:
PCT/CN2015/079385

(22) International Filing Date:
20 May 2015 (20.05.2015)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
14/285,204 22 May 2014 (22.05.2014) US

(71) Applicant: **HUAWEI TECHNOLOGIES CO., LTD.** [CN/CN]; Huawei Administration Building, Bantian,Longgang District, Shenzhen, Guangdong 518129 (CN).

(72) Inventors: **KOVVURI, Vaishnav**; 1055 Escalon Ave., Sunnyvale, CA California 94085 (US). **ZHAO, Jim**; 2035 Louise Ln., Los Altos, CA California 94024 (US).

(81) Designated States *(unless otherwise indicated, for every kind of national protection available)*: AE, AG, AL, AM, AO, AT, AU, AZ, BA, BB, BG, BH, BN, BR, BW, BY, BZ, CA, CH, CL, CN, CO, CR, CU, CZ, DE, DK, DM, DO, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, GT, HN, HR, HU, ID, IL, IN, IR, IS, JP, KE, KG, KN, KP, KR, KZ, LA, LC, LK, LR, LS, LU, LY, MA, MD, ME, MG, MK, MN, MW, MX, MY, MZ, NA, NG, NI, NO, NZ, OM, PA, PE, PG, PH, PL, PT, QA, RO, RS, RU, RW, SA, SC, SD, SE, SG, SK, SL, SM, ST, SV, SY, TH, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, ZA, ZM, ZW.

(84) Designated States *(unless otherwise indicated, for every kind of regional protection available)*: ARIPO (BW, GH, GM, KE, LR, LS, MW, MZ, NA, RW, SD, SL, ST, SZ, TZ, UG, ZM, ZW), Eurasian (AM, AZ, BY, KG, KZ, RU, TJ, TM), European (AL, AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HR, HU, IE, IS, IT, LT, LU, LV, MC, MK, MT, NL, NO, PL, PT, RO, RS, SE, SI, SK, SM, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, KM, ML, MR, NE, SN, TD, TG).

Declarations under Rule 4.17:

— *as to the applicant's entitlement to claim the priority of the earlier application (Rule 4.17(iii))*

Published:

— *with international search report (Art. 21(3))*

WO 2015/176659 A1

(54) Title: SYSTEM AND METHOD FOR PRE-FETCHING

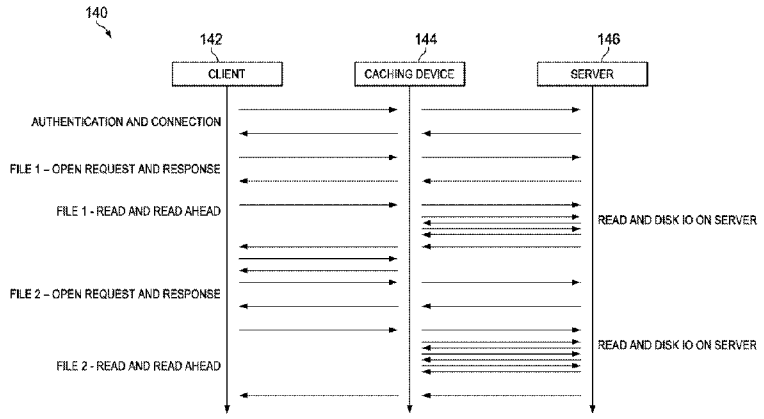

FIG. 3

(57) Abstract: In one embodiment, a method for pre-fetching files includes parsing a project file to produce a parsed project file and extracting a plurality of files from the parsed project file to produce a file list. The method also includes retrieving, by a caching device from a file server over a network, the plurality of files in accordance with the file list and storing the plurality of files in a cache.

SYSTEM AND METHOD FOR PRE-FETCHING

CROSS-REFERENCE TO RELATED APPLICATIONS

[0001] The application claims priority to U.S. non-provisional patent application Serial No. 14/285,204 filed on May 22, 2014 and entitled "System and Method for Pre-fetching", which is incorporated herein by reference as if reproduced in its entirety.

TECHNICAL FIELD

[0002] The present invention relates to a system and method for cache management, and, in particular, to a system and method for pre-fetching.

BACKGROUND

[0003] In today's enterprise world, there are geographically dispersed remote offices across the globe with a centralized headquarters and relatively few data centers. Data from the data centers may be shared around the globe across multiple remote offices over a wide area network (WAN). A WAN may be unreliable with limited bandwidth. Meanwhile, applications are becoming more bandwidth intensive, which indirectly creates performance issues for simple operations on files, such as reading and writing.

[0004] Applications use file sharing protocols. To improve performance when such protocols are used, intermediate caching devices are installed to cache the objects. Caches may be both read and write caches which cache the data for better user experience and provide better data consistency. Data caching is a mechanism for temporarily storing content on the edge side of the network to reduce bandwidth usage, server load, and perceived lag when that content is re-accessed by the user. Caching may be applied in a variety of different network implementations, such as in content distribution networks (CDNs), enterprise networks, internet service provider (ISP) networks, and others. Generally speaking, caching is performed by fetching content in response to a client accessing the content, storing the content in a cache for a period of time, and providing the content directly from the cache when the client attempts to re-access the content.

[0005] Protocols like common internet file system (CIFS) are chatty and perform multiple reads and writes of data. Also, protocols like hypertext transfer protocol (HTTP) bring in the same data over and over again when multiple users try to access the same data. Applications also

perform multiple iterations of the same file operations (open, read, close). Caching devices work around this by performing data caching and pre-fetching. Pre-fetching of data may be initiated when a user expresses interest in opening or reading a file. The user may experience slowness if the data is changed in the back-end file server, because the changed data flows in the network. In another example, an administrator of the device manually pre-loads the data before the user accesses the data. However, this may be error prone and not deterministic.

SUMMARY

[0006] An embodiment method for pre-fetching files includes parsing a project file to produce a parsed project file and extracting a plurality of files from the parsed project file to produce a file list. The method also includes retrieving, by a caching device from a file server over a network, the plurality of files in accordance with the file list and storing the plurality of files in a cache.

[0007] An embodiment method of opening files includes retrieving, by a caching device from a file server over a network, a plurality of files associated with a project file in a cache when a client initiates opening only the project file or a subset of the plurality of files and storing the plurality of files in a cache of the caching device. The method also includes receiving, by the caching device from a user, a file open request to open a first file, where the plurality of files includes the first file and reading the first file from the cache.

[0008] An embodiment caching device includes a processor and a computer readable storage medium storing programming for execution by the processor. The programming includes instructions to parse a project file to produce a parsed project file and extract a plurality of files from the parsed project file to produce a file list. The programming also includes instructions to retrieve, from a file server over a network, the plurality of files in accordance with the file list and store the plurality of files in a cache.

[0009] The foregoing has outlined rather broadly the features of an embodiment of the present invention in order that the detailed description of the invention that follows may be better understood. Additional features and advantages of embodiments of the invention will be described hereinafter, which form the subject of the claims of the invention. It should be appreciated by those skilled in the art that the conception and specific embodiments disclosed may be readily utilized as a basis for modifying or designing other structures or processes for carrying out the same purposes of the present invention. It should also be realized by those

skilled in the art that such equivalent constructions do not depart from the spirit and scope of the invention as set forth in the appended claims.

BRIEF DESCRIPTION OF THE DRAWINGS

[0010] For a more complete understanding of the present invention, and the advantages thereof, reference is now made to the following descriptions taken in conjunction with the accompanying drawing, in which:

[0011] Figure 1 illustrates an embodiment network for pre-fetching;

[0012] Figure 2 illustrates another embodiment network for pre-fetching;

[0013] Figure 3 illustrates a message diagram for file caching;

[0014] Figures 4A-D illustrates embodiment container files;

[0015] Figure 5 illustrates an embodiment system for pre-fetching;

[0016] Figure 6 illustrates a flowchart for an embodiment method of pre-fetching;

[0017] Figure 7 illustrates a flowchart for another embodiment method of pre-fetching; and

[0018] Figure 8 illustrates a block diagram of an embodiment general-purpose computer system.

[0019] Corresponding numerals and symbols in the different figures generally refer to corresponding parts unless otherwise indicated. The figures are drawn to clearly illustrate the relevant aspects of the embodiments and are not necessarily drawn to scale.

DETAILED DESCRIPTION OF ILLUSTRATIVE EMBODIMENTS

[0020] It should be understood at the outset that although an illustrative implementation of one or more embodiments are provided below, the disclosed systems and/or methods may be implemented using any number of techniques, whether currently known or in existence. The disclosure should in no way be limited to the illustrative implementations, drawings, and techniques illustrated below, including the exemplary designs and implementations illustrated and described herein, but may be modified within the scope of the appended claims along with their full scope of equivalents.

[0021] Remote offices are located around the world. Data transferred from centralized servers is affected by latency and bandwidth limitation of wide area networks (WANs), which generally are slower than a local area network (LAN). It is desirable, however, for a WAN user to have a LAN like user experience.

[0022] To improve the quality of user experience, intermediate caching devices initiate a pre-fetching of the file when the user expresses interest in it by initiating a first read on the file. In general, pre-fetching is initiated after the file is opened or the first block is read. However, users tend to work on a logical group of files or data sets associated as a project. Each project contains a few to many files. If the files are grouped together, the user tends to open some of the associated files soon after opening one of the associated files.

[0023] Files which are logically grouped together may form a project file or a container file. Project files contain metadata about the location of the files and the names of the files. The format of the project files may be text based for Make files, extensible markup language (XML) based for applications such as Visual Studio or AutoCAD, or in any other format, such as a batch file. When a remote user accesses the project files across a WAN, he is likely to open more than one file in the project. Because most of the file specific information is available in the project file, an embodiment caching system incorporates an infrastructure which parses the project files and performs pre-fetching operations on the files and/or directories. Because there are many applications with various formats of project files, an infrastructure takes in multiple formats in the form of plug-ins, where different plug-ins handle different types of projects. These plug-ins parse the respective formats and extract lists of pathnames and directories. This information is provided to the pre-fetch engine, which perform the pre-fetch of the files before the user actually issues an open or read on one of the files. The plug-ins may be loaded into the cache engine via

a common language infrastructure (CLI) or another means. The plug-in manager updates its database of available plug-ins directly so operations on the requested project file can be passed on to the correct plug-in. One example is application specific instead of protocol based. Applications such as AutoCAD, Eclipse, and Corel may be optimized differently even if they work on the same protocol across a WAN.

[0024] Figure 1 illustrates network environment 290 which supports file pre-fetching. As shown, the network environment 290 includes file server 292, caching device 296, network 294, and client 302. File server 292 may be any component or collection of components configured to store files. File server 292 may be a remote server which stores files to be accessed by remote clients, such as client 302.

[0025] Network 294 may be a WAN, a LAN, or another type of network. Files on file server 292 are accessed by client 302 over network 294.

[0026] Caching device 296 may be any component or collection of components configured to fetch files from files server 292 on behalf of client 302, and to cache the file so that the file may be accessed by client 302. Caching device 296 may include fetching module 298 for fetching the files and cache 300 for storing the files. Files are downloaded across network 294 from file server 292. Fetching module 298 fetches files from file server to cache 300 over network 294, from file server 292 over network 294 to client 302, and from cache 300 to client 302.

[0027] Client 302 may correspond to any entity (e.g., an individual, office, company, etc.) or group of entities (e.g., subscriber group, etc.) that access files stored in file server 292. In embodiments provided herein, caching device 296 may pre-fetch files and/or file updates from file server 292 prior to the files being re-accessed by client 302, and store the pre-fetched files in cache 300. The files may be pre-fetched based on a project opened by client 302, and may be provided directly from cache 300 to client 302 upon being re-accessed by client 302.

[0028] The embodiment pre-fetching techniques provided by this disclosure are applicable to any network environment in which files stored on one side of a network are cached on another side of the network, including content distributed networks (CDNs), enterprise networks, internet service provider (ISP) networks, wide area optimization networks, and others. Figure 2 illustrates network environment 100 with a data center and a branch office which communicate over a WAN. Data center 102 is coupled to branch office 104 via WAN 106. Data center 102

contains file server 112, which may be a Windows or Unix file servers. File server 112 stores files which may be remotely accessed. Data is stored in storage 110 and tape backup 114 in data center 102.

[0029] WAN optimization (WANO) box 116 performs WAN optimization to increase the data efficiency across WAN 106. WANO techniques include optimization in throughput, bandwidth requirements, latency, protocol optimization, and congestion avoidance.

[0030] Firewall 118 protects the data center. Firewall 118 is a network security system which controls the incoming and outgoing network traffic.

[0031] Router 120 interacts between data center 102 and WAN 106, while router 122 interacts between WAN 106 and branch office 104. Routers 120 and 122 forward data packets between data center 102 and branch office 104.

[0032] WAN 106 is coupled to router 122 in branch office 104. Firewall 124 protects branch office 104. Firewall 124 controls incoming and outgoing network traffic to provide security for branch office 104.

[0033] The data is received by WANO box 126 and disseminated to clients 128. WANO box 126 performs optimization to improve efficiency across WAN 106. Also, WANO box 126 contains cache for storing data. WANO boxes 116 and 126 may be any devices configured to provide an interface to the WAN 106, and may include fetching modules and/or other components for performing the pre-fetching and optimization techniques provided by this disclosure.

[0034] More information on pre-fetching is discussed in U.S. Patent Application Serial No. 14/231,508 filed on March 14, 2014, and entitled "Intelligent File Pre-Fetch Based on Access Patterns," which application is hereby incorporated herein by reference.

[0035] Figure 3 illustrates message diagram 140 for read ahead caching of individual files. Read ahead caching is performed on a per-file basis, where individual files are cached. When there is a collection of files, for example a project, files are pre-fetched one at a time. With an embodiment, multiple files may be pre-fetched at a time. The process begins when the client attempts to access a file, which prompts the caching device to send a file request to the file server to fetch a version of the file. Client 142 sends an authentication and connection request to caching device 144. Caching device 144 either authenticates or forwards the authentication and

connection request to server 146. In response, server 146 sends a response to caching device 144, which caching device 144 forwards to client 142.

[0036] Next, client 142 opens File 1 and requests to open the file. This request is sent to caching device 144 and passed on to server 146. Server 146 responds to caching device 144, and the response is sent to client 142. The file is then open.

[0037] Caching device 144 requests to read and read ahead from server 146 for file 1. Reading and disk input/output (IO) are performed on server 146 and the data is sent to caching device 144. Caching device 144 sends the read data to client 142. Also, caching device 144 pre-fetches on behalf of client 142 and performs read ahead.

[0038] Client 142 reopens and requests a response for file 2. As with file 1, Client 142 receives data for read and read ahead for file 2. This request is sent to caching device 144 and passed on to server 146. Server 146 responds to caching device 144, and the response is sent to client 142. The file is then open.

[0039] Often, files are logically grouped together in a collection of files as project files or container files. The project or container files contain the names and locations of the files in the project. Some examples of project or container files are .NET project files (.vcxproj), Eclipse project files (.project), Rstudio (.rproj), Qt project file (.pro), AutoCAD project file (.wdp, .wdd), Unix/Linus Makefile, A4desk (.a4p), Adobe device (.adcp), Anjuta integrated developer environment (IDE) (anjuta), Borland developer studio (.bdsproj), C# project file (.scproj), and Delphi Project (.dproj). Figures 4A-D illustrates some example project files. Figure 4A illustrates .NET project file 150, Figure 4B illustrates C# project file 160, Figure 4C illustrates Borland project file 170, and Figure 4D illustrates Makefile 180.

[0040] Figure 5 illustrates system 190 for pre-fetching project files. The files are pre-fetched when a container file is opened, or one of the member files of a container file is opened. System 190 detects a collection of files, and caches all the files in the associated project files. When a user requests to open a project file, open module 200 receives this request, and passes it on to plug-in manager 202. The request may be to open a project file, a file associated with a project file, or a file not associated with a project file. In one example, the file is already stored in cache. Alternatively, the file is not stored in cache.

[0041] Plug-in manager 202 manages plug-ins 192. Plug-in manager 202 is the master for plug-ins 192 and determines whether a file to be read is a recognized project file, associated with a

recognized plug-in, or neither. The type of plug-in for the format of the project file is determined, for example, based on a proprietary file format. When the file is a project file or a part of a project file, plug-in manager 202 passes the request to the correct plug-in, which parses the corresponding project file. The plug-in has a parser for the appropriate container file, and extracts the file to be fetched. The plug-in extracts the information from the project file, parses the information, prepares a list of complete file names, and passes it on to the plug-in manager.

[0042] The list of files is then passed to pre-fetch module 208. The files are fetched and saved in cache. These files are pre-fetched and stored by cache module 212 in cache 214, local persistent cache. The files are retrieved from remote server 204 over WAN 206 to be stored in cache 214. The files are stored in local persistent cache 214.

[0043] When a user requests to read one of these files, read module 210 retrieves the files from cache module 212. If the file is stored in cache 214 in a current version, cache module 212 reads the file from cache 214 and passes the data to read module 210, which provides a fast response. When the current version of the file is not stored in cache, it may be downloaded over the network from the remote server.

[0044] Figure 6 illustrates flowchart 220 for a method of pre-fetching project files. Initially, in step 222, a user initiates a file open. For example, the user opens a file stored on a remote server. The file may be a project file, a part of a project file, or a file not associated with a project file.

[0045] Next, in step 224, the open information is duplicated and sent to a plug-in manager. The open information is sent to the plug-in manager to open the file and other files in the project file.

[0046] Then, in step 226, the plug-in manager performs validation. The plug-in manager determines whether the plug-in is a project file or a part of a project file. When the file to be opened is not a part of a project file, only the file is opened. When the file to be opened is a project file or a part of a project file, the files in the project file are pre-fetched, because the user is likely to open them in the future. The plug-in manager determines the appropriate plug-in to open the files.

[0047] In step 228, the plug-in manager determines if the appropriate plug-in is available. The plug-in manager may download, update, or delete a plug-in to obtain the appropriate plug-in. When the appropriate plug-in is not available, the system does nothing in step 230. When the plug-in is available, the plug-in parses the project file in step 232.

[0048] After the project file is parsed, a list of files to be pre-fetched is extracted by the plug-in in step 234. In one example, all of the files in the project file are pre-fetched. Alternatively, only a portion of the files are pre-fetched.

[0049] Next, in step 236, the project files are pre-fetched by the pre-fetch module. The files from the list determined in step 234 are pre-fetched and stored in persistent cache 238. The files may later be accessed from the cache.

[0050] When the user later wants to open a file, the files may be quickly read from persistent cache 238. To read a file which is already stored in cache, the user initiates a read of file1 in step 240.

[0051] A read module verifies that the latest copy of the file is stored in cache 238 in step 242. There may be an older version of the file in the cache which is not the most current version. For example, a new version of the file may be updated on the remote server, but this new version has not yet been downloaded to the cache. Then, in step 244, it determines whether the local copy in cache is the latest version. When the latest copy is not stored in the cache, for example when the file has been updated, or if it was never pre-fetched, the system reads the file in step 248. The file is read across the WAN in step 250. This may lead to a delay.

[0052] When the latest copy is stored in cache, the system reads the file, in step 246, from persistent cache 238. This may be performed quickly.

[0053] Figure 7 illustrates flowchart 310 for a method of pre-fetching files. Initially, in step 340, a user initiates opening a file.

[0054] In step 316, the caching device determines whether the file is a container file. This may be done by determining whether the file is a proprietary container file. When the file is a part of a project file, the project file may be accessed. When the file is not a project file or a part of a project file, the caching device proceeds to step 314. When the file is a part of a project file or is a project file the caching device proceeds to step 318.

[0055] In step 314, the caching device determines whether the file is already in cache. When the file is already in the cache, the system proceeds to step 326. On the other hand, when the file is not stored in the cache, the system proceeds to step 324.

[0056] The caching device fetches a single file over a network in step 324. The network may be a WAN, or another network. The single file is read in over the network from a remote server. Also, the file is saved in cache for later access.

[0057] In step 326, the caching device determines whether the version of the file in the cache is the latest version of the file. When the version of the file in the cache is the latest version of the file, the system reads the file from the cache in step 328. When the version of the file in the cache is not the latest version of the file, the system fetches the file over the network in step 324. In this case, the file is opened with some delay. The file is also stored in the cache for later access.

[0058] In step 318, the caching device determines an appropriate plug-in for the project file, and that the plug-in is available. The plug-in manager examines the container file, and determines whether an appropriate plug-in is available. It may add a new plug-in, update an existing plug-in, or delete a plug-in as necessary. When the plug-in is not available, the system does not pre-fetch the project files in step 330. When the appropriate plug-in is available, the system proceeds to step 320.

[0059] In step 320, the caching device extracts the files from the container file. The container file is parsed and the files are extracted to create a list of files. The list may contain the name of the files and their locations.

[0060] Finally, in step 322, the files are pre-fetched over the network. At a later time, when the user initiates a read of one of the files in the container file, it may be quickly read from cache.

[0061] As used herein, the term "pre-fetching the file" refers to the action of fetching an electronic file without being prompted to do so by a client attempting to access the electronic file. Moreover, the term "file" is used loosely to refer to any object (e.g., file content) having a common characteristic or classification, and therefore the phrase "pre-fetching the file" should not be interpreted as implying that the electronic file being fetched is identical to "the [electronic] file" that was previously accessed by the client. For example, the file being pre-fetched may be an updated version of an electronic file that was previously accessed by the client. As another example, the file being pre-fetched may be a new instance of a recurring electronic file type that was previously accessed by the client, e.g., a periodic earnings report, an agenda, etc. In such an example, the client may not have accessed any version of the electronic file being pre-fetched. To illustrate the concept, assume the client is a newspaper editor that edits a final draft of the Tuesday's Sports Section, and that the caching device pre-fetches an electronic version of a final draft of Wednesday's Sport Section. The phrase "prefetching the file" should be interpreted to encompass such a situation even though the content of Wednesday's Sports Section differs from

that of Tuesday's Sports Section, as (in this instance) "the file" refers to a type or classification associated with Tuesday's and Wednesday's Sports Section, rather than the specific content of Tuesday's Sports Section.

[0062] Figure 8 illustrates a block diagram of processing system 270 that may be used for implementing the devices and methods disclosed herein. Specific devices may utilize all of the components shown, or only a subset of the components, and levels of integration may vary from device to device. Furthermore, a device may contain multiple instances of a component, such as multiple processing units, processors, memories, transmitters, receivers, etc. The processing system may comprise a processing unit equipped with one or more input devices, such as a microphone, mouse, touchscreen, keypad, keyboard, and the like. Also, processing system 270 may be equipped with one or more output devices, such as a speaker, a printer, a display, and the like. The processing unit may include central processing unit (CPU) 274, memory 276, mass storage device 278, video adapter 280, and I/O interface 288 connected to a bus.

[0063] The bus may be one or more of any type of several bus architectures including a memory bus or memory controller, a peripheral bus, video bus, or the like. CPU 274 may comprise any type of electronic data processor. Memory 276 may comprise any type of system memory such as static random access memory (SRAM), dynamic random access memory (DRAM), synchronous DRAM (SDRAM), read-only memory (ROM), a combination thereof, or the like. In an embodiment, the memory may include ROM for use at boot-up, and DRAM for program and data storage for use while executing programs.

[0064] Mass storage device 278 may comprise any type of storage device configured to store data, programs, and other information and to make the data, programs, and other information accessible via the bus. Mass storage device 278 may comprise, for example, one or more of a solid state drive, hard disk drive, a magnetic disk drive, an optical disk drive, or the like.

[0065] Video adaptor 280 and I/O interface 288 provide interfaces to couple external input and output devices to the processing unit. As illustrated, examples of input and output devices include the display coupled to the video adapter and the mouse/keyboard/printer coupled to the I/O interface. Other devices may be coupled to the processing unit, and additional or fewer interface cards may be utilized. For example, a serial interface card (not pictured) may be used to provide a serial interface for a printer.

[0066] The processing unit also includes one or more network interface 284, which may comprise wired links, such as an Ethernet cable or the like, and/or wireless links to access nodes or different networks. Network interface 284 allows the processing unit to communicate with remote units via the networks. For example, the network interface may provide wireless communication via one or more transmitters/transmit antennas and one or more receivers/receive antennas. In an embodiment, the processing unit is coupled to a local-area network or a wide-area network for data processing and communications with remote devices, such as other processing units, the Internet, remote storage facilities, or the like.

[0067] While several embodiments have been provided in the present disclosure, it should be understood that the disclosed systems and methods might be embodied in many other specific forms without departing from the spirit or scope of the present disclosure. The present examples are to be considered as illustrative and not restrictive, and the intention is not to be limited to the details given herein. For example, the various elements or components may be combined or integrated in another system or certain features may be omitted, or not implemented.

[0068] In addition, techniques, systems, subsystems, and methods described and illustrated in the various embodiments as discrete or separate may be combined or integrated with other systems, modules, techniques, or methods without departing from the scope of the present disclosure. Other items shown or discussed as coupled or directly coupled or communicating with each other may be indirectly coupled or communicating through some interface, device, or intermediate component whether electrically, mechanically, or otherwise. Other examples of changes, substitutions, and alterations are ascertainable by one skilled in the art and could be made without departing from the spirit and scope disclosed herein.

WHAT IS CLAIMED IS:

1. A method for pre-fetching files, the method comprising:

parsing a project file to produce a parsed project file;

extracting a plurality of files from the parsed project file to produce a file list;

retrieving, by a caching device from a file server over a network, the plurality of files in accordance with the file list; and

storing the plurality of files in a cache.

2. The method of claim 1, wherein the project file is an extensible markup language (XML) file.

3. The method of claim 1, wherein the project file is a text file.

4. The method of claim 1, wherein the network is a wide area network (WAN).

5. The method of any one of claims 1 to 4, wherein extracting the plurality of files is performed by a plug-in.

6. The method of claim 5, further comprising:

identifying the plug-in in accordance with a type of the project; and

determining whether the plug-in is available.

7. The method of claim 6, further comprising updating the plug-in when the plug-in is not available or a newer version of the plug-in is available.

8. The method of claim 6, further comprising downloading the plug-in when the plug-in is not available or a newer version of the plug-in is available.

9. The method of any one of claims 1 to 8, further comprising receiving, by the caching device from a user, a file open request to open the project file.

10. The method of any one of claims 1 to 8, further comprising receiving, by the caching device from a user, a file open request to open a first file associated with the project file.

11. The method of claim 10, further comprising receiving, by the caching device from a user, a file open request to open the first file after storing the plurality of files in the cache.

12. The method of claim 11, further comprising determining whether a version of the first file is stored in the cache.

13. The method of claim 12, further comprising determining whether the version of the first file is a current version.

14. The method of claim 13, further comprising reading the version of the first file from the cache when the version of the first file is the current version.

15. The method of claim 13, further comprising retrieving, by the caching device from the file server over the network, the first file when the version of the first file is not the current version.

16. A method of opening files, the method comprising:

retrieving, by a caching device from a file server over a network, a plurality of files associated with a project file in a cache when a client initiates opening only the project file or a subset of the plurality of files;

storing the plurality of files in a cache of the caching device;

receiving, by the caching device from a user, a file open request to open a first file, wherein the plurality of files comprises the first file; and

reading the first file from the cache.

17. The method of claim 16, further comprising determining whether a version of the first file in the cache is a current version, wherein reading the first file from the cache is performed when the version of the first file in the cache is the current version.

18. The method of claim 17, further comprising:

retrieving, by the caching device from the file server over the network, the first file when the version of the first file in the cache is not the current version; and

storing the first file in the cache.

19. A caching device comprising:

a processor; and

a computer readable storage medium storing programming for execution by the processor, the programming including instructions to

parse a project file to produce a parsed project file,

extract a plurality of files from the parsed project file to produce a file list,

retrieve, from a file server over a network, the plurality of files in accordance with

the file list, and

store the plurality of files in a cache.

20. A caching device comprising:

a processor; and

a computer readable storage medium storing programming for execution by the processor, the programming including instructions to

store a plurality of files associated with a project file in cache when a client initiates opening only the project file or a subset of the plurality of files,

receive, from a user, a file open request to open a first file, wherein the plurality of files comprises the first file, and

read the first file from the cache.

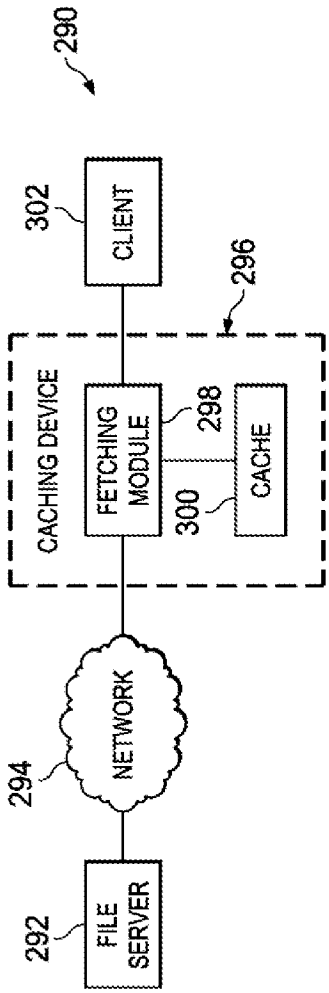
290
302
CLIENT
296
CACHING DEVICE
FETCHING MODULE
298
CACHE
300
294
NETWORK
292
FILE SERVER

FIG. 1

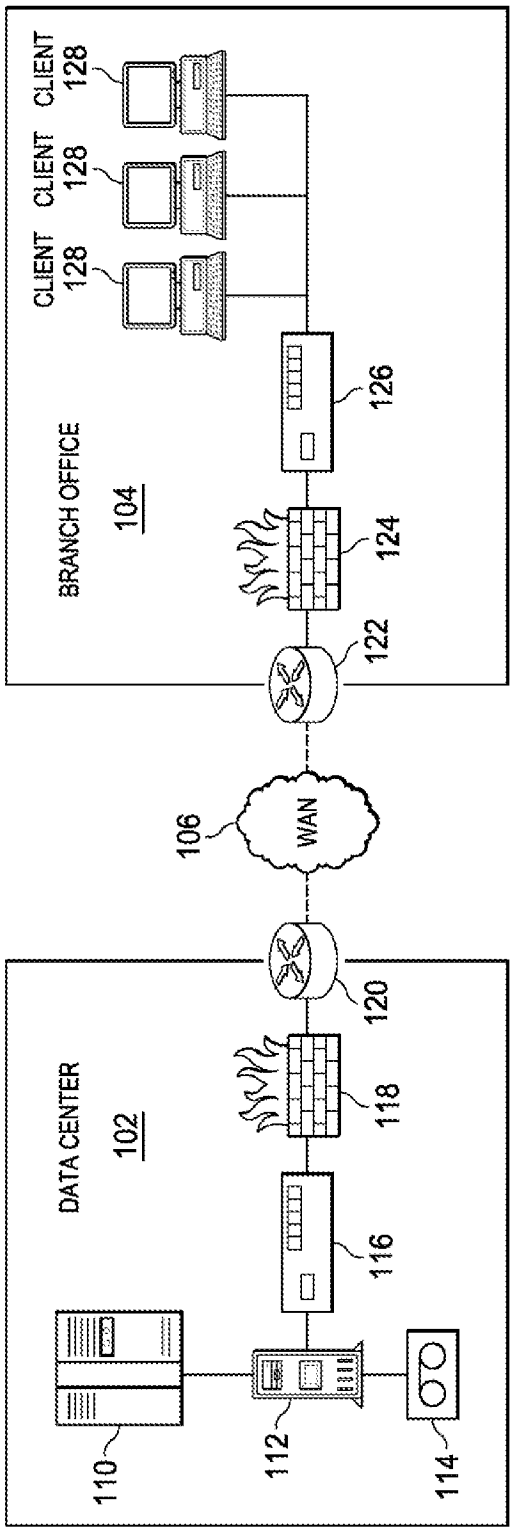
100
CLIENT 128
CLIENT 128
CLIENT 128
BRANCH OFFICE
104
126
124
122
106
WAN
120
DATA CENTER
102
118
116
110
112
114

FIG. 2

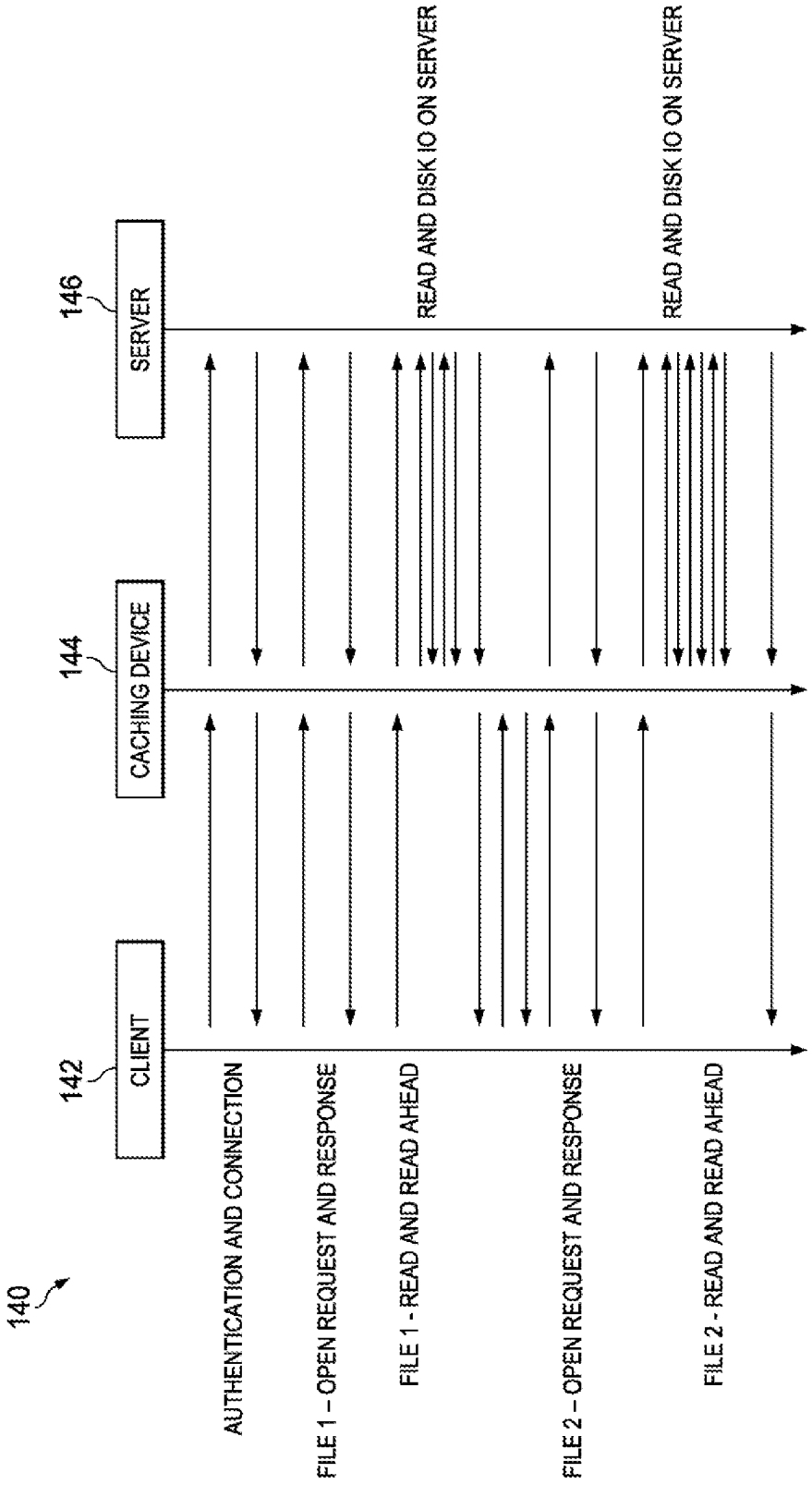
140
142
CLIENT
144
CACHING DEVICE
146
SERVER
AUTHENTICATION AND CONNECTION
FILE 1 – OPEN REQUEST AND RESPONSE
FILE 1 - READ AND READ AHEAD
READ AND DISK IO ON SERVER
FILE 2 – OPEN REQUEST AND RESPONSE
FILE 2 - READ AND READ AHEAD
READ AND DISK IO ON SERVER

FIG. 3

150

Xml

```
<Project DefaultTargets="Build" ToolsVersion="12.0" xmlns="http://schemas.microsoft.com/developer/msbuild/2003">
  <ItemGroup>
    <ProjectConfiguration Include="Debug|Win32">
      <Configuration>Debug</Configuration>
      <Platform>Win32</Platform>
    </ProjectConfiguration>
    <ProjectConfiguration Include="Release|Win32">
      <Configuration>Release</Configuration>
      <Platform>Win32</Platform>
    </ProjectConfiguration>
  </ItemGroup>
  <Import Project="$(VCTargetsPath)\Microsoft.Cpp.default.props" />
  <PropertyGroup>
    <ConfigurationType>Application</ConfigurationType>
    <PlatformToolset>v120</PlatformToolset>
  </PropertyGroup>
  <Import Project="$(VCTargetsPath)\Microsoft.Cpp.props" />
  <ItemGroup>
    <ClCompile Include="main.cpp" />
  </ItemGroup>
  <ItemGroup>
    <ClInclude Include="main.h" />
  </ItemGroup>
  <Import Project="$(VCTargetsPath)\Microsoft.Cpp.Targets" />
</Project>
```

FIG. 4A

```
<ItemGroup>
  <Reference Include="Microsoft.CSharp" />
  <Reference Include="System.Runtime.Serialization" />
  <Reference Include="System.ServiceModel" />
  ...
</ItemGroup>
<ItemGroup>
  <Compile Include="Controllers\AccountController.cs" />
  <Compile Include="Controllers\ContactsController.cs" />
  <Compile Include="Controllers\HomeController.cs" />
  ...
</ItemGroup>
<ItemGroup>
  <Compile Include="Content\Custom.css" />
  <Compile Include="CreateDatabase.sql" />
  <Compile Include="DropDatabase.sql" />
  ...
</ItemGroup>
```

160

FIG. 4B

```
<?xml version="1.0" encoding="utf-8"?>
<BorlandProject>
    <Transactions>
  <Transactions>2009/11/06 09:51:45.562,bdsproj,C:\Documents and Settings\Alain\My Documents\Borland Studio Projects\Project
  <Transactions>2009/11/06 09:51:52.468,pas,C:\Documents and Settings\Alain\My Documents\Borland Studio Projects\Unit1.pas=
  <Transactions>2009/11/06 09:51:52.468,dfm,C:\Documents and Settings\Alain\My Documents\Borland Studio Projects\Unit1.dfm=
  <Transactions>2009/11/06 10:26:48.375,pas,C:\Documents and Settings\Alain\My Documents\Borland Studio Projects\Unit1.pas=
  <Transactions>2009/11/06 10:26:48.375,dfm,C:\Documents and Settings\Alain\My Documents\Borland Studio Projects\Unit2.dfm=
  <Transactions>2009/11/06 10:26:53.109,pas,C:\Documents and Settings\Alain\My Documents\Borland Studio Projects\Unit2.pas=
  <Transactions>2009/11/06 10:26:53.109,dfm,C:\Documents and Settings\Alain\My Documents\Borland Studio Projects\Unit2.dfm=
  <Transactions>2009/11/06 17:55:14.513,bdsproj,C:\Documents and Settings\Alain\My Documents\Borland Studio Projects\KeyMan
    </Transactions>
</BorlandProject>
```

170

FIG. 4C

180

```
# Makefile to build monte_pi_sprng program
# - - - macros
CC=cc
CFLAGS= -O3 -I /usr/local/lib/sprng/include -I /usr/local/lib/pgplot -g
OBJECTS= monte_pi_sprng.o plot.o
LIBS = -L/usr/local/lib/sprng/lib -llcg -L/usr/local/lib/pgplot -lcpgplot -lpgplot -lX11 -lftn -lm


# - - - targets
all:        monte_pi_sprng
monte_pi_sprng:     $ (OBJECTS)
          $ (CC)   -o monte_pi_sprng   $ (OBJECTS)   $ (LIBS)

monte_pi_sprng.o:      /usr/local/lib/sprng/include/sprng.h   /usr/local/lib/pgplot/cpgplot.h monte_pi_sprng.c
          $ (CC) $ (CFLAGS) -c monte_pi_sprng.c

plot.o:  /usr/local/lib/pgplot/cpgplot.h plot.c
          $ (CC) $ (CFLAGS) -c plot.c

$ - - - remove binary and executable files
clean:
          rm -f monte_pi_sprng $ (OBJECTS)
```

FIG. 4D

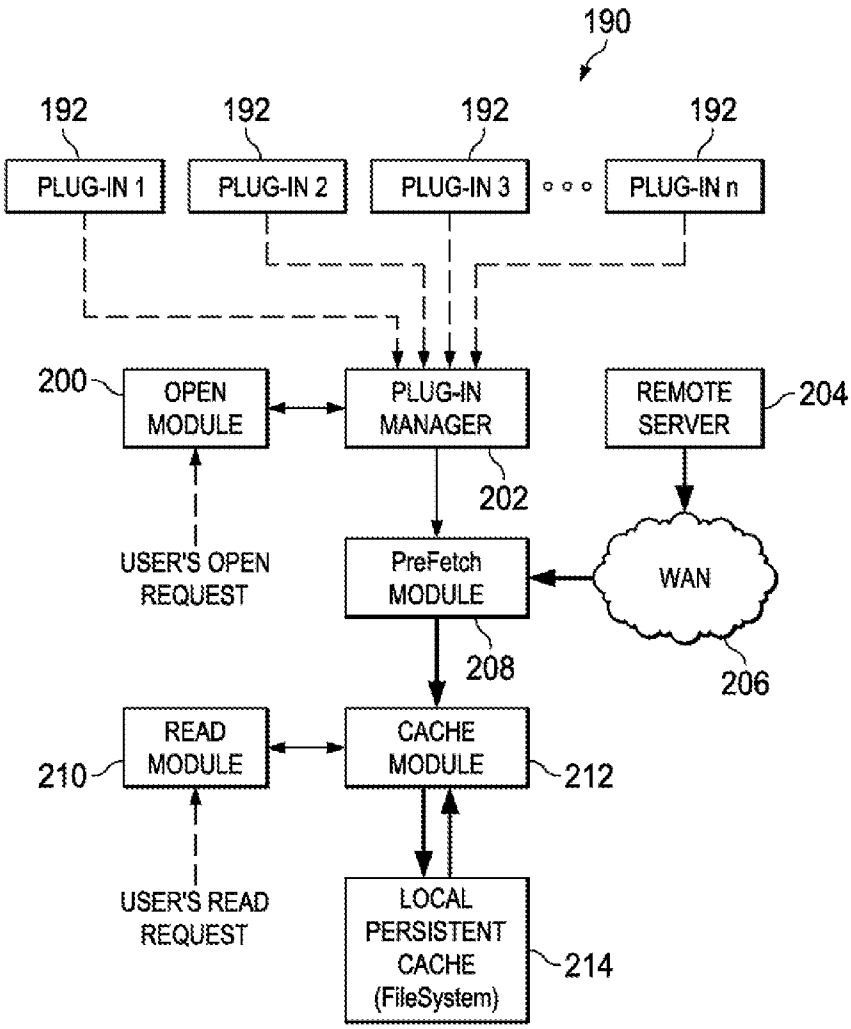
190
192
192
192
192
PLUG-IN 1
PLUG-IN 2
PLUG-IN 3
PLUG-IN n
200
OPEN MODULE
PLUG-IN MANAGER
REMOTE SERVER
204
202
USER'S OPEN REQUEST
PreFetch MODULE
WAN
206
208
READ MODULE
CACHE MODULE
210
212
USER'S READ REQUEST
LOCAL PERSISTENT CACHE (FileSystem)
214

FIG. 5

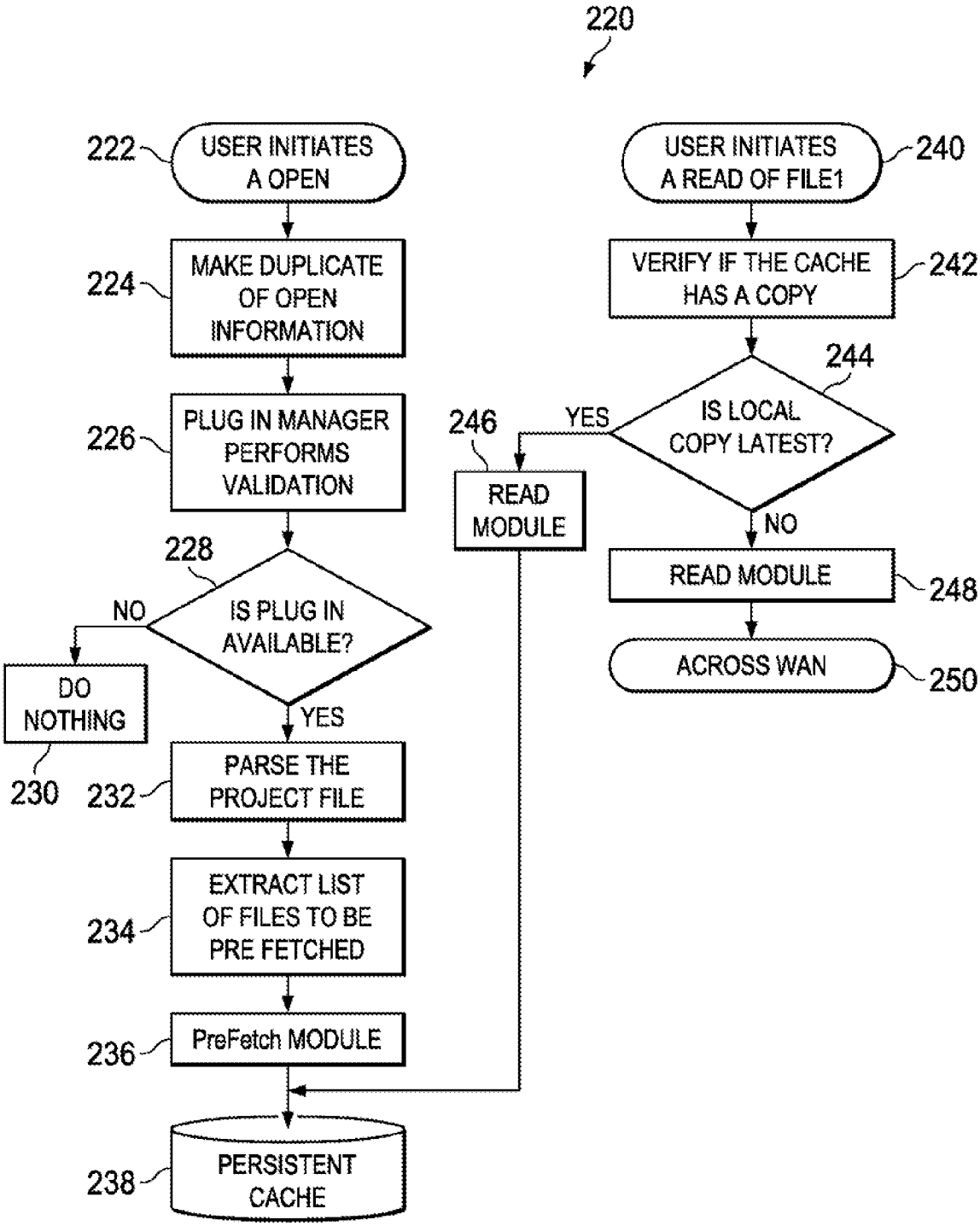
220
222
USER INITIATES A OPEN
224
MAKE DUPLICATE OF OPEN INFORMATION
226
PLUG IN MANAGER PERFORMS VALIDATION
228
IS PLUG IN AVAILABLE?
NO
DO NOTHING
230
YES
232
PARSE THE PROJECT FILE
234
EXTRACT LIST OF FILES TO BE PRE FETCHED
236
PreFetch MODULE
238
PERSISTENT CACHE
240
USER INITIATES A READ OF FILE1
242
VERIFY IF THE CACHE HAS A COPY
244
IS LOCAL COPY LATEST?
246
YES
READ MODULE
NO
248
READ MODULE
250
ACROSS WAN

FIG. 6

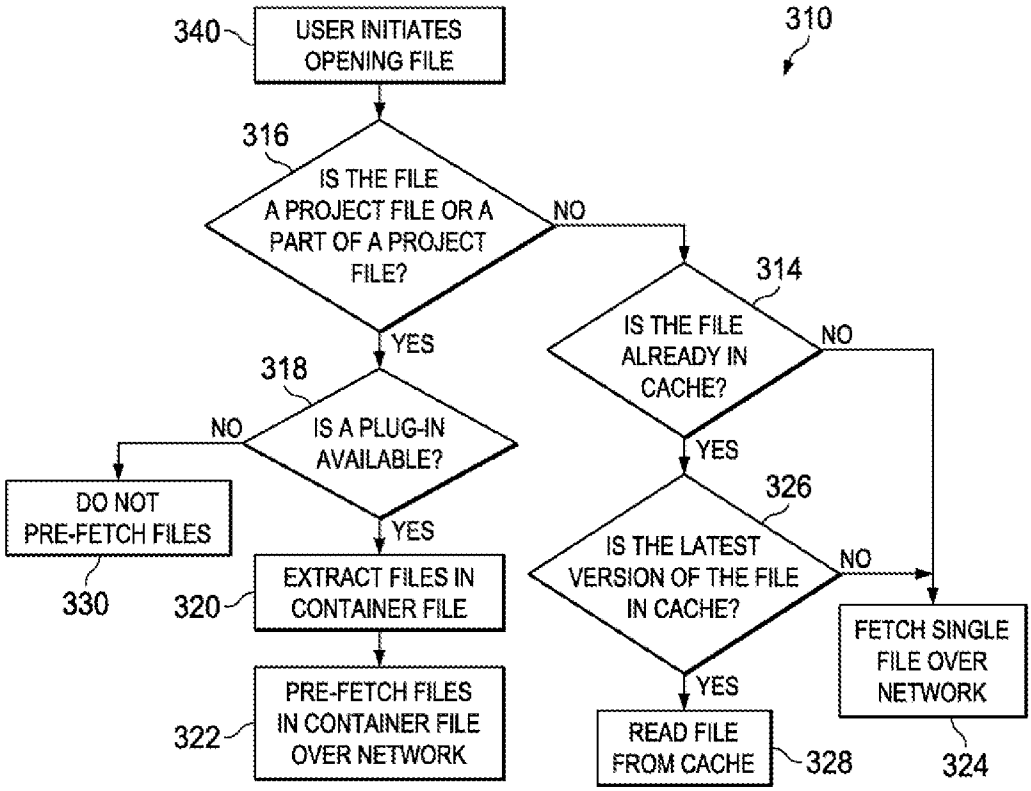
340
USER INITIATES OPENING FILE
310
316
IS THE FILE A PROJECT FILE OR A PART OF A PROJECT FILE?
NO
YES
314
IS THE FILE ALREADY IN CACHE?
NO
YES
318
IS A PLUG-IN AVAILABLE?
NO
YES
326
IS THE LATEST VERSION OF THE FILE IN CACHE?
NO
YES
DO NOT PRE-FETCH FILES
330
320
EXTRACT FILES IN CONTAINER FILE
322
PRE-FETCH FILES IN CONTAINER FILE OVER NETWORK
FETCH SINGLE FILE OVER NETWORK
324
READ FILE FROM CACHE
328

FIG. 7

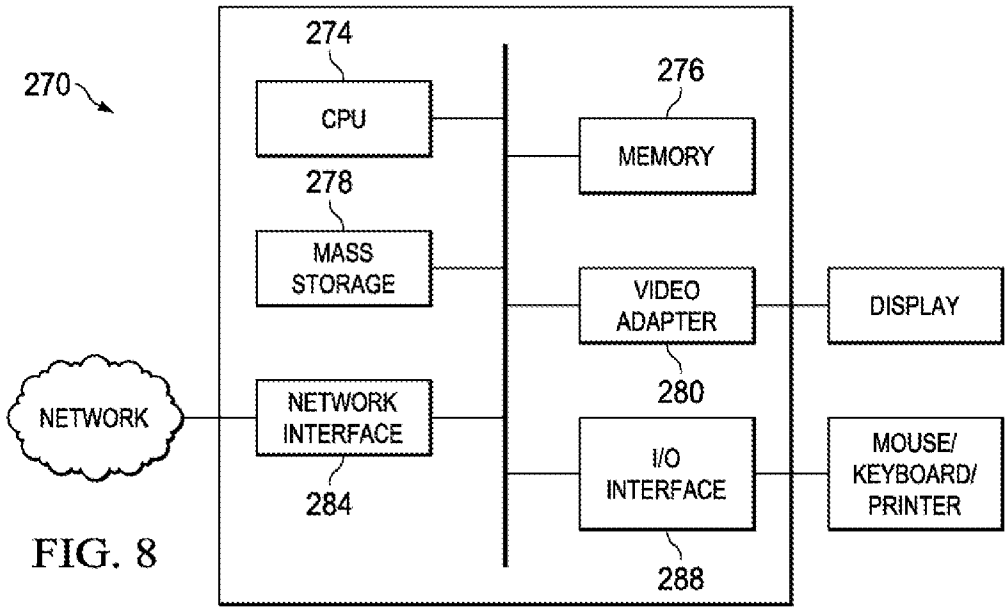
270
274
CPU
276
MEMORY
278
MASS STORAGE
VIDEO ADAPTER
DISPLAY
280
NETWORK
NETWORK INTERFACE
284
I/O INTERFACE
MOUSE/ KEYBOARD/ PRINTER
288

FIG. 8

INTERNATIONAL SEARCH REPORT

International application No.
PCT/CN2015/079385

A. CLASSIFICATION OF SUBJECT MATTER

G06F 17/30(2006.01)i

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)

G06F

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)

CNABS;VEN;CNKI:cache, file, pre fetch+, project, list?, pre load, distribut+, fetch

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	CN 102541985 A (DAWNING INFORMATION IND BEIJING CO. LTD.) 04 July 2012 (2012-07-04) description Para.[0017]-[0018]	1-4, 16-20
A	CN 102541985 A (DAWNING INFORMATION IND BEIJING CO. LTD.) 04 July 2012 (2012-07-04) the whole document	5-15
A	CN 101814038 A (HANGZHOU SHUNWANG TECHNOLOGY CO. LTD.) 25 August 2010 (2010-08-25) the whole document	1-20
A	US 7181574 B1 (VERITAS OPERATING CORP) 20 February 2007 (2007-02-20) the whole document	1-20

☐ Further documents are listed in the continuation of Box C. ☑ See patent family annex.

* Special categories of cited documents:

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier application or patent but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art

"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
10 July 2015	**23 July 2015**
Name and mailing address of the ISA/CN **STATE INTELLECTUAL PROPERTY OFFICE OF THE P.R.CHINA** **6, Xitucheng Rd., Jimen Bridge, Haidian District, Beijing 100088, China** Facsimile No. **(86-10)62019451**	Authorized officer **GUO,Shumei** Telephone No. **(86-10)62411657**

INTERNATIONAL SEARCH REPORT
Information on patent family members

International application No.
PCT/CN2015/079385

Patent document cited in search report	Publication date (day/month/year)	Patent family member(s)	Publication date (day/month/year)
CN 102541985 A	04 July 2012	None	
CN 101814038 A	25 August 2010	CN 101814038 B	03 October 2012
US 7181574 B1	20 February 2007	None	