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

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
3 November 2005 (03.11.2005)

PCT

(10) International Publication Number
WO 2005/103878 A2

(51) International Patent Classification[7]: G06F 7/00

(21) International Application Number:
PCT/IL2005/000419

(22) International Filing Date: 21 April 2005 (21.04.2005)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
60/565,298 26 April 2004 (26.04.2004) US

(71) Applicant *(for all designated States except US)*: **STOREWIZ, INC.** [US/US]; 21 Firstfield Road Suite, 260 Gaithersburg MD 20878 (US).

(72) Inventors; and

(75) Inventors/Applicants *(for US only)*: **SEVER, Gil** [IL/IL]; 76 Revivim Street, 48621 Rosh Haayin (IL). **AMIT, Noach** [IL/IL]; 20A Disraeli Street, 34334 Haifa (IL). **KEDEM, Nadav** [IL/IL]; 19 Shasha Argov Street, 69620 Tel Aviv (IL). **COHEN, Yakov** [IL/IL]; 10 Givat Hbraha, 44833 Elon Morhe (IL).

(74) Agent: **REINHOLD COHN AND PARTNERS**; P.O.B. 4060, 61040 Tel Aviv (IL).

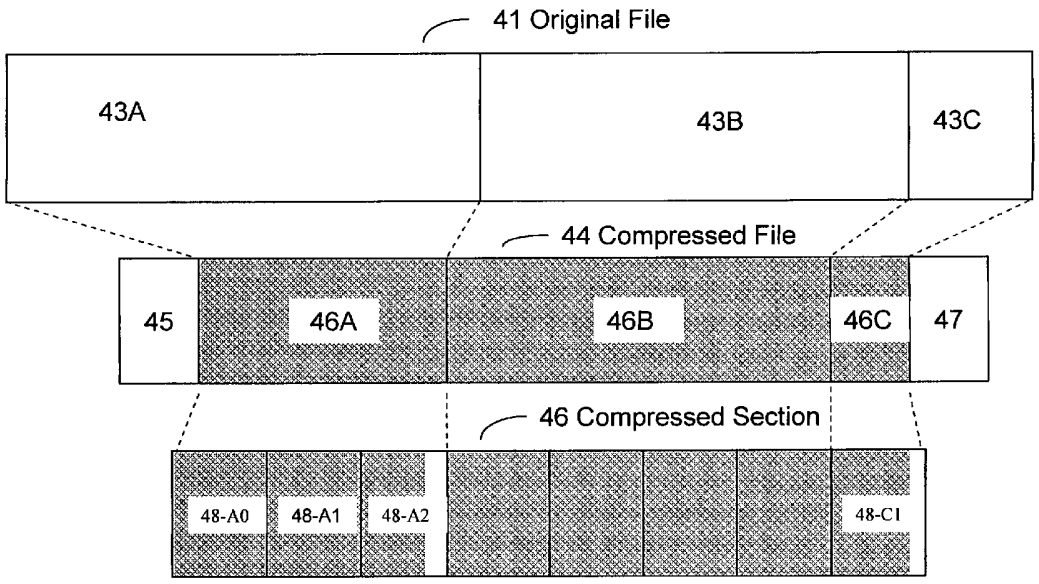
(81) Designated States *(unless otherwise indicated, for every kind of national protection available)*: AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BW, BY, BZ, CA, CH, CN, CO, CR, CU, CZ, DE, DK, DM, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KM, KP, KR, KZ, LC, LK,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

(84) Designated States *(unless otherwise indicated, for every kind of regional protection available)*: ARIPO (BW, GH, GM, KE, LS, MW, MZ, NA, SD, SL, SZ, TZ, UG, ZM, ZW), Eurasian (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European (AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HU, IE, IS, IT, LT, LU, MC, NL, PL, PT, RO, SE, SI, SK, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

Published:
— *without international search report and to be republished upon receipt of that report*

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

(54) **Title:** METHOD AND SYSTEM FOR COMPRESSION OF FILES FOR STORAGE AND OPERATION ON COMPRESSED FILES



(57) **Abstract:** A method and system for creating, reading and writing compressed files for use with a file access storage. The compressed data of a raw file are packed into plurality of compressed units and stored as compressed file. One or more corresponding compressed units may be read and/or updated with no need of restoring the entire file whilst maintaining de-fragmented structure of the compressed file.

WO 2005/103878 A2

METHOD AND SYSTEM FOR COMPRESSION OF FILES FOR STORAGE AND OPERATION ON COMPRESSED FILES

FIELD OF THE INVENTION

This invention relates to network infrastructure devices supporting network access to stored data and, in particular, to a method and apparatus facilitating compression and operation on file system with compressed data over file systems protocols.

BACKGROUND OF THE INVENTION

In current business environment, all types of business data are becoming more and more critical to business success. The tremendous growth and complexity of business-generated data is driving the demand for information storage, defining the way of sharing, managing and protection of information assets.

Typically, no single technology or architecture is able to address all needs of any organization. Main storage technologies are described in the White Paper by EMC, "Leveraging Networked storage for your business", 2003, (www.emc.com/pdf/products/networked_storage/leveraging_net_storage.pdf) and basically can be identified by connection type (direct attached storage (DAS), IP or channel networks) and by the method that data is accessed. There are three basic types of storage architectures to consider in connection with methods of data access: Block Access, File Access, and Object Access.

In block access architecture, the communication between a server/client and a storage device occurs in terms of blocks; information is pulled block by block directly of the disk. The operation system keeps track of where each piece of information is on the disk, while the storage device is usually not aware of the file system used to organize the data on the device. When something needs to get

read or be written, the data are directly accessed of the disk by that processor which knows where each block of data is located on the disk and how to put them together. The examples of block access storage technologies are DAS (Direct attached Storage), SAN (Storage Area Network) and Block Storage over IP (e.g. FCIP, iFCP, iSCSI, etc.).

File access requires the server or client to request a file by name, not by physical location. As a result, a storage device is usually responsible to map files back to blocks of data for creating, maintaining and updating the file system. The file server receives the file read or write request and handles the block access "behind the scene". The examples of file access storage technologies are NAS (Network Attached Storage with NFS, CIFS, HTTP, etc. protocols) and MPFS (Multi-pass File Serving). The file access storage may be implemented, for example, for general purpose files, web applications, engineering applications (e.g. CAD, CAM, software development, etc.), imaging and 3D data processing, multi-media streaming, etc.

Object access further simplifies data access by hiding all the details about block, file and storage topology from the application. The object access occurs over API integrated in content management application. The example of object access storage technology is CAS (Content Addressed Storage).

More efficient use of storage may be achieved by data compression before it is stored. Data compression techniques are used to reduce the amount of data to be stored or transmitted in order to reduce the storage capacity and transmission time respectively. The compression may be achieved by using different compression algorithms, for instance, a standard compression algorithm, such as that described by J. Ziv and A. Lempel, "A Universal Algorithm For Sequential Data Compression," IEEE Transactions on Information Theory, IT-23, pp. 337-343 (1997). It is important to perform compression transparently meaning that the data can be used with no changes to existing applications. In either case, it is necessary to provide a corresponding decompression technique to enable the original data to be reconstructed and accessible to applications. When an update

is made to a compressed data, it is generally not efficient to decompress and recompress the entire block or file, particularly when the update is to a relatively small part of data.

Various implementations of optimization of storage and access to the stored data are disclosed for example in the following patent publications:

U.S. Patent No. 5,761,536 (Franaszek) discloses a system and method for storing variable length objects such that memory fragmentation is reduced, while avoiding the need for memory reorganization. A remainder of a variable length object may be assigned to share a fixed-size block of storage with a remainder from another variable length object (two such remainders which share a block are referred to as roommates) on a best fit or first fit basis. One remainder is stored at one end of the block, while the other remainder is stored at the other end of the block. The variable length objects which are to share a block of storage are selected from the same cohort. Thus, there is some association between the objects. This association may be that the objects are from the same page or are in some linear order spanning multiple pages, as examples. Information regarding the variable length objects of a cohort, such as whether an object has a roommate, is stored in memory.

U.S. Patent No. 5,813,011 (Yoshida et al.) discloses a method and apparatus for storing compressed data, wherein compressed file consists of: a header that carries information showing the position of a compression management table; compressed codes; and the compression management table that holds information showing the storage location of the compressed code of each original record.

U.S. Patent No. 5,813,017 (Morris et al.) discloses a method and means for reducing the storage requirement in the backup subsystem and further reducing the load on the transmission bandwidth where base files are maintained on the server in a segmented compressed format. When a file is modified on the client, the file is transmitted to the server and compared with the segmented compressed base version of the file utilizing a differencing function but without

decompressing the entire base file. A delta file which is the difference between the compressed base file and the modified version of the file is created and stored on a storage medium which is part of the backup subsystem.

U.S. Patent No. 6,092,071 (Bolan et al.) discloses a system for control of compression and decompression of data based upon system aging parameters, such that compressed data becomes a system managed resource with a distinct place in the system storage hierarchy. Processor registers are backed by cache, which is backed by main storage, which is backed by decompressed disk storage, which is backed by compressed disk storage then tape, and so forth. Data is moved from decompressed to compressed form and migrated through the storage hierarchy under system control according to a data life cycle based on system aging parameters or, optionally, on demand: data is initially created and stored; the data is compressed at a later time under system control; when the data is accessed, it is decompressed on demand by segment; at some later time, the data is again compressed under system control until next reference. Large data objects are segmented and compression is applied to more infrequently used data.

U.S. Patent No. 6,115,787 (Obara et al.) discloses a disk storage system, wherein data to be stored in the cache memory is divided into plural data blocks each having two cache blocks in association with track blocks to which the data belongs and are compressed, thus providing the storage of plural compressed records into a cache memory of a disk storage system in an easy-to-read manner. The respective data blocks after the compression are stored in one or plural cache blocks. Information for retrieving each cache block from an in-track address for the data block is stored as part of retrieval information for the cache memory. When the respective data blocks in a record is read, the cache block storing the compressed data block is determined based on the in-track address of the data block and the retrieval information.

U.S. Patent No. 6,349,375 (Faulkner et al.) discloses a combination of data compression and decompression with a virtual memory system. A number of computer systems are discussed, including so-called embedded systems, in

which data is stored in a storage device in a compressed format. In response to a request for data by a central processing unit (CPU), the virtual memory system will first determine if the requested data is present in the portion of main memory that is accessible to the CPU, which also happens to be where decompressed data is stored. If the requested data is not present in the decompressed portion of main memory, but rather is present in a compressed format in the storage device, the data will be transferred into the decompressed portion of main memory through a demand paging operation. During the demand paging operation, the compressed data will be decompressed. Likewise, if data is paged out of the decompressed portion of main memory, and that data must be saved, it can also be compressed before storage in the storage device for compressed data.

U.S. Patent No. 6,584,520 (Cowart et al.) discloses a method of storage and retrieval of compressed files, the method involves dynamically generating file allocation table to retrieve compressed file directly from compact disk read only memory.

U.S. Patent No. 6,678,828 (Pham et al.) discloses a secure network file access appliance supporting the secure access and transfer of data between the file system of a client computer system and a network data store. An agent provided on the client computer system and monitored by the secure network file access appliance ensures authentication of the client computer system with respect to file system requests issued to the network data store. The secure network file access appliance is provided in the network infrastructure between the client computer system and network data store to apply qualifying access policies and selectively pass through to file system requests. The secure network file access appliance maintains an encryption key store and associates encryption keys with corresponding file system files to encrypt and decrypt file data as transferred to and read from the network data store through the secure network file access appliance.

U.S. Patent Application No. 2004/030,813 (Benveniste et al.) discloses a method and system of storing information, includes storing main memory

compressed information onto a memory compressed disk, where pages are stored and retrieved individually, without decompressing the main memory compressed information.

U.S. Patent Application No. 2005/021,657 (Negishi et al.) discloses a front-end server for temporarily holding an operation request for a NAS server, which is sent from a predetermined client, is interposed between the NAS server and clients on a network. This front-end server holds information concerning a correlation among data files stored in the NAS server, optimizes the operation request received from the client based on the information, and transmits the operation request to the NAS server.

SUMMARY OF THE INVENTION

There is a need in the art to provide for a new system and method of compressed storage for use with file access storage systems with no derogating of storing and retrieving capabilities and with no need of a user's awareness of compression/decompression operations as well as the storage location of the compressed data. The invention, in some of its aspects, is aimed to provide a novel solution capable of facilitating random access to data in compressed stored files and, thus, enabling operations on the compressed data with no need in decompression of entire files.

In accordance with certain aspects of the present invention, there is provided a method and system for creating, reading and writing compressed files for use with file access storage; said method and system facilitating direct access to the compressed data whilst maintaining de-fragmentation of the compressed file.

According to certain aspects of the present invention said method comprises:

- compressing a raw file and thereby generating the compressed data, wherein at least one fixed-size portion of data (cluster) of the raw file is

sequentially processed into corresponding compressed section divided into at least one fixed-size compression logical units (CLU);

- facilitating storing of the compressed data as a compressed file, the compressed file containing the compressed sections corresponding to the clusters of the raw file and a header comprising unique file descriptor;
- creating a section table comprising at least one record describing a compressed section, said records holding at least information on CLUs corresponding to the compressed section and storage location pointers pertaining to said CLUs.

For reading data stored in a file compressed as above, said method further comprises:

- determining a serial number of first compressed section comprising data to be read;
- determining the CLUs corresponding to said compressed section and storage location thereof by referring to the section table;
- restoring the cluster from said compressed section;
- if the range of data to be read exceeds the size of the restored clusters, repeating the stages b) and c) for compressed sections with serial numbers incremented by 1, until all data to be read are restored.

For writing data to a file compressed as above, said method further comprises:

- determining a serial number of first compressed section comprising data to be updated (original section);
- determining the CLUs corresponding to said original compressed section and storage location thereof by referring to the section table;
- restoring the cluster from said original compressed section;
- calculating the offset of updating data within said cluster and facilitating the update of the required data range;
- compressing the updated cluster into an updated compressed section;

- facilitating overwriting of said original compressed section with said updated compressed section;
- updating the section table
- if the range of data to be write exceeds the size of the restored clusters, repeating the stages from b) to g) for compressed sections with serial numbers incremented by 1, until all required data are written.

In accordance with further aspects of the present invention the method comprises handling a list of free CLUs released during writing data to the compressed file, said list is handling during all sessions related to the file until the file is closed. The method further comprising comparing the numbers of CLUs required to said original N_o and updated N_u compressed section and facilitating one of the following:

- overwriting all CLUs corresponding to the original compressed section with CLUs corresponding to the updated compressed section if $N_o=N_u$;
- overwriting first N_u CLUs corresponding to the original compressed section with CLUs corresponding to the updated compressed section and updating the list of free CLUs about released CLUs if $N_o>N_u$;
- overwriting all CLUs corresponding to the original compressed section with CLUs corresponding to the updated compressed section and writing the rest of CLUs corresponding to the updated compressed section to the CLUs contained in the list of free CLUs, if $N_o<N_u< N_o+N_f$, where N_f is a number of CLUs in said list;
- overwriting all CLUs corresponding to the original compressed section with CLUs corresponding to the updated compressed section, writing the CLUs corresponding to the updated compressed section to the CLUs contained in the list of free CLUs, and continuous writing the rest of CLUs corresponding to the updated compressed section to next free storage location if $N_o+N_f<N_u$.

In accordance with further aspects of the present invention the method further comprises checking the list of free CLUs before closing the file and, if not empty,

- defining a CLU with the highest storage location pointer among CLUs comprised in the compressed sections (first CLU);
- facilitating moving the compressed data from said first CLU to a free CLU with lower storage location pointer (second CLU);
- assigning said second CLU to pertaining compressed section and said first CLU to the list of free CLUs;
- repeating the stages b)-d) until the storage location pointers of all CLUs comprised in compressed sections are lower than a pointer of any of CLU comprising in the list of free CLUs;

According to further aspects of the present invention said system for compressing files for storage comprises:

- a subsystem for compressing a raw file and thereby generating the compressed data, wherein each fixed-size portion of data (cluster) from the raw file is sequentially processing into a compressed section divided into fixed-size compression logical units (CLU);
- a subsystem for facilitating storing of the compressed data as a compressed file, the compressed file containing the compressed sections corresponding to the clusters of the raw file and a header comprising unique file descriptor;
- a subsystem for creating a section table comprising records of all compressed sections, each of said records holding information on CLUs corresponding to the compressed section and storage location pointers pertaining to said CLUs.

It is to be understood that the system according to the invention may be a suitably programmed computer. Likewise, the invention contemplates a computer program being readable by a computer for executing the method of the

invention. The invention further contemplates a machine-readable memory, tangibly embodying a program of instructions executable by the machine for executing the method of the invention.

BRIEF DESCRIPTION OF THE DRAWINGS

In order to understand the invention and to see how it may be carried out in practice, a preferred embodiment will now be described, by way of non-limiting example only, with reference to the accompanying drawings, in which:

Fig.1 is a schematic block diagram of typical NAS storage network architecture as is known in the art.

Figs. 2a) and 2b) are schematic block diagrams of NAS storage network architecture in accordance with certain embodiments of the present invention.

Fig. 3 is a schematic block diagram of the system functional architecture in accordance with certain embodiments of the present invention.

Fig.4 is a schematic diagram of raw and compressed files in accordance with certain embodiments of the present invention.

Fig. 5 is an exemplary structure of section table in accordance with certain embodiments of the present invention.

Fig. 6 is a generalized flowchart of operation of compressed file creation in accordance with certain embodiments of the present invention.

Fig. 7 is a generalized flowchart of read operation on a compressed file in accordance with certain embodiments of the present invention.

Fig. 8 is a generalized flowchart of write operation on a compressed file in accordance with certain embodiments of the present invention.

Fig. 9 is a generalized flowchart illustrating sequence of write operation on a compressed section in accordance with certain embodiments of the present invention.

Fig.10 is a generalized flowchart of CLU management during close operation on a file.

Figs. 11a- 11c are schematic illustrations of relationship between CLUs and assigned disk memory segments in accordance with certain embodiments of the present invention.

DETAILED DESCRIPTION OF THE INVENTION

In the following detailed description, numerous specific details are set forth in order to provide a thorough understanding of the invention. However, it will be understood by those skilled in the art that the present invention may be practiced without these specific details. In other instances, well-known methods, procedures, components and circuits have not been described in detail so as not to obscure the present invention.

Unless specifically stated otherwise, as apparent from the following discussions, it is appreciated that throughout the specification discussions, utilizing terms such as , "processing", "computing", "calculating", "determining", or the like, refer to the action and/or processes of a computer or computing system, or processor or similar electronic computing device, that manipulate and/or transform data represented as physical, such as electronic, quantities within the computing system's registers and/or memories into other data, similarly represented as physical quantities within the computing system's memories, registers or other such information storage, transmission or display devices.

Embodiments of the present invention may use terms such as, processor, computer, apparatus, system, sub-system, module, unit, device (in single or plural form) for performing the operations herein. This may be specially constructed for the desired purposes, or it may comprise a general purpose computer selectively activated or reconfigured by a computer program stored in the computer. Such a computer program may be stored in a computer readable storage medium, such as, but not limited to, any type of disk including floppy disks, optical disks, CD-ROMs, magnetic-optical disks, read-only memories (ROMs), random access memories (RAMs), electrically programmable read-only

memories (EPROMs), electrically erasable and programmable read only memories (EEPROMs), magnetic or optical cards, or any other type of media suitable for storing electronic instructions, and capable of being coupled to a computer system bus.

The processes/devices (or counterpart terms specified above) and displays presented herein are not inherently related to any particular computer or other apparatus, unless specifically stated otherwise. Various general purpose systems may be used with programs in accordance with the teachings herein, or it may prove convenient to construct a more specialized apparatus to perform the desired method. The desired structure for a variety of these systems will appear from the description below. In addition, embodiments of the present invention are not described with reference to any particular programming language. It will be appreciated that a variety of programming languages may be used to implement the teachings of the inventions as described herein.

Bearing this in mind, attention is drawing to **Fig.1** illustrating a schematic diagram of typical NAS storage network architecture as is known in the art. The files from clients **11** and/or servers **12** are transferred via IP network **13** to file storage device(s) **14**. The file storage devices may be specialized NAS file servers, general purpose file servers, SAN storage, stream storage device, etc. The IP network may be a local area network (LAN), wide area network (WAN), their combination, etc.

Referring to **Fig. 2a**, there is illustrated a schematic diagram of NAS storage network architecture in accordance with certain embodiments of the present invention. The communication between servers **12** and/or clients **11** with file storage device **14** passes through a platform **20**. The platform **20** comprises client/server interface **21** connecting one or more client/servers and file storage interface **22** for bridge/proxy connection with one or more file storage devices. The platform 20 acts as a proxy on selected transaction thus keeping the throughput on control transactions and proxy on data transactions. The platform 20 may support any physical interfaces (e.g. Ethernet, Fiber Channel, etc.) and

may preserve the storage device features such as, for example, redundancy, mirroring, snapshots, failover, rollback, management, etc. The platform may be seamlessly integrated with existing network infrastructure. For implementing the compression and access functions described below, typically, the platform does not require changes in any level above the file itself. A user need not be aware of the compression and decompression operations and the storage location of compressed data.

During "write" operation on the files to be compressed before storage, the files from clients **11** and/or servers **12** flow through the network **13** to the platform **20**, compressed by the platform and written at the file storage device **14**. Files containing different kinds of data (e.g. text, image, voice, etc.) may be compressed by different compression algorithms (see, for example, at http://www.data-compression.com). A "read" operation proceeds in reverse direction; the required files are retrieved by the platform, decompressed (partly or entirely, in accordance with required data range) and sent to the appropriate client/server. The compression/decompression operations in accordance with certain aspects of the present invention are further described with reference to **Figs. 3 – 9** below. **Fig. 2b** illustrates storage architecture with other embodiment of the present invention when the files received from the client/servers are compressed and written at the platform before transferring via the network **13**.

In accordance with certain embodiment, the platform **20** may provide also security functions as, for example, encryption, authentication, etc.

The platform **20** is configured to transfer selected transactions (typically, control related transactions, e.g. copy file, delete file, rename file, take a snapshot, etc.) between clients/servers and the file storage device in a transparent manner, while intervening in data related transactions (e.g. open, close, read, write, create, etc.) and some control related transactions as, for example, directory list command.

In certain embodiments of the invention the platform may be configured to compress also uncompressed files a priori stored at the file storage device 14

with no passing through the platform **20**. The platform may extract all or selected files from the storage device in accordance with pre-defined criteria files, compress them in accordance with present invention and re-write in the storage device in a compressed form.

In certain embodiments of the invention the platform **20** may also be configured to compress only selected passing files in accordance with pre-defined criteria (e.g. size, IP address, directory, file type).

The raw file (or its relevant part) is compressed by the platform **20** during or before writing to the file storage device **14**. Similarly, the file (or its relevant part) is decompressed by the platform **20** during or after reading from the file storage device **14**. In some embodiments of the invention the files may be transferred between the platform **20** and the file storage device **14** in a compressed manner; for example, a raw file may be compressed and written within the platform and the resulting compressed file may be further transferred to the file storage device and/or compressed file may be received from the file storage device for further read/write operations within the platform.

Note that the invention is not bound by the specific network architecture described with reference to **Figs. 1** and **2**. Those versed in the art will readily appreciate that the invention is, likewise, applicable to any network including NAS or other based on file-access storage sub-network with IP or channel-based communication. The functions of the platform **20** (or part of them) may be implemented in a stand-alone server(s) (as illustrated in **Fig. 2**), distributed between several platforms or integrated within other storage network elements (e.g. file servers, enterprise switches, etc.).

Fig. 3 illustrates a schematic functional block diagram of the platform **20** in accordance with certain embodiments of the present invention.

The main function of Input/Output block **31** is socket handling; the block is responsible on the integrity of the IP communication. The I/O block 31 receives new TCP packets, unpacks them and moves to a session manager 32.

The block **31** also gets data from the session manager, packs it in TCP structure and sends it to the requested address.

Session starts by "Open File" request and ends by "Close File" request received from the same IP address (user). The session manger **32** holds all the session's private data as, for example, source IP address, all files instance in use, session counters, session status, all instances for to the buffers in use, etc. The session manager also handles the "File Block" and releases all the relevant resources on disconnect. The session manager **32** loads session tasks to the dispatcher **33** for filtering of the received packets. The dispatcher sends CIFS/NFS packets to a CIFS/NFS manager **34** and forwards all other packets back to the I/O block (via the session manager) with no touch. The CIFS/NFS manager **34** integrates accordingly CIFS and NFS transactions and requests a file manger **35** for data related transactions (e.g. Open , Read , Write , Close, etc.) and a compression/decompression block **37** for compression/decompression operations in accordance with certain embodiments with the present invention. Generally, compression algorithms have several compression levels characterized by trade-off between compression efficiency and performance parameters. The compression block **37** may select the optimal compression level and adjust the compression ratio to number of sockets currently handling by input/output block **31** (and/or CPU utilization). The information on the selected compression level is kept in the compression portion of data. The file manager **35** is responsible for the integrity and operations on a file. It also combines all requests related to a file to enable sharing of the file manipulation. The compression/decompression block **37** is capable to read and decompress the buffer as well as to compress and write the CLUS. Memory buffer recourses are managed by a buffer manager **36** connected with the CIFS/NFS manager. Integrity manager **38** is connected with the I/O block, the session manager, the buffer manager and the file manager and responsible for synchronization and general control of all processes in the platform.

Those skilled in the art will readily appreciate that the invention is not bound by the configuration of **Figs. 3**; equivalent and/or modified functionality may be consolidated or divided in another manner.

Fig.4 illustrates a schematic diagram of raw and compressed files in accordance with certain embodiments of the present invention. The uncompressed raw file **41** is segmented into portions of data **43** with substantially equal predefined size (hereinafter referring as clusters). These clusters serve as atomic elements of compression/decompression operations during input/output transactions on the files. The segmentation of the raw file into clusters may be provided "on-the-fly" during the compression process, wherein each next portion **43** with a predefined size constitutes a cluster that is subjected to compression. In certain other embodiments of the invention, the segmentation may be provided before compression. The size of the last portion of the raw file may be equal or less than the predefined size of the cluster; in both cases this portion is handled as if it has a size of a complete cluster. The size of the clusters may be configurable; larger clusters provide lower processing overhead and higher compression ratio, while smaller clusters provide more efficient access but higher processing overhead. Also, the size of cluster depends on available memory and required performance, as compression/decompression process of each file session requires at least one cluster available in the memory while performance defines a number of simultaneous sessions. The number of clusters is equal to the integer of (size of the raw file divided by the size of cluster) and plus one if there is a remainder.

Alternatively, in certain other embodiments of the invention, the size of cluster may vary in accordance with predefined criteria depending, for example, on type of data (e.g. text, image, voice, combined, etc.). For example, each type of data may have predefined size of cluster and the platform during compression may select the appropriate size of cluster in accordance with data type dominating in the compressing portion of the raw file.

Each intra-file cluster **43** (e.g. **43A-43C** as illustrated in **Fig.4**) is compressed into respective compressed section **46** (e.g. **46A-46C** as illustrated in **Fig.4**). The clusters with the same size may naturally result in compressed sections with different size, depending on the nature of data in each cluster and compression algorithms. If a ratio of a cluster compression is less than a pre-defined value, the corresponding compressed section in the compressed file may comprise uncompressed data from this cluster. For instance, if the raw data in a give cluster is compressed to no less than X% (say 95%) of the original cluster size, than due to the negligible compression ratio, the corresponding section would accommodate the raw cluster data instead of the compressed data.

In certain embodiments of the invention, the compression process may include adaptive capabilities, providing optimal compression algorithm for each cluster in accordance with its content (e.g. different compression algorithms best suited for clusters with dominating voice, text, image, etc. data)

In accordance with certain embodiments of the present invention each compressed file **44** comprises a header **45**, several compressed sections **46** and a section table 47. The header **45** of the compressed file comprises unique file descriptor, the size of the raw file **41** and a signature indicating whether the file was processed by the platform **20** (also for files which were not compressed by the platform, e.g., because, of obtainable compression ratio less than a predefined value).

The number of compressed sections within the compressed file is equal to the number of clusters. In accordance with certain embodiments of the present invention, the data in the compressed sections **46** are stored in compression logical units (CLU) **48** all having equal predefined size (e.g., as illustrated in **Fig.4**, compression logical units **48A0-48A2** correspond to the compressed section **46A** which corresponds to the cluster **43A**). This predefined CLU size is configurable; larger CLUs provide lower overhead, while smaller CLUs lead to higher resolution. Also, in certain embodiments of the invention, the CLU size may be adjusted to the maximum and/or optimal CIFS/NFS packet length

The number of CLUs within a compressed section is equal to the integer of (size of the compressed section divided by the size of CLU) and plus one if there is a remainder. The last CLU in compressed section may be partly full (as, e.g. **48-A2, 48-C1** in **Fig. 4**). Such CLUs may be handled in the same manner as full CLUs. In certain embodiments of the invention, the last CLU in the last compressed section (as, e.g., illustrated by **48-C1** in **Fig. 4**) may be handled in a special manner; namely, to be cut to the exact compression size if partly full (further described with reference to **Fig. 9** below).

CLUs may be considered as a virtual portion of the compressed file formed by a virtual sequence of segments in the memory. The relationship between CLUs and assigned memory segments is further described with reference to **Fig. 11** below.

The section table **47** comprises records of all compressed sections **46** and specifies where to find CLUs corresponding to each of compressed sections. The record in respect of each compressed section (hereinafter section record) comprises a signature indicating if the section was compressed, overall size of the compressed section and a list of pointers pertaining to all CLUs contained in the section. Optionally the record may comprise indication of compressed algorithm used during compression of the corresponding cluster and size of cluster (if variable per predefined criteria). Preferably, the section table **47** is placed at the end of the compressed file as its length may change when the content of the file is updated (as be further illustrated the length of section table is proportional to a number of compressed sections and, accordingly, number of clusters).

Fig. 5 illustrates, by way of non-limiting example, an exemplary structure of section table of an exemplary file.

This exemplary file **50** (referred to also in further examples) has original size 3MB + 413bit, predefined cluster size 1M and CLU size 60K. Accordingly, the raw file contains 4 clusters (3 clusters of 1 MB and one which is partly full, but handled as complete cluster).

A record **51** of a compressed section comprises a signature **52**, size of the section **53** and several entries **54**. Each entry **54** of the section record comprises information about one of CLUs contained in the compressed section. The section table comprises relation between the physical location and the logical CLU #.

The clusters of the exemplary file **50** are compressed into compressed sections with respective sizes of, e.g., 301123, 432111, 120423 and 10342 bytes. As CLU length of 60K means 61440 bytes, the section #0 has 5 allocated CLUs ([301123 / 61440] + 1); section #1 has 8 allocated CLUs ([432111 / 61440] + 1); section #2 has 2 allocated CLUs ([120423 / 61440] + 1) and section #3 has 1 allocated CLU ([10342/ 61440] + 1). Totally, the compressed file will comprise 16 CLUs (with total size 15 * 61440 bytes + 10342 bytes), fixed length header (e.g. 24 bytes including 4 byte for the signature, 16 byte for the file ID (unique descriptor) and 4 byte for the info about original size), and section table with 4 section records.

If the exemplary file **50** was created as a new compressed file, the CLUs will be allocated sequentially, for example,

First 5 CLUs with pointers 1, 2,3,4,5 will be allocated to Section 0;

Next 8 CLUs with pointers 6, 7, 8,9,10,11,12,13 will be allocated to Section 1;

Next 2 CLUs with pointers 14, 15 will be allocated to Section 2;

Next 1 CLUs with pointer 16 will be allocated to Section 3.

The distribution of CLUs within the file may be changed after an update (as will be further described with a reference to **Figs. 8-11** below). For example,

CLUs with pointers 1, 4,5,6,9 will be allocated to Section 0;

CLUs with pointers 2,3,7,10,11,12,15,14 will be allocated to Section 1;

CLUs with pointers 8, 13 will be allocated to Section 2;

CLUs with pointer 16 will be allocated to Section 3.

(In the current example the updates had no impact on the size of the compressed sections).

When a file is created as a new compressed file, the virtual (logical) sequence of CLUs is the same as physical sequence of disk segments corresponding to the CLUs. In an updated file, virtual (logical) sequence of CLUs may differ from the physical sequence of disk segments corresponding to the CLUs. For example in the example above, the second CLU of the first cluster was initially located at a physical segment #2 wherein after the update it is located at the physical segment # 4. Each CLU is assigned to a segment in a memory, the correspondent segment is written in the offset of the header **45** length plus CLU's length multiplied by the segment serial number. For example, in the exemplary file above, when the second CLU of the first cluster is located at the physical segment #2, it is written in the storage location memory in the offset 24 bytes of the header plus 2*61440 bytes. When after an update this CLU is located at the physical segment #4, it's offset becomes 24 bytes of the header plus 4*61440 bytes.

In certain embodiments of the invention, the number of entries in each section record is constant and corresponds to the maximal number of CLUs which may be required for storing the cluster, accordingly the size of each section record is constant regardless of the actual number of CLUs comprised in the section; not in use entries may have special marks. The number of entries in the section records is equal to integer of size of cluster divided by the size of CLU plus one.

In the illustrated example with clusters predefined size 1MB and CLU's predefined size 60 K, each record of compressed section has 17 entries (integer of 1MB/60K plus one) each one having 4 bytes. Respectively, the illustrated section record **50** of the compressed section #0 has 5 entries containing information about physical location of the correspondent CLUs and 12 empty entries (marked, e.g. as -1). The size of section record is 72 bytes (4 bytes for info on the compressed section size and signature plus 17 entries * 4 bytes). The overall size of the section table is 288 bytes (4 compressed sections * 72 bytes for each section record).

In certain embodiments of the invention, the compressed data may be stored separately of the section table **47**. The section tables may be stored in the separate files within the same storage space as the compressed data or at different, potentially physically remote, storage space, e.g. on the platform **20**. The platform **20** shall be configured in a manner facilitating maintenance of association between the compressed data and the corresponding section tables during read/write operations.

Figs. 6-11 illustrate input/output operations performed on compressed file in accordance with certain embodiments of the present invention. Note that the platform **20** intervenes also in commands referring to a size of a raw file (e.g. DIR, STAT, etc.) keeping the size in the header of correspondent compressed file and providing said data upon request. Thus, for example, consider a file having file size X (in its raw form) and Y (<X) in its compressed form (as stored in the disk). In accordance with the specified characteristics, the file size stored in the header would be X (raw file size) maintaining thus full transparency insofar as system commands such as DIR, STAT are concerned.

Upon receiving an inbound request to open a specific file compressed in accordance with certain embodiments of the present invention (a user may be not aware that the file is compressed), the platform **20** transfers the request to the NAS system (storage device **14**) and receives a "Handle" reply serving as a key for the file management (or "Null" if the file not found). Following the received "Handle", the platform **20** reads the header **45** comprising the file ID (unique file descriptor) and the size of corresponding raw file. Per the file ID the platform **20** checks if there is a concurrent session related to the file. If "No", the platform generates a File Block comprising a unique file descriptor and the size of raw file. If the file is already in use, the platform adds additional session to the existing File Block. The "Handle" then is returned to a user to be sent to the platform with the following requests on file operations.

Open file operation also includes reading the section table 47 of the compressed file and obtaining information of all CLUs corresponding to the file.

From the moment the file is opened and until it is closed, the platform is aware of CLUs structure of the file and offset of any byte within the file.

Referring to **Fig. 6**, there is illustrated a generalized flowchart of compressed file creation in accordance with certain embodiments of the present invention. The process is initiated by an inbound create request. The platform **20** generates **60** outbound request to the file storage device **14**; and after confirmation, starts writing **61** a header of the compressed file at the file storage device. As described in **Figs. 4**, the header will include a file descriptor, a size of the raw uncompressed file and a signature indicating that the file was processed by the platform **20**. At the next step **62** the platform processes the first fixed-size portion (cluster) of the raw file into compressed section having size X. (The compression may be provided with a help of any appropriate commercial or specialized algorithm). The platform defines first free storage location for the first CLU, starts and handles continuous writing **63** of the compressed section in this and sequential CLUs for storing at the file storage device, and prepares **64** the pointers of the CLUs occupied during the process to be recorded in the section table. The platform repeats **65** the process for next clusters until the data of the entire file are written in the compressed form and the section table is created **66**. In certain embodiments of the invention, the section table may be stored out of the compressed file.

Referring to **Fig. 7**, there is illustrated a generalized flowchart of read operation on a compressed file in accordance with certain embodiments of the present invention.

The read operation starts with inbound read request **70** comprising input parameters (e.g. File Handle, Seek Number (data offset) and data length Y) and output parameters (e.g. target buffer address). An inbound read request identifies the offset (in raw file) and the range Y of data to read. The platform **20** calculates 71 the serial number of the 1st cluster to be read (hereinafter the starting cluster) as integer of (offset divided by size of the cluster) and plus one if there is a remainder. The number of clusters to be read is defined by integer of (range of

data to be read divided by size of the cluster) plus one. As a result, the platform defines the compressed section(s) with one-to-one correspondence to the clusters to be read and generates outbound read request **72**. The request is based on meta-data of compressed file (header and section table) pointing to the CLUs corresponding to the compressed section(s) to be read. In certain embodiments of the invention, the offset of the section table placed at the end of compressed file may be easily calculated as following: size of compressed file minus number of clusters multiplied by fixed size of section record.

In other embodiments the platform may be configured to facilitate association between the compressed data and the corresponding meta-data stored in a separate file.

In certain embodiments of the invention, the outbound read request may be sent specifying all the range of the data to be read. Alternatively, as illustrated in **Fig. 7**, the overall read request is handled in steps, and for read operation the platform maintains a buffer substantially equal to the size of cluster. The first outbound read request comprises pointers to CLUs contained in the compresses section of the starting cluster. The entire compressed section corresponding to the starting cluster is read **73** and then uncompressed **74** by the platform to the target buffer. At the next step the platform calculates **75** the required offset within the cluster and copies the required data **76** to be passed to the application. The required length of copying data is calculated as follows:

$$Length = Minimum\ \{data\ range\ Y; (cluster\ size - offset \bmod cluster\ size)\}$$

If the data range Y exceeds the cluster size, the operation is repeated **77**.

For example, referring to the exemplary file **50**, request is to read file data of 20 bytes length from the offset 1 MB + 1340. Reading will start from the second cluster and, accordingly, the required data are contained in compressed file starting from 2nd compressed section. The offset of the section table is defined as the size of compressed file minus number of clusters (4) * size of section record (72 bytes). The record of the 2nd compressed section in the section table contains CLUs with pointers 2,3,7,10,11,12,15,14. Accordingly, these

CLUs will be read to a temporary buffer in the platform **20** and uncompressed to 1MB buffer in the platform. Then 20 bytes from the buffer offset 1340 will be moved to the target (user's) buffer. The required length of copying data is 20 bytes (equal to minimum between 20 bytes and (1 MB-1340 bytes)). If the other request were to read file data of 2MB length from the same offset, the operation would be repeated in a similar manner to 3rd and 4th compressed sections; and the required length of data copying from the starting cluster is 1MB-1340 bytes (equal to minimum between 2MB and (1 MB-1340 bytes)).

Referring to **Fig. 8**, there is illustrated a generalized flowchart of write operation on a compressed file in accordance with certain embodiments of the present invention. An inbound write request **80** identifies the offset (in raw file) and the range Y of data to write. The platform **20** calculates **81** the serial number of the 1st cluster to be updated (overwrite) as integer of (offset divided by size of the cluster) and plus one if there is a remainder. The number of clusters to overwrite is defined by integer of (range of data to write divided by size of the cluster) and plus one if there is a remainder. As a result, the platform defines the compressed section(s) to overwrite and generates outbound read request in a manner similar to that described with reference to **Fig.7**. After the entire compressed section corresponding to the starting cluster is read **82** and then uncompressed **83** by the platform to the buffer, the platform calculates **84** the required offset within the cluster as described with reference to **Fig.7** and updates (overwrites) the required data range **85**. Then, the platform compresses **86** the updated cluster, updates the section table and requests to write **87** the new compressed section to the compressed file. If the data range Y exceeds the cluster size, the operation is repeated **88** for successive clusters. Upon the end of the process, the platform updates the section table **89**.

As described above, in certain embodiments of the present invention the storage location of required data may be accessed directly and, accordingly, read/update (and similar) operations require restoring merely the clusters containing the required data range and not the entire files.

Typically, file updating may cause fragmentation because of unused space aroused in allocated storage. **Figs. 9** and **10** illustrate fragmentation handling algorithms of CLU management in accordance with certain embodiments of the present invention. **Fig. 9** illustrates an algorithm of CLU management during write/update operation on a compressed section (step **87** in **Fig.8**) in accordance with certain embodiments of the present invention. Before writing the updated compressed section, the platform compares **91** the number of CLUs required for the updated and old compressed sections. If the number of CLUs is unchanged, the platform **20** requests to write the updated compressed section sequentially to all CLUs **92** corresponding to the old compressed section. If the new number of the required CLUs is less than the old number, the compressed section will be written sequentially on a part of CLUs corresponding to the old compression section. The information about released CLUs is updated **93** in a special list (queue) of free CLUs handled by platform **20** until the file is closed. If the new number of the required CLUs is more than the old number, the compressed section will be written sequentially on all CLUs corresponding to the old compression section **94** and then on CLUs taken from the free CLUs queue **95**. If still more CLUs are required, the platform will define the last CLU allocated to the file (*#n*) and request to write sequentially on CLUs starting with number (n+1) (**96**); the list of allocated CLUs will be accordingly updated **97**.

In certain embodiments of the invention the last CLU in the last compressed section (as illustrated by **48-C1** in Fig. **4**) may be handled in a special manner; namely, to be cut to the exact compression size if partly full. The section table will be written on the offset of the header length + (N-1)*CLU size + S_L, where N is a total number of allocated CLUs and S_L is the size of compressed data in the last CLU.

Fig.10 illustrates an algorithm of CLU management during close operation on a file, in accordance with certain embodiments of the invention.

Before closing 102 the file, the platform checks 101 if the list of free CLUs is empty. If the list still comprises CLUs, the platform 20 defines a CLU

with the highest storage location pointer among CLUs in-use. Compressed data contained in said CLU are transferred **103** to a free CLU with a lower pointer and the emptied CLU is added to the list of free CLUs. The process is repeated **104** until all the pointers of CLUs in-use are lower than the pointer of any CLU comprising in the list of free CLUs. The section table will be accordingly updated **105**. Such updates may occur per each of said CLU re-writing, after the end of entire re-writing process or in accordance with other predefined criteria. Upon the end of the process the file is closed and free CLUs are released **106**. The selection of free CLU for above process may be provided in accordance with different algorithms. For example, in certain embodiments of the invention said compressed data from the CLU with the highest storage location pointer may be transferred to the free CLU with the lowest storage location pointer.

Referring to **Figs. 11a- 11c**, there are illustrated relationship between CLUs and assigned disk memory segments in accordance with certain embodiments of the present invention. **Fig. 11a** illustrates exemplary file **50** illustrated in **Fig.5** when created as new compressed file. The virtual (logical) sequence of CLUs is the same as physical sequence of disk segments corresponding to the CLUs (numbers within CLUs are illustrating pointers to the respective disk memory segments). **Fig. 11b** illustrates the new distribution of CLUs within the updated compressed file with unchanged size of the compressed sections as in the updated exemplary file described with reference to **Fig.5**. The virtual (logical) sequence of CLUs differs from the physical sequence of disk segments corresponding to the CLUs whilst maintaining de-fragmented structure of the file. **Fig. 11c** illustrates the de-fragmented distribution of CLUs within updated exemplary compressed file **50**, wherein the size of 2nd compressed section has been changed after an update from 432111 to 200100 bytes. If, for example, the update offset is 1MB + 314 bytes, the first compressed section is unaffected during the update. The new size of 2nd compressed section requires allocation of only 4 CLUs ([200100 / 61440] + 1). Note, as shown in Fig. 11B, that before the update the second compressed section accommodated 8 CLUs

(Nos. 2, 3,7,10 11, 12, 15 and 16). As described with reference to **Fig. 9**, the platform **20** will write the updated 2nd compressed section on first 4 CLUs from the compressed section (2, 3,7,10 in the present example) and send CLUs with pointers 11, 12, 15 and 16 to the list of free CLUs. 3rd and 4th compressed sections are also unaffected during this particular update. As described with reference to **Fig.10**, the platform **20** before closing the file will check if the list of free CLUs is empty. By this example the list contains CLUs with storage location pointers 11, 12, 15 and 16. As described with reference to **Fig. 10**, the platform will re-write compressed data from CLU with pointer 13 to CLU with pointer 11; compressed data from CLU with pointer 16 to CLU with pointer 12 and release CLUs with pointers 13-16. Thus the updated file has 12 allocated CLUs with no de-fragmentation.

Note that the invention is not bound by the specific inbound/outbound communication between the platform **20** and storage device 14 as described with reference to Figs. **6-11**. Those versed in the art will readily appreciate that the invention is, likewise, applicable to any other communication depending on network architecture and a manner of implementation of the platform **20** functions. For example, if compressed files are writing to and/or reading from internal platform resources, there is no need in generating of outbound requests. Likewise, there is no need in outbound requests if the platform functions are integrated with storage resources. Additional requirements (e.g. secure access, data integrity, etc.) may lead to more complicated communication between the platform and the file storage device as well as entire network resources.

It is also to be understood that the invention is not limited in its application to the details set forth in the description contained herein or illustrated in the drawings. The invention is capable of other embodiments and of being practiced and carried out in various ways. Hence, it is to be understood that the phraseology and terminology employed herein are for the purpose of description and should not be regarded as limiting. As such, those skilled in the art will appreciate that the conception upon which this disclosure is based may

readily be utilized as a basis for designing other structures, methods, and systems for carrying out the several purposes of the present invention.

Those skilled in the art will readily appreciate that various modifications and changes can be applied to the embodiments of the invention as hereinbefore described without departing from its scope, defined in and by the appended claims.

CLAIMS:

1. For use with a file access storage, a method of creating compressed file for storage, said method comprising:
   a) compressing a raw file and thereby generating compressed data, wherein at least one fixed-size portion of data (cluster) of the raw file is sequentially processed into corresponding compressed section divided into at least one fixed-size compression logical units (CLU);
   b) facilitating storing of the compressed data as a compressed file, the compressed file containing the compressed sections corresponding to the clusters of the raw file and a header comprising unique file descriptor;
   c) creating a section table comprising at least one record describing a compressed section, said record holding at least information on CLUs corresponding to the compressed section and storage location pointers pertaining to said CLUs.
2. The method of Claim 1 wherein the clusters have pre-defined equal size.
3. The method of Claim 1 wherein a size of a cluster is selected from a list of predefined sizes in accordance with type of data comprised in the cluster.
4. The method of Claim 1 wherein a cluster is processed with a compression algorithm in accordance with type of data comprised in the cluster.
5. The method of Claim 1 wherein the compressed file contains at least one section accommodating non compressed data of the corresponding cluster.
6. The method of Claim 1 wherein the processing of a cluster includes assessment of an obtainable compression ratio and generating the compressed data within the compressed section only if said compression ratio meets a predefined criterion.
7. The method of Claim 1 wherein the header of the compressed file holds information indicating whether at least one cluster was processed into compressed section.

8. The method of Claim 1 wherein the header of the compressed file holds information on a size of the raw file.

9. The method of Claim 1 wherein the records describing the compressed sections have equal pre-defined size.

10. The method of Claim 1 wherein a record of a compressed section holds information indicating if the compressed section contains compressed data.

11. The method of Claim 1 wherein compressing of a file is providing only if the file fits predefined criteria.

12. The method of Claim 1 wherein the section table forms a part of the compressed file.

13. The method of Claim 12 wherein the section table is stored at the position following the last compressed section contained in the compressed file.

14. The method of Claim 1 wherein the section table forms a part of a file other than pertaining compressed file.

15. The method of Claim 1 wherein the size of the last CLU in the last compressed section is defined by actual size of compressed data stored in said CLU.

16. A method of reading data stored in a file compressed in accordance with Claim 1, said method comprising:

a) determining a serial number of first compressed section comprising data to be read;

b) determining the CLUs corresponding to said compressed section and storage location thereof by referring to the section table;

c) facilitating restoring the cluster from said compressed section.

17. The Method of Claim 16 further comprising repeating the stages b) and c) for compressed sections with serial numbers incremented by 1 if the range of data to be read exceeds the size of the restored clusters, until all data to be read are restored.

18. A method of writing data at a given data range to a file compressed in accordance with Claim 1, said method comprising:

a) determining a serial number of first compressed section comprising data to be updated constituting the original compressed section;

b) determining the CLUs corresponding to said original compressed section and storage location thereof by referring to the section table;

c) facilitating restoring the cluster from said original compressed section;

d) calculating an offset of the updating data within said cluster and facilitating the update at the given data range;

e) compressing the updated cluster into an updated compressed section;

f) facilitating overwriting of said original compressed section with updated compressed section;

g) updating the section table.

19. The method of Claim 18 further comprising repeating stages b) to g) for compressed sections with serial numbers incremented by 1 if the range of data to be write exceeds the size of the restored clusters, until all required data are written.

20. The method of Claim 18 further comprising handling a list of free CLUs released during writing data to the compressed file, said list is handled during all sessions related to the file until the file is closed.

21. The method of Claim 20 further comprising comparing the number of CLUs required to said original N_o and said updated N_u compressed sections and facilitating one of the following:

a) overwriting all CLUs corresponding to the original compressed section with CLUs corresponding to the updated compressed section if $N_o=N_u$;

b) overwriting first N_u CLUs corresponding to the original compressed section with CLUs corresponding to the updated compressed section and updating the list of free CLUs about released CLUs if $N_o>N_u$;

c) overwriting all CLUs corresponding to the original compressed section with CLUs corresponding to the updated compressed section and writing the rest of CLUs corresponding to the updated compressed section to the

CLUs contained in the list of free CLUs, if $N_o<N_u< N_o+N_f$, where N_f is a number of CLUs in said list;

d) overwriting all CLUs corresponding to the original compressed section with CLUs corresponding to the updated compressed section, writing the CLUs corresponding to the updated compressed section to the CLUs contained in the list of free CLUs, and continuous writing the rest of CLUs corresponding to the updated compressed section to next free storage location if $N_o+N_f <N_u$.

22. The method of Claim 20 further comprising:

a) checking the list of free CLUs before closing the file;

b) if said list is not empty defining a CLU with the highest storage location pointer among CLUs comprised in the compressed sections (first CLU);

c) facilitating moving the compressed data from said first CLU to a free CLU with lower storage location pointer (second CLU);

d) assigning said second CLU to pertaining compressed section and said first CLU to the list of free CLUs;

e) repeating the stages b)-d) until the storage location pointers of all CLUs comprised in compressed sections are lower than a pointer of any of CLU comprising in the list of free CLUs;

f) updating the section table.

23. For use with a file access storage, a method of reading a raw file stored as compressed data, wherein said compressed data are packed into one or more compressed sections corresponding to fixed-size portions of data (clusters) of the raw file, said compressed sections divided into fixed-size compression logical units (CLU), the method comprising:

a) determining a serial number of first compressed section comprising data to be read;

b) determining the storage location of the compressed data to be read by referring to a section table which holds information on CLUs corresponding to said compressed section and storage location thereof;

c) facilitating restoring the cluster from said compressed section.

24. The Method of Claim 23 further comprising repeating the stages b) and c) for next compression sections with serial numbers incremented by 1 if the range of data to be read exceeds the size of the restored clusters, until all data to be read are restored.

25. For use with a file access storage, a method of writing data at a given data range to a file stored as compressed data, wherein said compressed data are packed into one or more compressed sections corresponding to fixed-size portions of data (clusters) from the raw file, said compressed sections divided into fixed-size compression logical units (CLU), the method comprising:

a) determining a serial number of first compressed section comprising data to be updated constituting the original compressed section;

b) determining the storage location of the compressed data to be updated by referring to a section table which holds information on CLUs corresponding to said compressed section and storage location thereof;

c) facilitating restoring the cluster from said original compressed section ;

d) calculating the offset of updating data within said cluster and facilitating the update of the given data range;

e) compressing the updated cluster into an updated compressed section;

f) facilitating overwriting of said original compressed section with updated compressed section;

g) updating the section table.

26. The method of Claim 25 further comprising repeating stages b) to g) for next compressed sections with serial numbers incremented by 1 if the range of data to be written exceeds the size of the restored clusters, until all required data are written.

27. The method of Claim 25 further comprising handling a list of free CLUs released during writing data to the compressed file, said list is handled during all sessions related to the file until the file is closed.

28. The method of Claim 27 further comprising comparing the number of CLUs required to said original N_o and updated N_u compressed section and facilitating one of the following:

a) overwriting all CLUs corresponding to the original compressed section with CLUs corresponding to the updated compressed section if $N_o=N_u$;

b) overwriting first N_u CLUs corresponding to the original compressed section with CLUs corresponding to the updated compressed section and updating the list of free CLUs about released CLUs if $N_o>N_u$;

c) overwriting all CLUs corresponding to the original compressed section with CLUs corresponding to the updated compressed section and writing the rest of CLUs corresponding to the updated compressed section to the CLUs contained in the list of free CLUs, if $N_o<N_u< N_o+N_f$, where N_f is a number of CLUs in said list;

d) overwriting all CLUs corresponding to the original compressed section with CLUs corresponding to the updated compressed section, writing the CLUs corresponding to the updated compressed section to the CLUs contained in the list of free CLUs, and continuous writing the rest of CLUs corresponding to the updated compressed section to next free storage location if $N_o+N_f<N_u$.

29. The method of Claim 27 further comprising:

a) checking the list of free CLUs before closing the file;

b) if said list is not empty, defining a CLU with the highest storage location pointer among CLUs comprised in the compressed sections (first CLU);

c) facilitating moving the compressed data from said first CLU to a free CLU with lower storage location pointer (second CLU);

d) assigning said second CLU to pertaining compressed section and said first CLU to the list of free CLUs;

e) repeating the stages b)-d) until the storage location pointers of all CLUs comprised in compressed sections are lower than a pointer of any of CLU comprising in the list of free CLUs;

f) updating the section table.

30. The method of Claim 1 further comprising maintaining de-fragmented structure of the compressed file.

31. The method of Claim 23 further comprising maintaining de-fragmented structure of the compressed file.

32. The method of Claim 25 further comprising maintaining de-fragmented structure of the compressed file.

33. Method of Claim 1 wherein the file access storage is network attached storage (NAS).

34. Method of Claim 23 wherein the file access storage is network attached storage (NAS).

35. Method of Claim 25 wherein the file access storage is network attached storage (NAS).

36. Method of Claim 1 further comprising encryption of data contained in the raw file.

37. A system for compressing files for storage comprising:

a) subsystem for compressing a raw file and thereby generating the compressed data, wherein a fixed-size portion of data (cluster) of the raw file is sequentially processing into a compressed section divided into at least one fixed-size compression logical unit (CLU);

b) subsystem for facilitating storing of the compressed data as a compressed file, the compressed file containing the compressed sections corresponding to the clusters of the raw file and a header comprising unique file descriptor;

c) subsystem for creating a section table comprising at least one record of a compressed section, said record holding information on CLUs corresponding to the compressed section and storage location pointers pertaining to said CLUs.

38. For use with a file access storage, a program storage device readable by machine, tangibly embodying a program of instructions executable by the

machine to perform method steps of creating compressed file for storage, said method comprising:

a) compressing a raw file and thereby generating the compressed data, wherein at least one fixed-size portion of data (cluster) of the raw file is sequentially processed into corresponding compressed section divided into at least one fixed-size compression logical units (CLU);

b) facilitating storing of the compressed data as a compressed file, the compressed file containing the compressed sections corresponding to the clusters of the raw file and a header comprising unique file descriptor;

c) creating a section table comprising records of compressed sections, said records holding information on CLUs corresponding to the compressed section and storage location pointers pertaining to said CLUs.

39. For use with a file access storage, a computer program product comprising a computer useable medium having computer readable program code embodied therein of creating compressed file for storage, said computer program product comprising:

a) computer readable program code for causing the computer to compress a raw file and thereby generating the compressed data, wherein at least one fixed-size portion of data (cluster) of the raw file is sequentially processed into corresponding compressed section divided into at least one fixed-size compression logical units (CLU);

b) computer readable program code for causing the computer to facilitate storing of the compressed data as a compressed file, the compressed file containing the compressed sections corresponding to the clusters of the raw file and a header comprising unique file descriptor;

c) computer readable program code for causing the computer to create a section table comprising records of compressed sections, said records holding information on CLUs corresponding to the compressed section and storage location pointers pertaining to said CLUs.

40. A method of compressing a file for storage, said method facilitating direct access to the compressed data whilst maintaining de-fragmented structure of the compressed file.

41. For use with file access storage, a method of writing data to a file stored as compressed data, wherein the compressed data of a raw file are packed into one or more compressed units, said method facilitating update of one or more corresponding compressed unit with no need of restoring the entire file whilst maintaining de-fragmented structure of the compressed units.

Prior Art

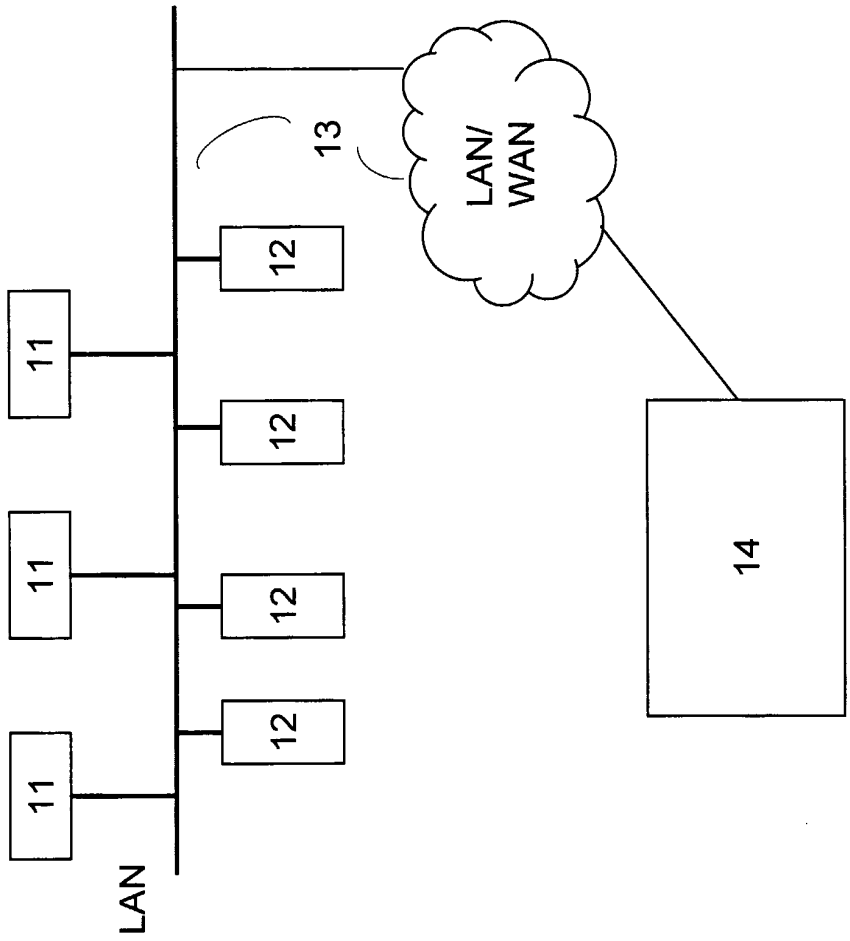
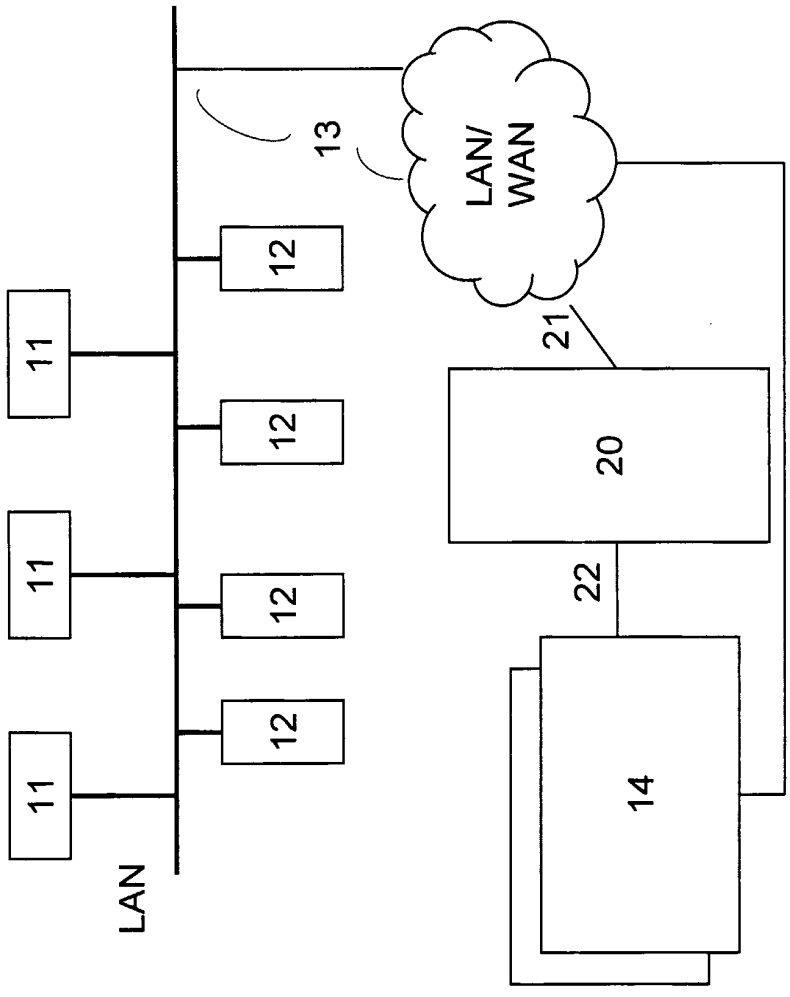


Figure 1

Figure 2a



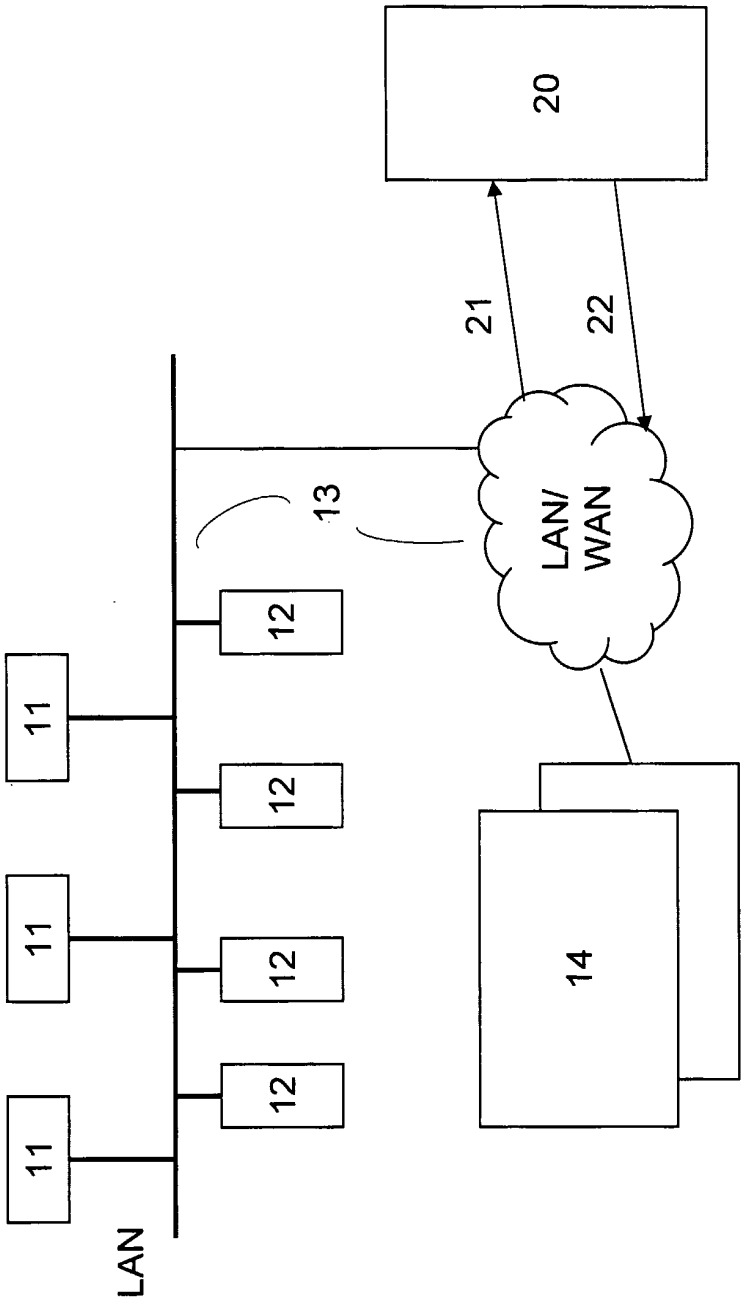


Figure 2b

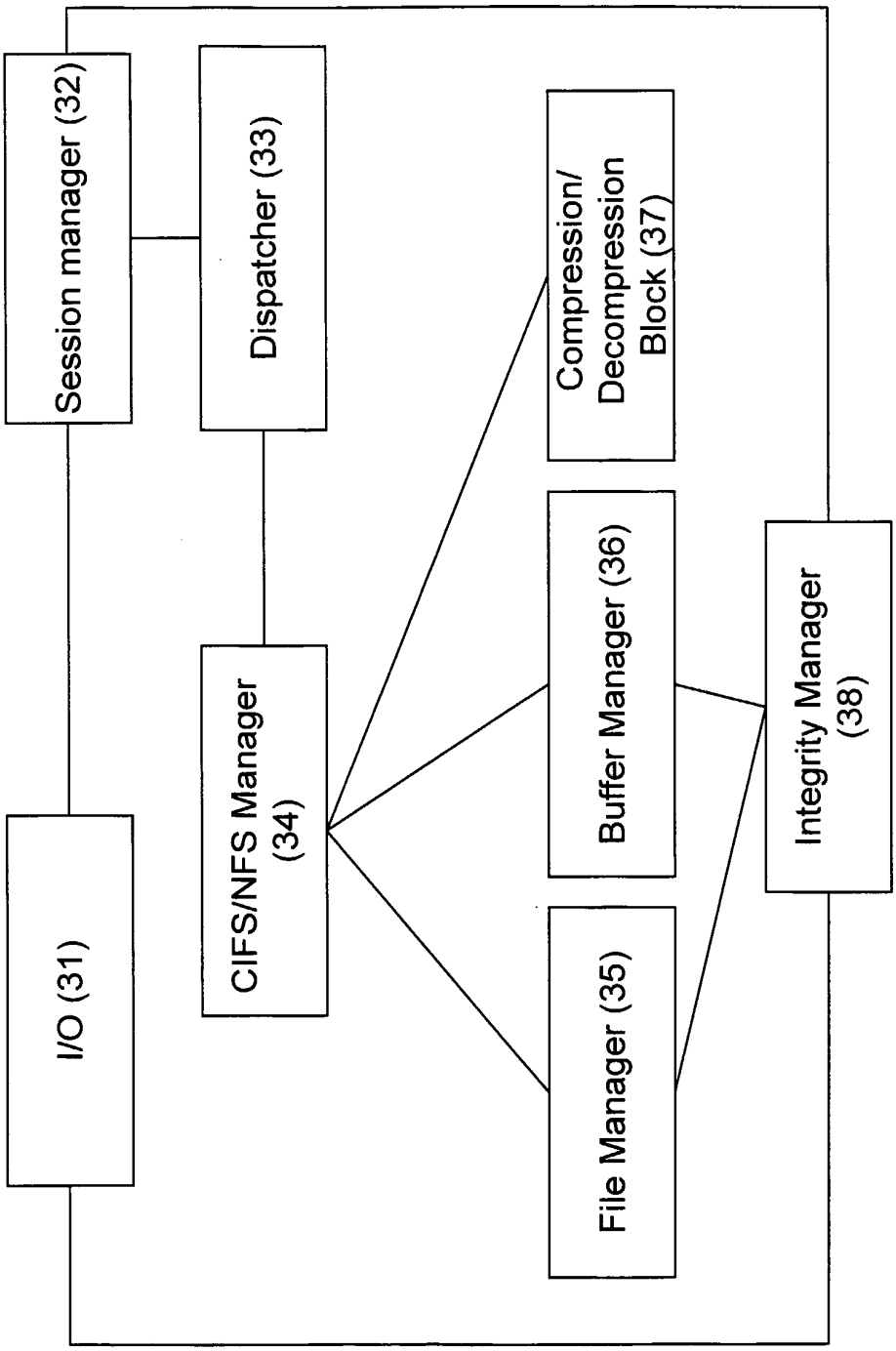

Session manager (32)
Dispatcher (33)
Compression/ Decompression Block (37)
Buffer Manager (36)
Integrity Manager (38)
I/O (31)
CIFS/NFS Manager (34)
File Manager (35)

Figure 3

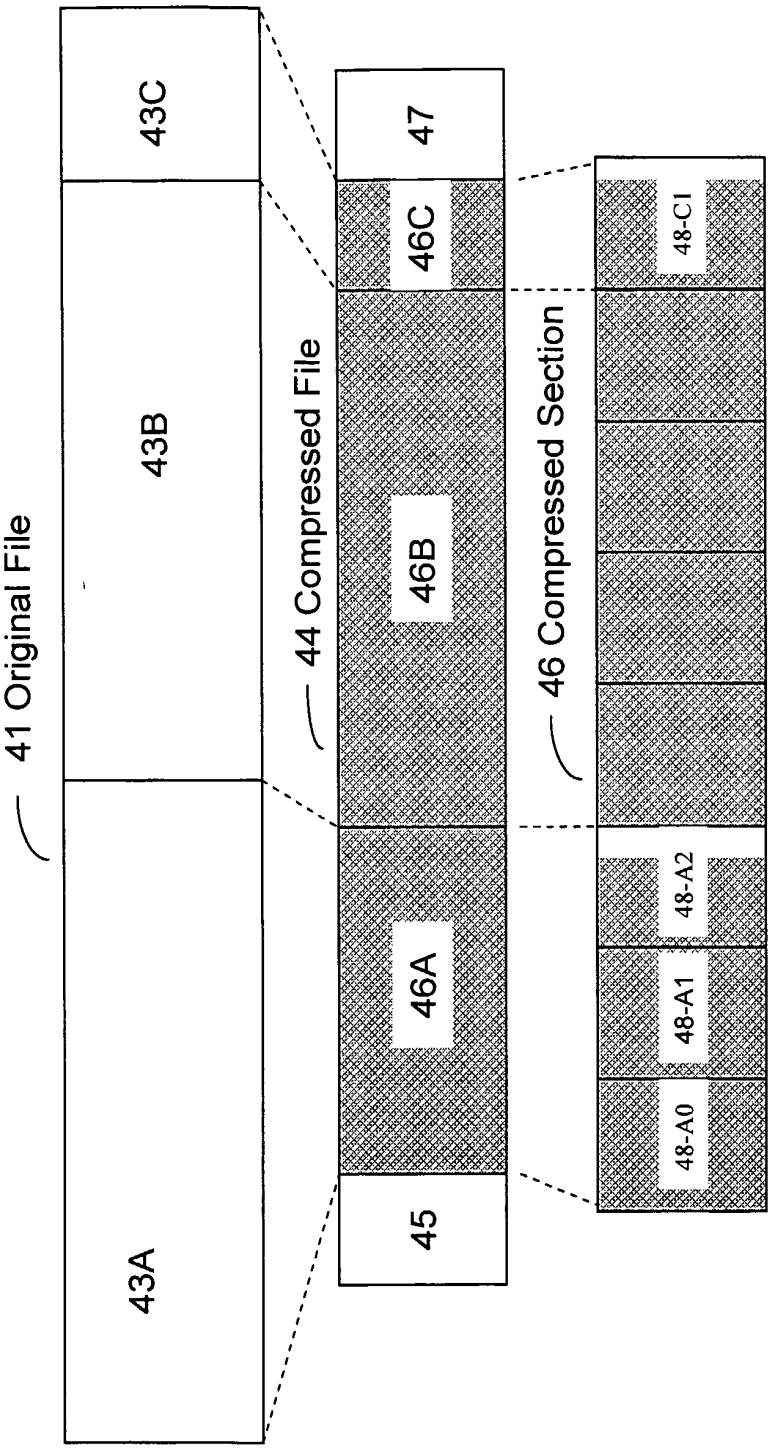

41 Original File
43A
43B
43C
44 Compressed File
45
46A
46B
46C
47
46 Compressed Section
48-A0
48-A1
48-A2
48-C1

Figure 4

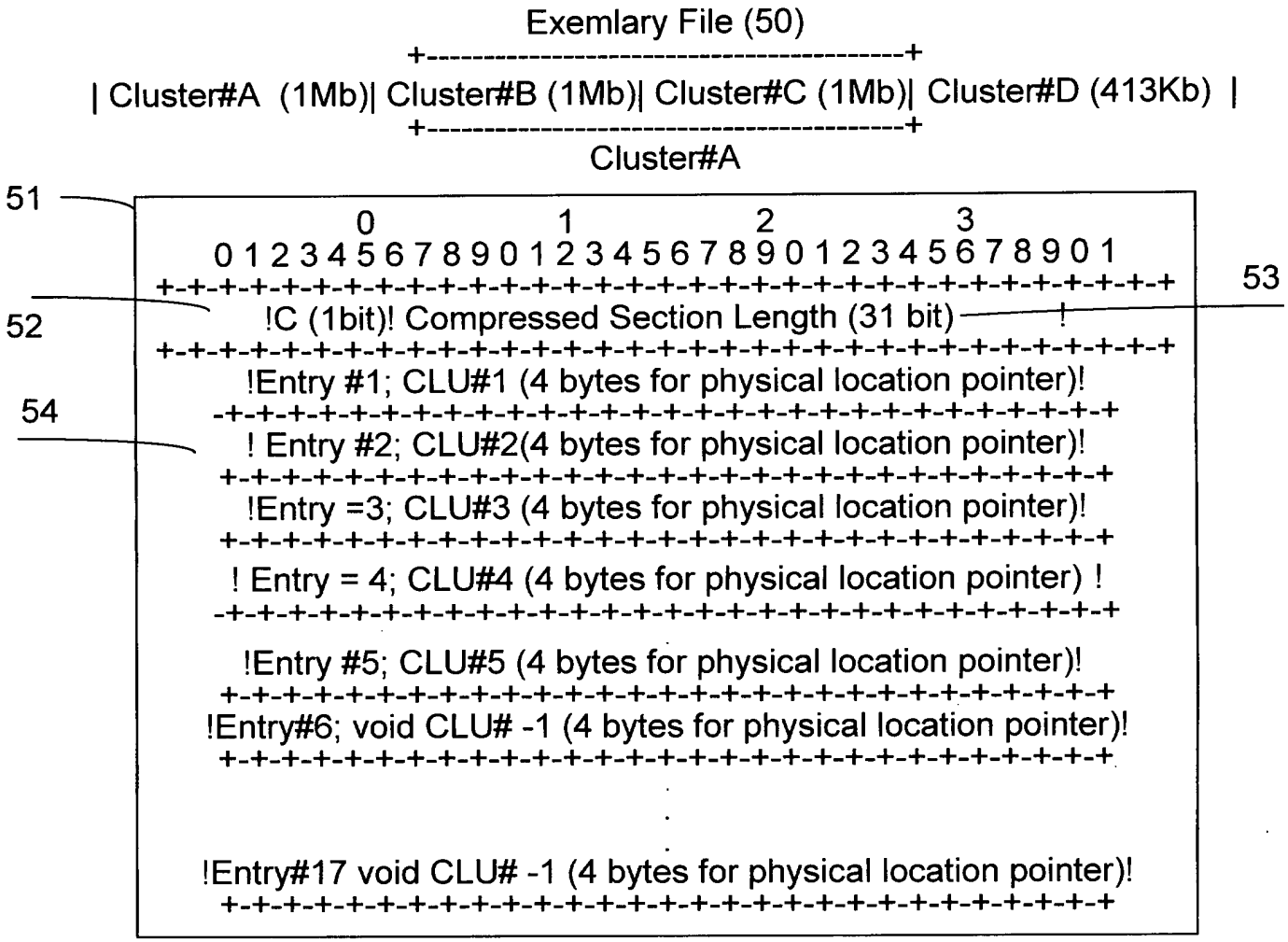

Exemlary File (50)
| Cluster#A (1Mb)| Cluster#B (1Mb)| Cluster#C (1Mb)| Cluster#D (413Kb) |
Cluster#A
51
52
53
54
0 1 2 3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
!C (1bit)! Compressed Section Length (31 bit)
!Entry #1; CLU#1 (4 bytes for physical location pointer)!
! Entry #2; CLU#2(4 bytes for physical location pointer)!
!Entry =3; CLU#3 (4 bytes for physical location pointer)!
! Entry = 4; CLU#4 (4 bytes for physical location pointer) !
!Entry #5; CLU#5 (4 bytes for physical location pointer)!
!Entry#6; void CLU# -1 (4 bytes for physical location pointer)!
!Entry#17 void CLU# -1 (4 bytes for physical location pointer)!

Figure 5

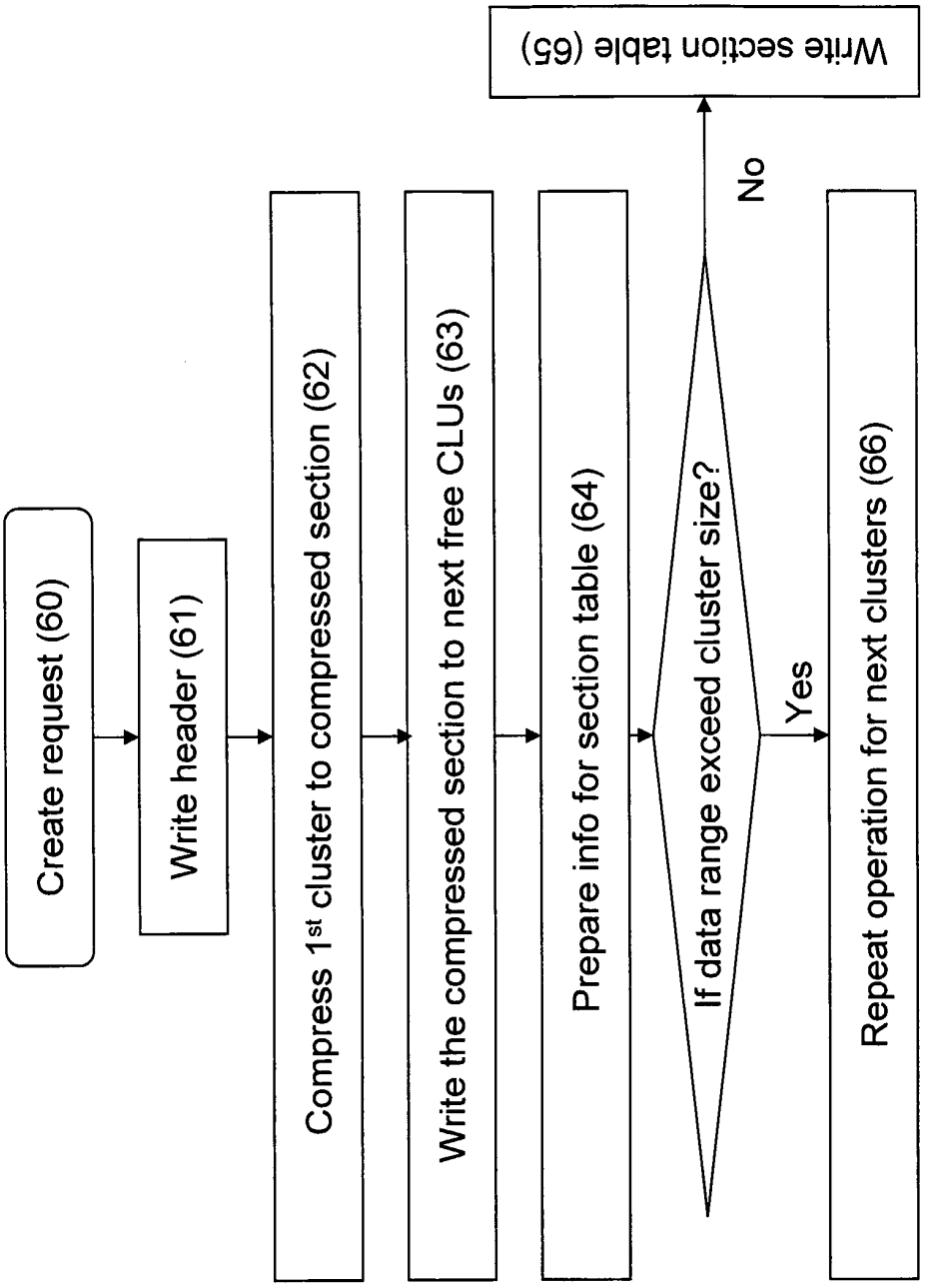

Create request (60)
Write header (61)
Compress 1st cluster to compressed section (62)
Write the compressed section to next free CLUs (63)
Prepare info for section table (64)
If data range exceed cluster size?
No
Yes
Write section table (65)
Repeat operation for next clusters (66)

Figure 6

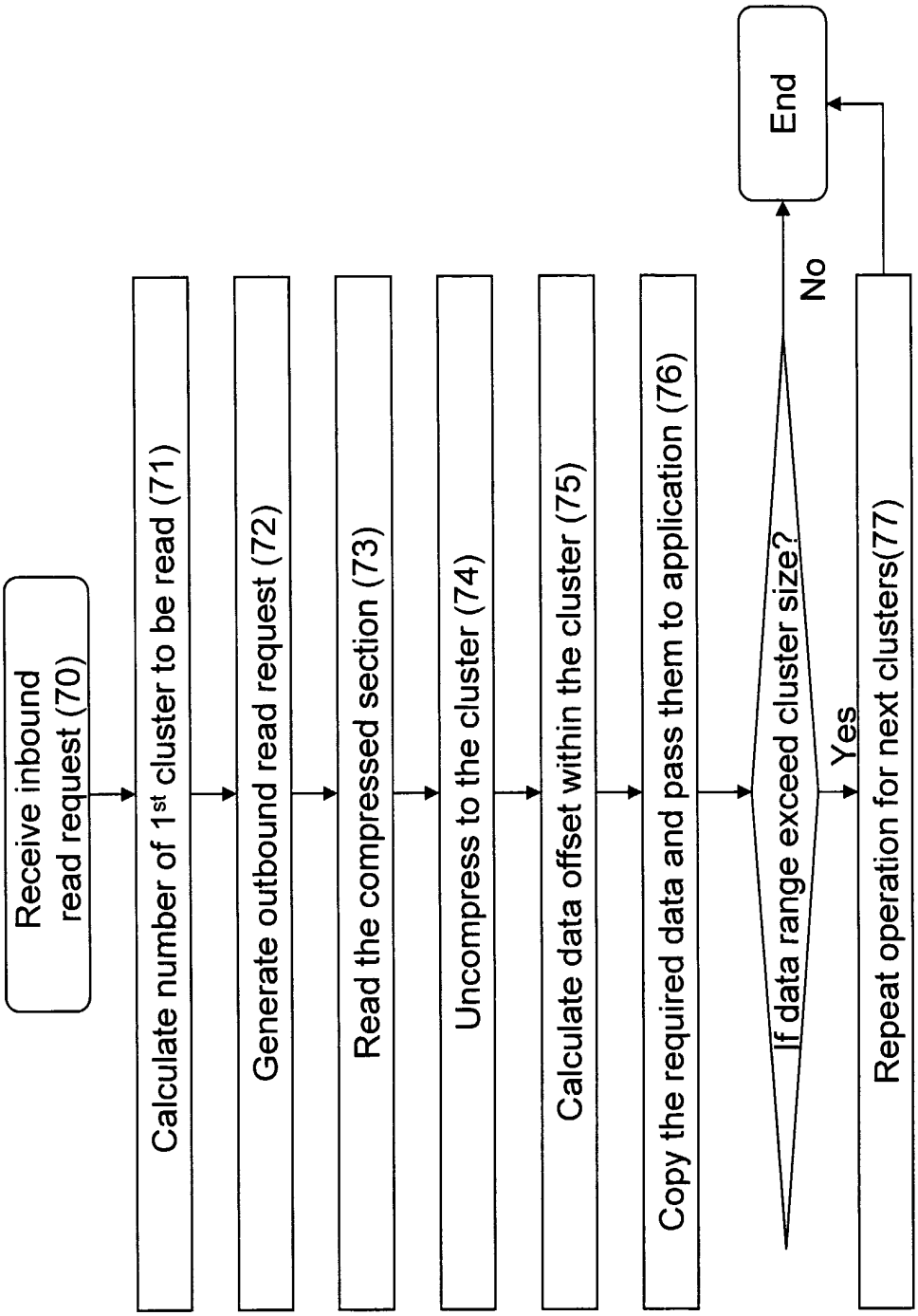

Receive inbound read request (70)
Calculate number of 1st cluster to be read (71)
Generate outbound read request (72)
Read the compressed section (73)
Uncompress to the cluster (74)
Calculate data offset within the cluster (75)
Copy the required data and pass them to application (76)
If data range exceed cluster size?
No
Yes
End
Repeat operation for next clusters(77)

Figure 7

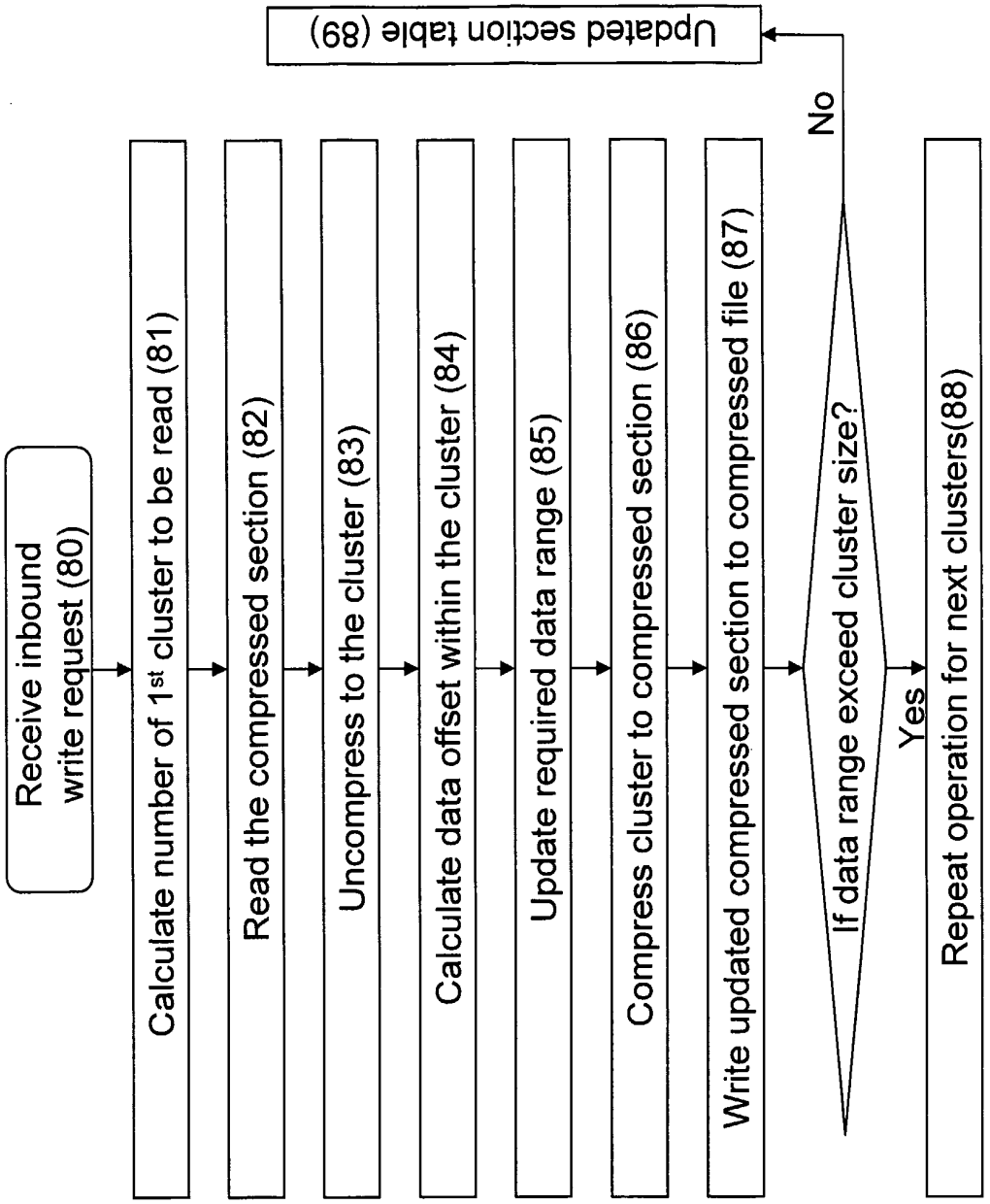

Receive inbound write request (80)
Calculate number of 1st cluster to be read (81)
Read the compressed section (82)
Uncompress to the cluster (83)
Calculate data offset within the cluster (84)
Update required data range (85)
Compress cluster to compressed section (86)
Write updated compressed section to compressed file (87)
If data range exceed cluster size?
No
Yes
Updated section table (89)
Repeat operation for next clusters(88)

Figure 8

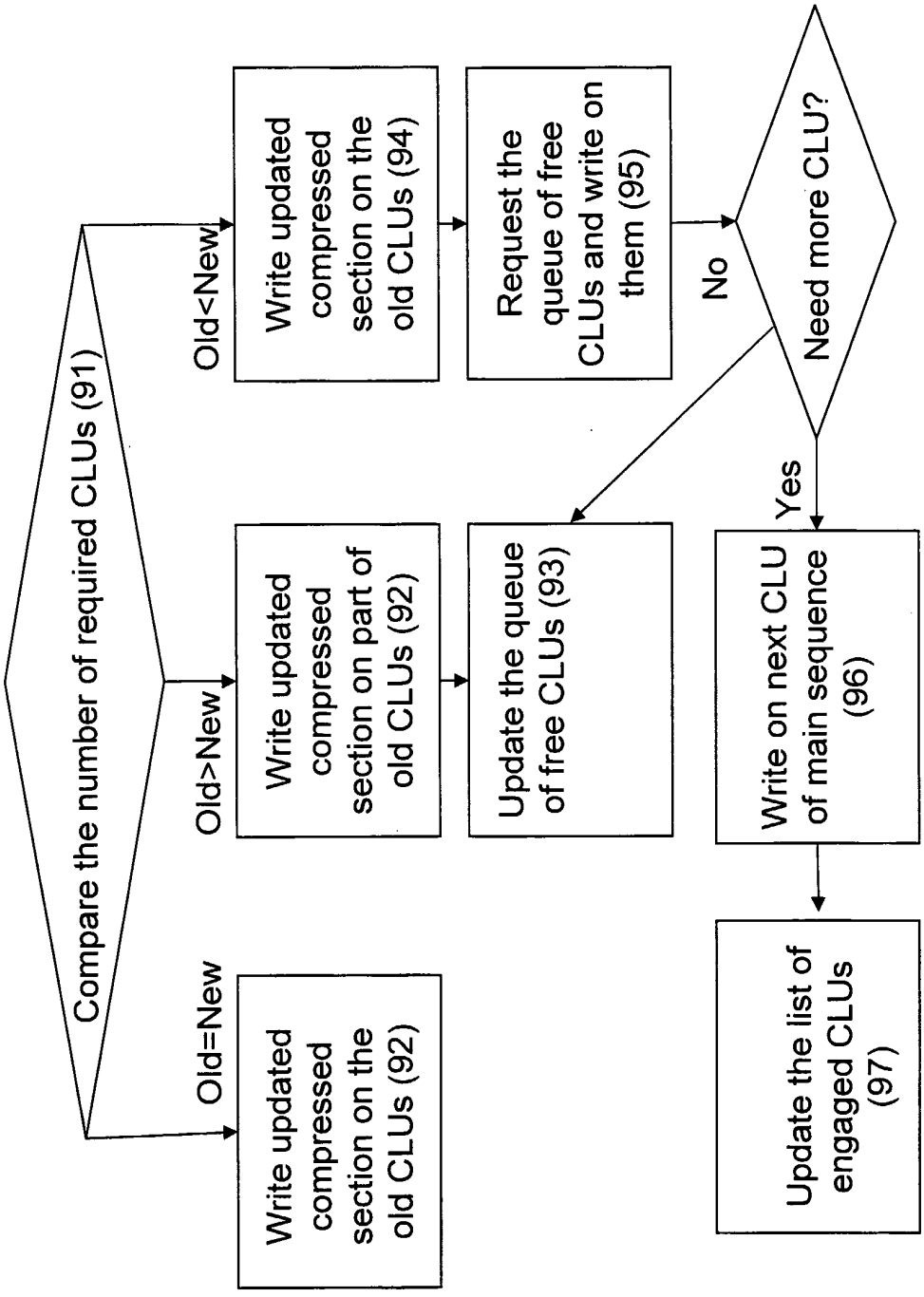

Compare the number of required CLUs (91)
Old=New
Write updated compressed section on the old CLUs (92)
Old>New
Write updated compressed section on part of old CLUs (92)
Update the queue of free CLUs (93)
Old<New
Write updated compressed section on the old CLUs (94)
Request the queue of free CLUs and write on them (95)
No
Need more CLU?
Yes
Write on next CLU of main sequence (96)
Update the list of engaged CLUs (97)

Figure 9

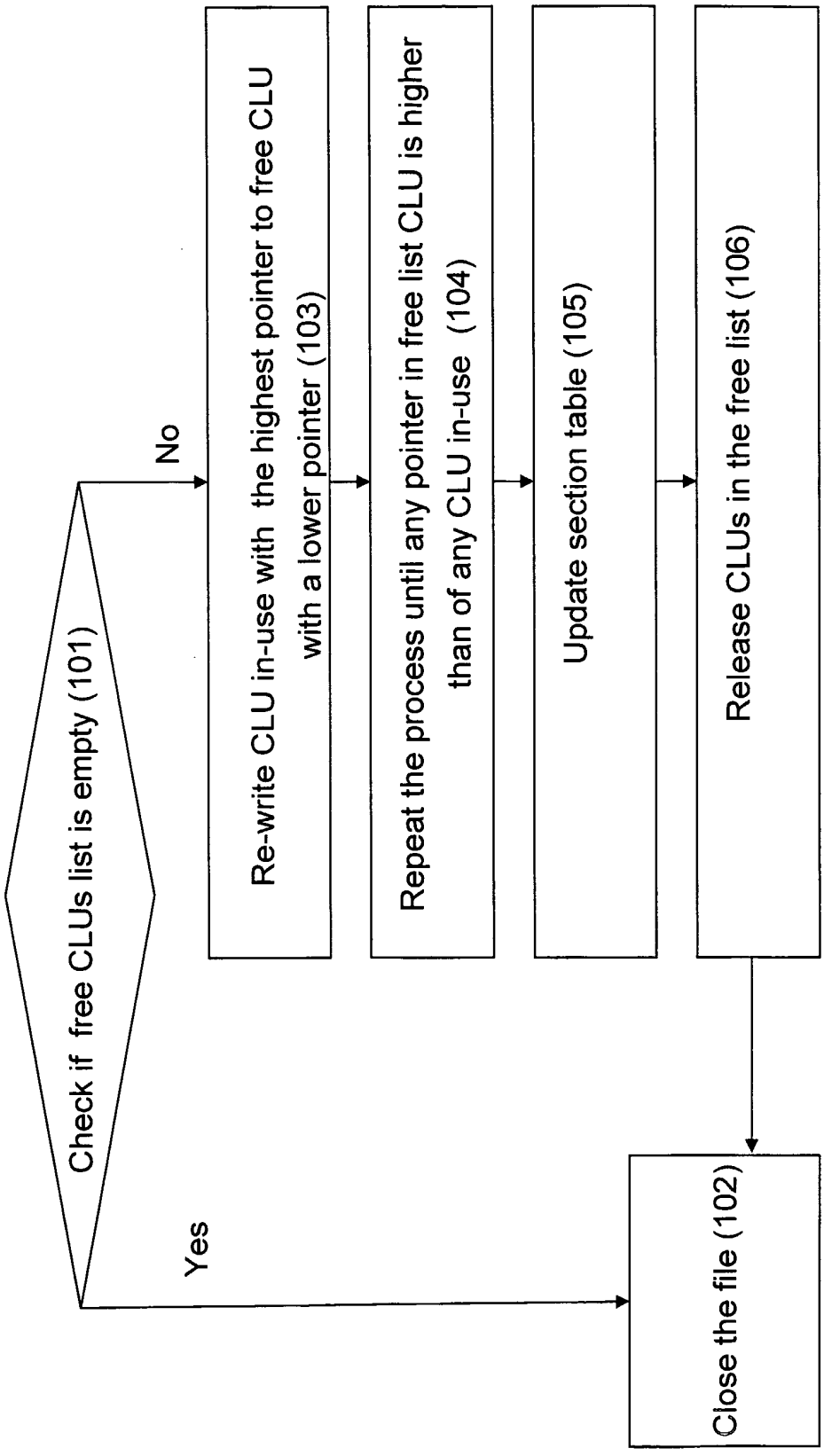

Check if free CLUs list is empty (101)
No
Yes
Re-write CLU in-use with the highest pointer to free CLU with a lower pointer (103)
Repeat the process until any pointer in free list CLU is higher than of any CLU in-use (104)
Update section table (105)
Release CLUs in the free list (106)
Close the file (102)

Figure 10

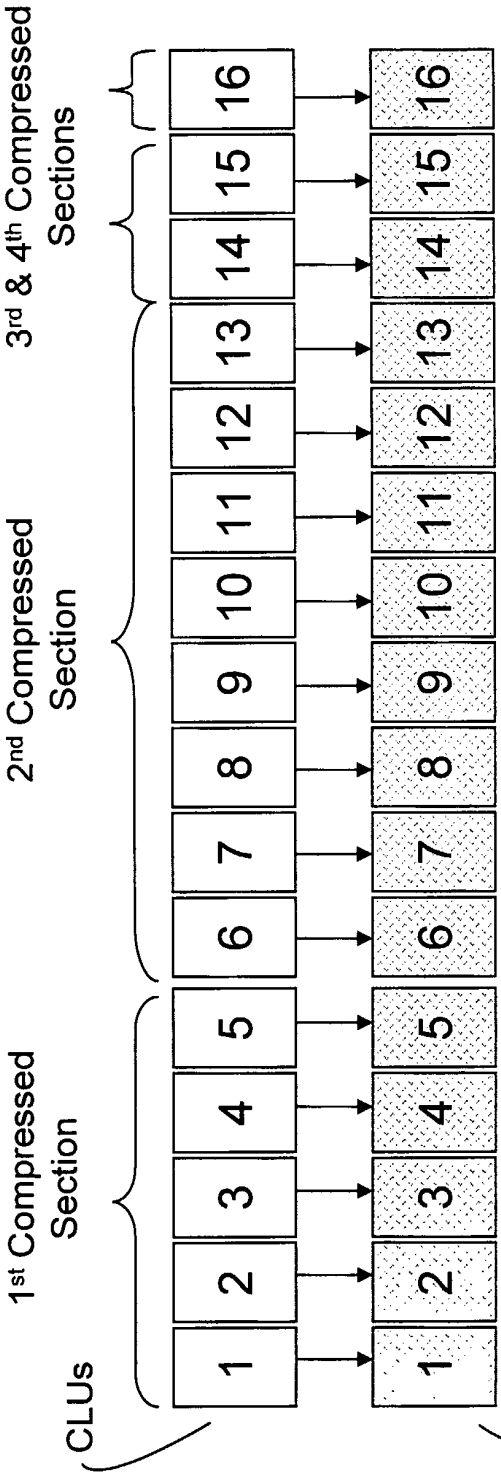

1st Compressed Section
2nd Compressed Section
3rd & 4th Compressed Sections
CLUs
Disk memory segments
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16

Figure 11a

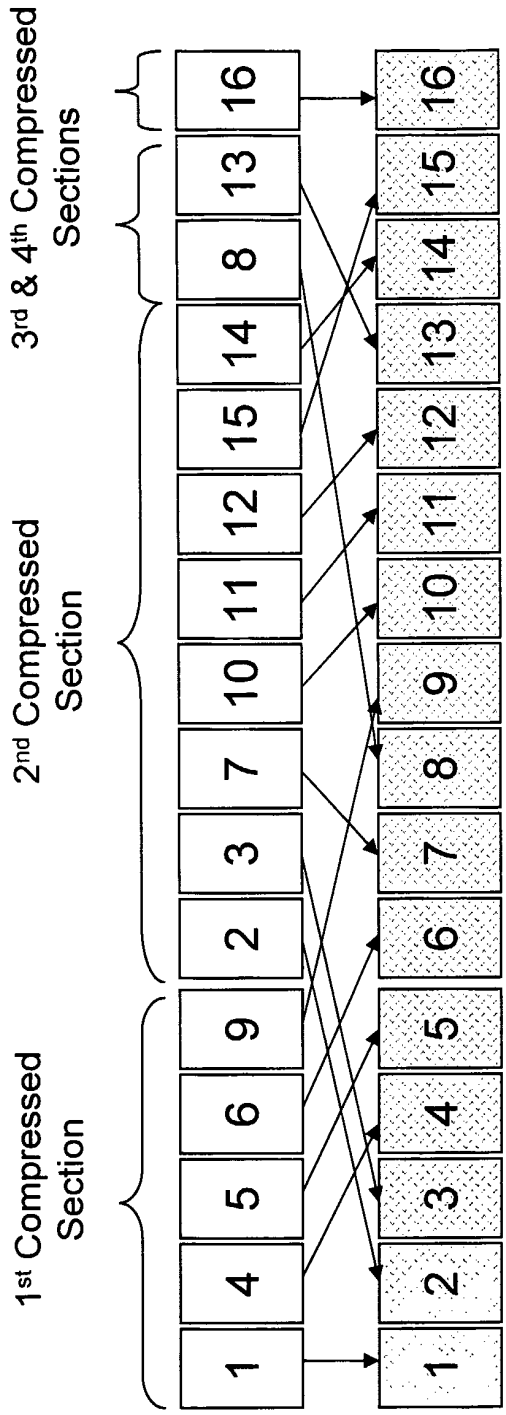


Figure 11b

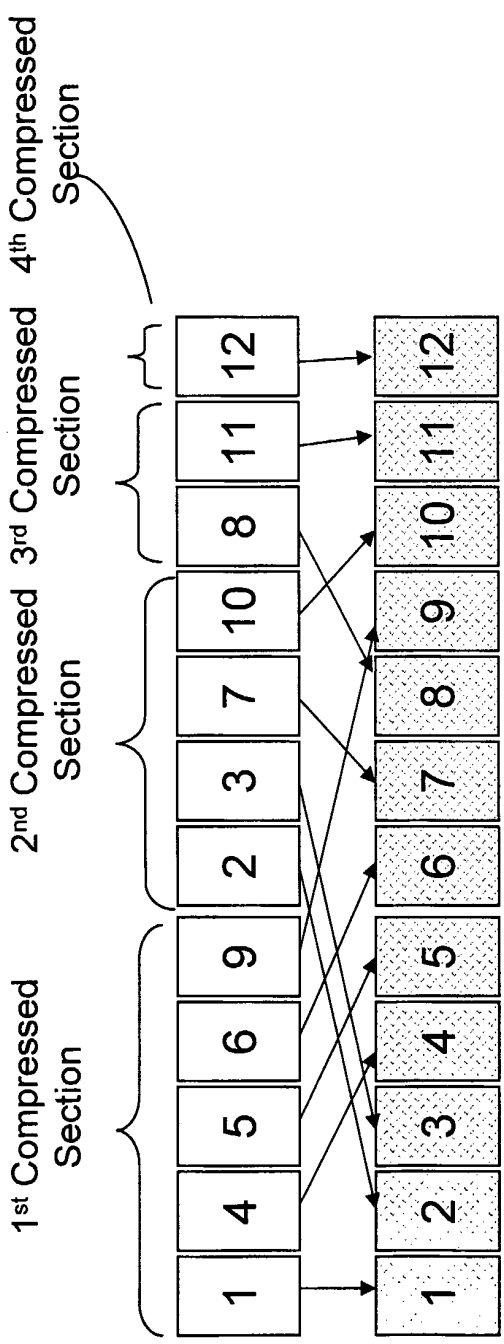


Figure 11c