



Office de la Propriété

Intellectuelle  
du Canada

Un organisme  
d'Industrie Canada

Canadian  
Intellectual Property  
Office

An agency of  
Industry Canada

CA 2786613 A1 2011/07/21

(21) **2 786 613**

(12) **DEMANDE DE BREVET CANADIEN**  
**CANADIAN PATENT APPLICATION**

(13) **A1**

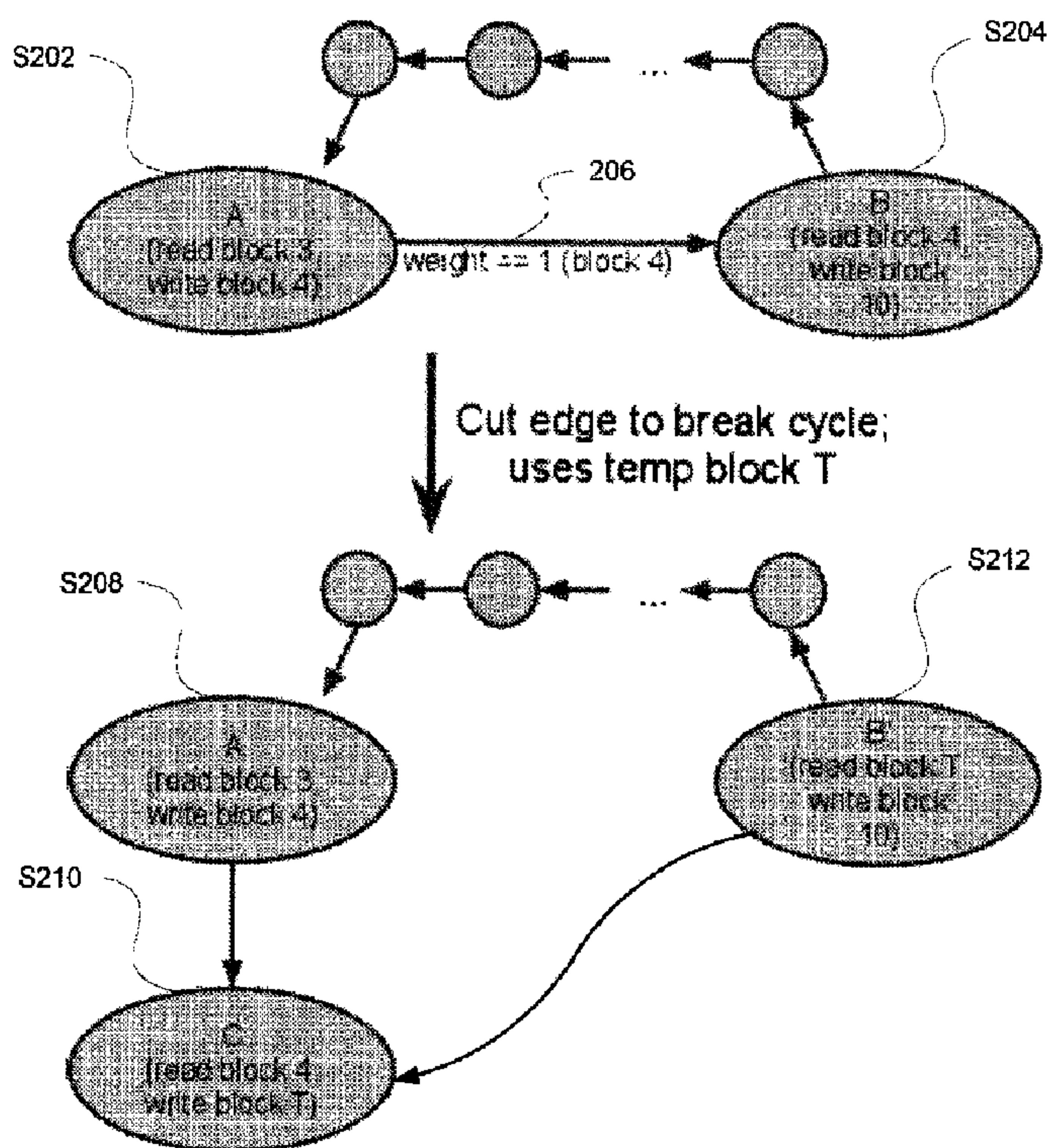
(86) Date de dépôt PCT/PCT Filing Date: 2011/01/11  
(87) Date publication PCT/PCT Publication Date: 2011/07/21  
(85) Entrée phase nationale/National Entry: 2012/07/06  
(86) N° demande PCT/PCT Application No.: US 2011/020790  
(87) N° publication PCT/PCT Publication No.: 2011/088022  
(30) Priorité/Priority: 2010/01/12 (US61/294,266)

(51) Cl.Int./Int.Cl. *G06F 9/44* (2006.01),  
*G06F 15/16* (2006.01)  
(71) Demandeur/Applicant:  
GOOGLE INC., US  
(72) Inventeur/Inventor:  
DE LOS REYES, ANDREW, US  
(74) Agent: MOFFAT & CO.

(54) Titre : PROCEDE DE MISE A JOUR AUTOMATIQUE D'UN SYSTEME D'EXPLOITATION  
(54) Title: OPERATING SYSTEM AUTO-UPDATE PROCEDURE

FIG. 2

200



(57) Abrégé/Abstract:

The present invention pertains to differential updating of an operating system in a client device (302, 306, 308, 310, 312). A delta update file (100) includes an ordered list of operations to be performed on the new install partition in memory (326) that will port

**(57) Abrégé(suite)/Abstract(continued):**

from the old install partition. A binary differential compression algorithm may be used to determine differences for the update and to compress the data being transmitted to the client device. Blocks in the partition may cyclically depend from one another (S202, S204). Edges are cut to break cycles. During system operation, disk images are scanned (1 in FIG. 3). File operations are created and a blocks vector is developed (2 in FIG. 3). A graph is created (3 in FIG. 3) where the edge weights are set equal to the number of blocks. Given this, cycles are then broken (4 in FIG. 3) and a final order resulting from a topological sort is produced (5 in FIG. 3).

## ABSTRACT

The present invention pertains to differential updating of an operating system in a client device (302, 306, 308, 310, 312). A delta update file (100) includes an ordered list of operations to be performed on the new install partition in memory (326) that will port from the old install partition. A binary differential compression algorithm may be used to determine differences for the update and to compress the data being transmitted to the client device. Blocks in the partition may cyclically depend from one another (S202, S204). Edges are cut to break cycles. During system operation, disk images are scanned (1 in FIG. 3). File operations are created and a blocks vector is developed (2 in FIG. 3). A graph is created (3 in FIG. 3) where the edge weights are set equal to the number of blocks. Given this, cycles are then broken (4 in FIG. 3) and a final order resulting from a topological sort is produced (5 in FIG. 3).

## OPERATING SYSTEM AUTO-UPDATE PROCEDURE

## CROSS-REFERENCE TO RELATED APPLICATION

[0001] The present application claims priority to United States Provisional Application No. 61/294,266, filed January 12, 2010, the entire disclosure of which is hereby incorporated by reference.

## BACKGROUND OF THE INVENTION

[0002] Software packages such as operating systems may be updated from time to time to introduce new features, correct errors and address security flaws. Due to file sizes for large applications, it may be inconvenient or inefficient to send and install an entirely new package with an update. One solution is to send a differential update to the client, which covers only the specific changes from the prior software. If the differential update is not performed correctly, the modified software may operate poorly or not at all.

## SUMMARY OF THE INVENTION

[0003] The present invention relates generally to operating systems. More particularly, the present invention relates to updating the version of an operating system.

[0004] In accordance with one embodiment, a method of generating an update for a computer readable operating system is provided. The method comprises identifying a version number of a current version of the operating system; creating, with a processor, an ordered list of operations for updating the current version of the operating system to a new version of the operating system, the processor performing iterations over each regular file on the new version of the operating system to obtain the ordered list for all data blocks associated with the new version; and assembling, with the processor, a differential update file including a magic number indicator showing the differential update file is an actual update file, a new version number identifying the new version

of the operating system, and a protocol buffer including the ordered list of operations.

**[0005]** In one example, one or more operations in the ordered list are each associated with a respective data blob indicating a chunk of data. Here, the differential update file is assembled to include the respective data blobs. In another example, each operation in the ordered list is applicable to one or more specific data blocks of a partition of a client device. The operations include one or more of: a copy operation where at least one of the data blocks in the partition is to be copied to another block in a new partition of the client device for the new version of the operating system; a difference operation where at least a given one of the data blocks is read into memory and a difference routine is performing on the at least one given data block using a data blob of the differential update file; a replace operation where a selected data blob of the differential update file is configured to be written to specified blocks in the new partition; and a replace with uncompression operation where a compressed data blob is included in the differential update file and is configured to be written to selected specified blocks in the new partition for the new version of the operating system.

**[0006]** In one alternative, each operation in the ordered list is associated with a file object, and the method further comprises creating a vertex in a graph for each file object; and creating a vector representing each block. In one variation, the method further comprises setting reader and writer parameters for the vector of each block; and for each block with a different reader and writer, creating an edge in the graph from the writer to the reader. The edge points to a file operation to be completed before a source file operation associated with the edge starts. In an example, each edge has a weight, and the weight identifies a number of blocks in the graph associated with that edge. In another example, if

the graph includes cycles, the method further comprises breaking each of the cycles. In this case, breaking a given one of the cycles may include finding a lowest-weight edge associated with the given cycle; and cutting the lowest-weight edge. Here, cutting the lowest-weight edge may comprise creating a new node representing an operation of copying an extent to a scratch space; and making the lowest-weight edge's source node point to the new node. And a new edge may be made from a destination node of the lowest-weight edge in order to ensure that a new copy operation occurs before a consumer of the new copy operation.

**[0007]** In accordance with another embodiment of the present invention, a device for generating an update for a computer readable operating system is provided. The device comprises memory for storing differential update information associated with the operating system, and a processor coupled to the memory. The processor is configured to identify a version number of a current version of the operating system; create, using the differential update information, an ordered list of operations for updating the current version of the operating system to a new version of the operating system, including performing iterations over each regular file on the new version of the operating system to obtain the ordered list for all data blocks associated with the new version; assemble a differential update file including a magic number indicator showing the differential update file is an actual update file, a new version number identifying the new version of the operating system, and a protocol buffer including the ordered list of operations; and store the differential update file in the memory.

**[0008]** In one example, each operation in the ordered list is associated with a file object, and the processor is further configured to create a vertex in a graph for each file object, and create a vector representing each block. In an alternative, the processor is further configured to set reader

and writer parameters for the vector of each block; and for each block with a different reader and writer, create an edge in the graph from the writer to the reader. The edge points to a file operation to be completed before a source file operation associated with the edge starts. In this case, if the graph includes cycles, the processor is optionally operable to break each of the cycles. Here, breaking a given one of the cycles may include finding a lowest-weight edge associated with the given cycle, and cutting the lowest-weight edge. In this case, the processor may be operable to cut the lowest-weight edge by creating a new node representing an operation of copying an extent to a scratch space; and making the lowest-weight edge's source node point to the new node. And a new edge may be made from a destination node of the lowest-weight edge in order to ensure that a new copy operation occurs before a consumer of the new copy operation.

**[0009]** In a further embodiment, a tangible computer-readable storage medium stores computer readable instructions of a computer program. The instructions, when executed by a computer, cause the computer to perform a method of generating an update for a computer readable operating system. The method comprises identifying a version number of a current version of the operating system; creating, with a processor, an ordered list of operations for updating the current version of the operating system to a new version of the operating system, the processor performing iterations over each regular file on the new version of the operating system to obtain the ordered list for all data blocks associated with the new version; and assembling, with the processor, a differential update file including a magic number indicator showing the differential update file is an actual update file, a new version number identifying the new version of the operating system, and a protocol buffer including the ordered list of operations.

**[0010]** In one example, each operation in the ordered list is associated with a file object, and the method further comprises creating a vertex in a graph for each file object; and creating a vector representing each block. In this case, the method may further comprise setting reader and writer parameters for the vector of each block; and for each block with a different reader and writer, creating an edge in the graph from the writer to the reader. The edge points to a file operation to be completed before a source file operation associated with the edge starts. Here, if the graph includes cycles, the method may further comprise breaking each of the cycles by finding a lowest-weight edge associated with each given cycle; and cutting the lowest-weight edge for each cycle. Cutting the lowest-weight edge may include creating a new node representing an operation of copying an extent to a scratch space; and making the lowest-weight edge's source node point to the new node. And a new edge may be made from a destination node of the lowest-weight edge in order to ensure that a new copy operation occurs before a consumer of the new copy operation.

**[0011]** In a further embodiment, a client device comprises memory for storing a current version of an operating system and a processor coupled to the memory. The processor is configured to transmit a request to a remote device regarding an update for the current version of the operating system, the request including a version number identifying the current version of the operating system; receive a differential update file from the remote device, the differential update file including a magic number indicator showing the differential update file is an actual update file, a new version number identifying the new version of the operating system, and a protocol buffer including an ordered list of operations; verify the magic number; extract the ordered list of operations from the protocol buffer; and perform a differential update by executing the ordered list of

WO 2011/088022

PCT/US2011/020790

operations in order to update the current version of the operating system to the new version of the operating system.

**[0012]** In one example, the processor is operable to perform the differential update of the operating system to the new version of the operating system without saving the differential update file in the memory while the differential update file is being streamed to the client device.

#### BRIEF DESCRIPTION OF THE DRAWINGS

**[0013]** FIG. 1 illustrates a file format in accordance with aspects of the invention.

**[0014]** FIG. 2 is a process of for cutting edges to break cycles in accordance with aspects of the present invention.

**[0015]** FIG. 3 illustrates a differential update process in accordance with aspects of the present invention.

**[0016]** FIGS. 4A-B illustrate computer systems for use in the invention.

#### DETAILED DESCRIPTION

**[0017]** The aspects, features and advantages of the present invention will be appreciated when considered with reference to the following description of preferred embodiments and accompanying figures. The same reference numbers in different drawings may identify the same or similar elements.

**[0018]** From time to time different features of an operating system are updated. Those updates may be sent to client computers that perform a predefined update process. This may involve deleting, changing and/or adding files to the client. In one process, an update is prepared by the operating system provider. This may include creating a differential update file, which indicates to the client what changes need to be made. The differential update file is then sent to each client to be updated. The clients desirably sequentially execute the operations in the file to perform the update.

**[0019]** A partition containing an operating system may be configured as one or more blocks, with each block containing file information or bookkeeping information. Each block may be, e.g., 4 kbytes, although the invention is not limited to this or any particular block size. The operating system itself comprises programs (e.g., instructions) as well as data that run on a computer processor and manages computer hardware resources and provides common services for efficient execution of various application software. When updating the operating system, a new partition may be created. The new partition is pre-populated with the old version of the operating system. Each disk block on the resultant install partition (which will contain the root file system) is desirably exactly (bit for bit) specified by the operating system vendor so that it can be signed on the server (e.g., for a verified boot).

**[0020]** It is desirable for updates to be as small as possible. In one scenario, the updates are applied in place, so that many delta (differential) updates can be installed without rebooting. Thus, if a user is booted into version N, and N+1 is released, the user (client) downloads the N→N+1 update and installs it. Then the user is still booted into N when N+2 comes out. The user then downloads the N+1→N+2 update and installs it in-place over N+1.

**[0021]** In one example, the client contacts the update server and provides the version number of the system installed on the present install partition. The server provides the client with an incremental (delta) update that is downloaded to the client. The delta update file contains an ordered list of operations to perform on the install partition that will take it from the old (existing) version to the new version.

**[0022]** Desirably, the update file is an ordered list of operations to be performed by the client. Each operation operates on specific blocks of the partition. Each operation may contain an optional data "blob" inside the delta file. The term "blob" indicates a large chunk of data that may have

an arbitrary collection of bytes. There are several types of install operations, including "Copy," "Diff", "Replace" and "Replace with Uncompression." In Copy, some of the blocks in the current install partition are copied over to other blocks in the new install partition. For Diff, some blocks are read into memory, and a binary difference routine is performed using an attached data blob. The results are written into specified blocks in the new install partition. For Replace, the attached data blob is written to specified blocks in the new install partition. A compression process may also be performed, where a compressed data blob is sent to the client, uncompressed, and the results are written to specified blocks in the new install partition. Here, Gzip, Bzip2 or another compression algorithm may be employed.

**[0023]** In one scenario, a patch program is instructed that the old file is the install partition. Rather than having the program read the entire partition into memory, it is instructed which blocks to read to get the file in memory. Then, a patch operation is performed in memory. Finally, the result is written directly to the new install partition, but not at the beginning of the device. Instead, the program is told which blocks to write the results to.

**[0024]** A delta/differential update file may be generated according to the following discussion. The update file format may be as shown in file format 100 of FIG. 1. Specifically, the format preferably includes a "magic number" indicator or sanity check to show the file is actually an update file. In one instance, the magic number includes four bytes containing the binary version of a short phrase, such as "CrAU." Next is a version number, followed by a protocol buffer offset and length, each of which may be 8 bytes. The protocol buffer itself follows. The protocol buffer is a series of instructions that the client is to perform in order. Next, one or more data blobs are provided, followed by an end of file (EOF) indicator. While the ordering may be as shown in

this figure, the different components may be ordered differently if desired.

**[0025]** In one scenario, the concept of an "extent" is employed. An extent is a contiguous range of disk blocks. For example, rather than specify blocks {10, 11, 12, 13, 14, 15, 17, 18} it can be simpler to specify { (10, 6), (17, 2) } (a list of extents). An exemplary process for generating delta updates is as follows.

```
message Manifest {
  message InstallOperation {
    enum CompressionType {
      NO_CHANGE = 0, // file is unchanged; just move data
                      blocks
      BSDIFF = 1, // Read source data blocks as old file,
                   included binary blob is diff, output to new blocks
      FULL = 2, // Output included binary blob to new
                 blocks
      FULL_BZIP = 3 // Bunzip binary blob into new blocks
    }
    uint64 blob_offset; // if present, the offset in the
                        update image of the binary blob for this file
    uint64 blob_length; // if present, binary blob length
    message extent {
      uint64 offset; // in blocksize
      uint64 length; // in blocksize
    }
    repeated extent input_extents;
    repeated extent output_extents;
  }
  repeated InstallOperation install_operations;
}
```

**[0026]** The manifest indicates a list of install operations. Input extents indicate blocks to be read. Output extents indicate blocks to be written.

**[0027]** To generate a delta update, iterations are performed over each regular file on the new file system to obtain an ordered list of all data blocks it has. This is stored in a file structure having the following format:

```
struct Extent {
  uint64 start;
  uint64 length;
}
struct File {
  string path; // path within the filesystem
```

```
vector<Extent> dst_extents; // ordered list of all
extents on the new filesystem
vector<Extent> src_extents; // Applies only for
NO_CHANGE and BSDIFF
enum CompressionType; // one of: NO_CHANGE, BSDIFF,
FULL, FULL_BZIP
}
```

[0028] Eventually, each file object will be converted into an InstallOperation message in the protocol buffer. For each file, it is desirable to determine the optimal way to compress it. There are four cases. In one case, the file hasn't changed. In the other three cases, the file has changed. In one scenario, the changed file can be compressed to the smallest size using Bzip2 or another compression algorithm. In another scenario, the file has changed and it is smallest uncompressed. In the third scenario, the file has changed and it binary-diffs the smallest. If a file is new, meaning that it was not present in the old version of the operating system, then only two of the cases apply. The data may be sent down in full, or sent down in full with compression.

[0029] In accordance with one aspect of the invention, a vertex is created in a graph for each file object. Along side the graph, a vector is created to represent each block in the install partition.

```
struct Block {
InstallOperation* reader;
InstallOperation* writer;
}
vector<Block> blocks; // length is the size of the
install partition
```

[0030] The process then goes through each block in each file object. For each block, reader and writer parameters are set for the block's vector.

[0031] Next, iterations are performed through the block's array. For each block with a different reader and writer (which are both non-null), an edge (arrow) is created in the graph from the writer to the reader. An edge in the (directed) graph points to a file operation that must complete

before the edge's source file operation starts. Thus, this process tries to ensure that if a block is both read and written by different file operations, the block is read before it is written. The edge represents blocks in the graph, so the edge's weight is the number of blocks.

**[0032]** At this point, the result is likely to have a graph with cycles. The cycles should be broken. The cycles may be found using Donald B. Johnson's circuit finding algorithm as set forth in "Finding all the Elementary Circuits of a Directed Graph," SIAM J. Comput., vol.4 no.1, March 1975, the entire disclosure of which is incorporated by reference herein. The cycles may also be found using Tarjan's Algorithm, as set forth in "Tarjan's Strongly Connected Components Algorithm," David Eppstein, Ed., Wikipedia, the entire disclosure of which is incorporated by reference herein. For each cycle, the lowest-weight edge is found and cut. An edge may be cut as follows. First, create a new node that represents an operation of copying some extents to scratch space. Then, make the edge's source node point to the new node. Here, an edge may be made from the cut edge's destination node to the new node to enforce that the new copy operation occurs before the consumer of the copy. Preferably, modify the cut edge's destination node to read from the scratch space rather than from the blocks represented by the edge being cut.

**[0033]** An example of cutting an edge to break a cycle is shown in process 200 of FIG. 2. Arrows indicate that an operation pointed to needs to occur before the operation that's doing the pointing, and needs to happen in that order because the pointing operation will overwrite data needed by the pointee. As shown in this figure, at S202 operation A reads block 3 to write block 4. And at S204 operation B reads block 4 to write block 10. The cycle is broken by cutting the edge (arrow 206) between A and B; this may be done by using temp block T. Thus, as shown, at S208 operation A reads block

3 and writes block 4. At S210, operation C reads block 4 and writes block T. And at S212, operation B' reads block T and writes block 10. Once the cycles are broken, a topological sort may be used to order all the nodes. That covers installation of all file-data blocks.

**[0034]** In one example, it would be possible to pick an edge and cut it, such as B'->C->A->Temp. In this case, the Temp operation would copy the block A to a temp area. B would be modified to B', which would then read from the temp area. However, if a second edge (e.g., C->A) is cut, then the result would be: B'->C->Temp2, and A'->Temp. This would result in two completely independent graphs, and the final order of operations could be arranged as follows:

Temp2 (the client executes this first)  
C

B'

Temp

A' (the client executes this last)

**[0035]** However, in this case B' would read the data written by the Temp operation, but Temp occurs after B'. This may happen because the graph transform done when the A->B was cut edge did not specifically state that Temp must occur before B'. In order to avoid this situation, another edge is added to make this explicit. The result is:

A->B->C->A (original)

cut A->B to obtain:

B'->C->A->Temp<-B'

Then, cut C->A to obtain:

B'->C'->Temp2<-A->Temp<-B'

Now, when a final order of operation is chosen based on the graph, the operations will occur in the correct order.

**[0036]** An exemplary process is shown in FIG. 3. At stage 1, disk images are scanned. Block indices (0-7) may be associated with different images (e.g., "sh," "foo," "bar," and "dog"). New images may be associated with the same or

different blocks. Thus, "foo" may now be associated with blocks 2-3 instead of block 4. "Sh" may be associated with blocks 4-5 instead of blocks 2-3. "Bar" may be associated with block 6 instead of block 5. And new image "cat" may be associated with block 7.

**[0037]** At stage 2, file operations are created. This corresponds to a file on the new partition. This is followed by a blocks vector. A blocks vector is an ordered list, where each element includes a file operation for a given block. Each image may have one or more source blocks and one or more destination blocks. Thus, the source block for "foo" is block 4, and its destination blocks are shown as "dst: (2,2)", indicating that the first block is block 2 and there are 2 blocks allocated. This portion also shows how each image may be provided to the client. Thus, foo and bar may be provided using a binary difference algorithm such as bsdiff, sh may merely copied, and cat may be sent in full without compression. Bsdiff is a known algorithm that computes the different between two files.

**[0038]** As shown at stage 3, a graph is created where the edge weight (e.g., 1 or 2) is equal to the number of blocks. At stage 4, the cycles are broken. Desirably, cycle(s) are cut by cutting the least weight edge in each cycle. Thus, in this example, the cycle including the arrow from sh to foo (having a weight of 1) is broken and a modified foo (foo') is obtained. And at stage 5, the final order resulting from the topological sort is shown. This is the order as executed on the client.

**[0039]** After setting the file data, the client will overwrite the non-file-data blocks with a final InstallOperation that unzips the remaining data into all non-file-data blocks. In a typical example, this is about 2 megabytes of data compressed. It may be feasible to perform a delta compression process on this data.

**[0040]** Because the protocol buffer (which lists all operations) occurs at the beginning of the file, the update doesn't need to be saved to disk. It can be applied while streaming from the server. It is desirable to make sure that the update is signed by the OS vendor. The system can begin to apply the update and not mark it bootable until after the delta update signature is verified.

**[0041]** The embodiments presented here have been found to give the best compression ratio in practice. It also allows the operating system vendor employ alternative compression schemes in the future. For instance, one process developed by Google is Courgette, which may replace bsdiff.

**[0042]** Another possible solution is to delta-compress the entire partition. However, bsdiff would be unworkable in this scenario because its memory requirements are too high. During patching, bsdiff needs enough memory to store the original and new files, which may exceed 1 gigabyte. Another delta compression program, Xdelta, uses a sliding window. In testing, this resulted in poor compression (e.g., hundreds of megabytes). A further alternative is to use rdiff, which works by storing only changed blocks in the delta file. It uses a sliding window so that blocks don't need to be aligned. When tested, an rdiff delta of the entire partition was 104 megabytes. This is well above the roughly 10 megabytes that may be used in accordance with the aforementioned procedure.

**[0043]** In the future, it may be possible to use rdiff- (i.e., rsync-) style delta compression at the file level. This could be used alongside bsdiff in the future.

**[0044]** The updating procedures according to aspects of the invention may be implemented with the following exemplary computer system. FIG. 4A presents a schematic diagram of a computer system depicting various computing devices that can be used alone or in a networked configuration in the invention. For example, this figure illustrates a computer network 300 having a plurality of computers 302, 304, 306 and

WO 2011/088022

PCT/US2011/020790

308 as well as other types of devices such as portable electronic devices such as a mobile phone 310 and a PDA 312. However, the invention is no so limited, and other devices including netbooks and pad-type handheld computers (not shown) may also be used. Such devices may be interconnected via a local or direct connection 314 and/or may be coupled via a communications network 316 such as a LAN, WAN, the Internet, etc., and which may be wired or wireless.

**[0045]** Each device may include, for example, one or more processing devices and have user inputs such as a keyboard 318 and mouse 320 and/or various other types of input devices such as pen-inputs, joysticks, buttons, touch screens, etc., as well as a display 322, which could include, for instance, a CRT, LCD, plasma screen monitor, TV, projector, etc. Each computer 302, 304, 306 and 308 may be a personal computer, server, etc. By way of example only, computers 302 and 306 may be personal computers while computer 304 may be a server and computer 308 may be a laptop.

**[0046]** As shown in FIG. 4B, each computer such as computers 302 and 304 contains a processor 324, memory/storage 326 and other components typically present in a computer. For instance, memory/storage 326 stores information accessible by processor 324, including instructions 328 that may be executed by the processor 324 and data 330 that may be retrieved, manipulated or stored by the processor. The instructions 328 at the server may include operations for creating a differential update to be installed by one or more client devices. And the instructions at the client device may include operations for performing the differential update.

**[0047]** The data may include one or more differential updates maintained in a database for service to client devices. The memory/storage may be of any type or any device capable of storing information accessible by the processor, such as a hard-drive, ROM, RAM, CD-ROM, flash memories, write-capable or read-only memories. The processor 324 may comprise

any number of well known processors, such as processors from Intel Corporation or Advanced Micro Devices. Alternatively, the processor may be a dedicated controller for executing operations, such as an ASIC or other processing device.

**[0048]** The instructions 328 may comprise any set of instructions to be executed directly (such as machine code) or indirectly (such as scripts) by the processor(s). In that regard, the terms "instructions," "steps" and "programs" may be used interchangeably herein. The instructions may be stored in any computer language or format, such as in object code or modules of source code. The functions, methods and routines of instructions in accordance with the present invention are explained in more detail below.

**[0049]** Data 330 may be retrieved, stored or modified by processor 324 in accordance with the instructions 328. The data may be stored as a collection of data. For instance, although the invention is not limited by any particular data structure, the data may be stored in computer registers, in a relational database as a table having a plurality of different fields and records, in a web page cache, as XML documents, etc.

**[0050]** The data may also be formatted in any computer readable format such as, but not limited to, binary values, ASCII or Unicode. Moreover, the data may include any information sufficient to identify the relevant information, such as descriptive text, proprietary codes, pointers, references to data stored in other memories (including other network locations) or information which is used by a function to calculate the relevant data. Furthermore, a given item may comprise one or more files, a data set stored in a database, a web cache, etc. Depending on the size and content of the data, parts thereof may be stored or otherwise maintained separately.

**[0051]** Although the processor 324 and memory 326 are functionally illustrated in FIG. 4B as being within the same

block, it will be understood that the processor and memory may actually comprise multiple processors and memories that may or may not be stored within the same physical housing or location. For example, some or all of the instructions and data may be stored on a removable CD-ROM, DVD-ROM or flash drive, and others within a read-only computer chip. Some or all of the instructions and data may be stored in a location physically remote from, yet still accessible by, the processor. Similarly, the processor may actually comprise a collection of processors which may or may not operate in parallel. Data may be distributed and stored across multiple memories 326 such as hard drives or the like.

**[0052]** In one aspect, server 304 may communicate with one or more client computers 302, 306 and/or 308, as well as devices such as mobile phone 310 and PDA 312. Each client computer or other client device may be configured similarly to the server 304, with a processor, memory and instructions, as well as one or more user input devices 318, 320 and a user output device, such as display 322. Each client computer may be a general purpose computer, intended for use by a person, having all the components normally found in a personal computer such as a central processing unit ("CPU"), display, CD-ROM or DVD drive, hard-drive, mouse, keyboard, touch-sensitive screen, speakers, microphone, modem and/or router (telephone, cable or otherwise) and all of the components used for connecting these elements to one another.

**[0053]** The server 304, user computers and other devices are capable of direct and indirect communication with other computers, such as over network 316. Although only a few computing devices are depicted in FIGS. 4A-B, it should be appreciated that a typical system can include a large number of connected servers and clients, with each different computer being at a different node of the network. The network 316, and intervening nodes, may comprise various configurations and protocols including the Internet, intranets, virtual private

networks, wide area networks, local networks, private networks using communication protocols proprietary to one or more companies, Ethernet, WiFi, Bluetooth or TCP/IP.

**[0054]** Communication across the network, including any intervening nodes, may be facilitated by any device capable of transmitting data to and from other computers, such as modems (e.g., dial-up or cable), network interfaces and wireless interfaces. Server 304 may be a web server. Although certain advantages are obtained when information is transmitted or received as noted above, other aspects of the invention are not limited to any particular manner of transmission of information. For example, in some aspects, the information may be sent via a medium such as a disk, tape, CD-ROM, or directly between two computer systems via a dial-up modem. In other aspects, certain information may be transmitted in a non-electronic format and manually entered into the system.

**[0055]** Moreover, computers and user devices in accordance with the systems and methods described herein may comprise any device capable of processing instructions and transmitting data to and from humans and other computers, including network computers lacking local storage capability, PDA's with modems such as PDA 312, Internet-capable wireless phones such as mobile phone 310, netbooks and pad-type handheld computers.

**[0056]** As shown in FIG. 4A, the network 300 may also include an update database 332 for serving differential updates to client devices. The update database may be directly or indirectly coupled to server 304. In an alternative, the update database 332 may be part of or otherwise logically associated with the server 304.

**[0057]** Although the invention herein has been described with reference to particular embodiments, it is to be understood that these embodiments are merely illustrative of the principles and applications of the present invention. It is therefore to be understood that numerous modifications may be made to the illustrative embodiments and that other

WO 2011/088022

PCT/US2011/020790

arrangements may be devised without departing from the spirit and scope of the present invention as defined by the appended claims. For instance, while certain embodiments are shown with regard to operating systems, differential updates according to aspects of the invention may be employed with other software packages, applications and services. Furthermore, while particular processes are shown in a specific order in the appended drawings, such processes are not limited to any particular order unless such order is expressly set forth herein, and may be performed in a different order or in parallel. And additional processes may be added or other processes omitted unless it is specifically stated otherwise.

#### INDUSTRIAL APPLICABILITY

**[0058]** The present invention enjoys wide industrial applicability including, but not limited to, computer system operation and updates of applications for such computer systems.

## IN THE CLAIMS

1. A method of generating an update for a computer readable operating system, the method comprising:

identifying a version number of a current version of the operating system;

creating, with a processor, an ordered list of operations for updating the current version of the operating system to a new version of the operating system, the processor performing iterations over each regular file on the new version of the operating system to obtain the ordered list for all data blocks associated with the new version; and

assembling, with the processor, a differential update file including a magic number indicator showing the differential update file is an actual update file, a new version number identifying the new version of the operating system, and a protocol buffer including the ordered list of operations.

2. The method of claim 1, wherein one or more operations in the ordered list are each associated with a respective data blob indicating a chunk of data, and wherein the differential update file is assembled to include the respective data blobs.

3. The method of any of the preceding claims, wherein each operation in the ordered list is applicable to one or more specific data blocks of a partition of a client device, the operations including one or more of: a copy operation where at least one of the data blocks in the partition is to be copied to another block in a new partition of the client device for the new version of the operating system; a difference operation where at least a given one of the data blocks is read into memory and a difference routine is performing on the at least one given data block using a data blob of the differential update file; a replace operation where a selected data blob of the differential update file is

configured to be written to specified blocks in the new partition; and a replace with uncompression operation where a compressed data blob is included in the differential update file and is configured to be written to selected specified blocks in the new partition for the new version of the operating system.

4. The method of any of the preceding claims, wherein each operation in the ordered list is associated with a file object, and the method further comprises:

    creating a vertex in a graph for each file object;  
and

    creating a vector representing each block.

5. The method of claim 4, further comprising:

    setting reader and writer parameters for the vector of each block; and

    for each block with a different reader and writer, creating an edge in the graph from the writer to the reader, the edge pointing to a file operation to be completed before a source file operation associated with the edge starts.

6. The method of claim 5, wherein each edge has a weight, and the weight identifies a number of blocks in the graph associated with that edge.

7. The method of claim 5 or claim 6, wherein if the graph includes cycles, the method further comprises breaking each of the cycles.

8. The method of claim 7, wherein breaking a given one of the cycles includes:

    finding a lowest-weight edge associated with the given cycle; and

    cutting the lowest-weight edge.

9. The method of claim 8, wherein cutting the lowest-weight edge comprises:

creating a new node representing an operation of copying an extent to a scratch space; and

making the lowest-weight edge's source node point to the new node.

10. The method of claim 9, wherein a new edge is made from a destination node of the lowest-weight edge in order to ensure that a new copy operation occurs before a consumer of the new copy operation.

11. A device for generating an update for a computer readable operating system, the device comprising:

memory for storing differential update information associated with the operating system; and

a processor coupled to the memory, the processor being configured to:

identify a version number of a current version of the operating system;

create, using the differential update information, an ordered list of operations for updating the current version of the operating system to a new version of the operating system, including performing iterations over each regular file on the new version of the operating system to obtain the ordered list for all data blocks associated with the new version;

assemble a differential update file including a magic number indicator showing the differential update file is an actual update file, a new version number identifying the new version of the operating system, and a protocol buffer including the ordered list of operations; and

store the differential update file in the memory.

12. The device of claim 11, wherein each operation in the ordered list is associated with a file object, and the processor is further configured to:

create a vertex in a graph for each file object; and  
create a vector representing each block.

13. The device of claim 12, wherein the processor is further configured to:

set reader and writer parameters for the vector of each block; and

for each block with a different reader and writer, create an edge in the graph from the writer to the reader, the edge pointing to a file operation to be completed before a source file operation associated with the edge starts.

14. The device of claim 13, wherein if the graph includes cycles, the processor is further operable to break each of the cycles.

15. The device of claim 14, wherein breaking a given one of the cycles includes:

finding a lowest-weight edge associated with the given cycle; and

cutting the lowest-weight edge.

16. The device of claim 15, wherein the processor is operable to cut the lowest-weight edge by:

creating a new node representing an operation of copying an extent to a scratch space; and

making the lowest-weight edge's source node point to the new node.

17. The device of claim 16, wherein a new edge is made from a destination node of the lowest-weight edge in order to

ensure that a new copy operation occurs before a consumer of the new copy operation.

18. A tangible computer-readable storage medium on which computer readable instructions of a computer program are stored, the instructions, when executed by a computer, cause the computer to perform a method of generating an update for a computer readable operating system, the method comprising:

identifying a version number of a current version of the operating system;

creating, with a processor, an ordered list of operations for updating the current version of the operating system to a new version of the operating system, the processor performing iterations over each regular file on the new version of the operating system to obtain the ordered list for all data blocks associated with the new version; and

assembling, with the processor, a differential update file including a magic number indicator showing the differential update file is an actual update file, a new version number identifying the new version of the operating system, and a protocol buffer including the ordered list of operations.

19. The storage medium of claim 18, wherein each operation in the ordered list is associated with a file object, and the method further comprises:

creating a vertex in a graph for each file object; and

creating a vector representing each block.

20. The storage medium of claim 19, wherein the method further comprises:

setting reader and writer parameters for the vector of each block; and

for each block with a different reader and writer, creating an edge in the graph from the writer to the reader, the edge pointing to a file operation to be completed before a source file operation associated with the edge starts.

21. The storage medium of claim 20, wherein if the graph includes cycles, the method further comprises breaking each of the cycles by:

finding a lowest-weight edge associated with each given cycle; and

cutting the lowest-weight edge for each cycle.

22. The storage medium of claim 21, wherein cutting the lowest-weight edge comprises:

creating a new node representing an operation of copying an extent to a scratch space; and

making the lowest-weight edge's source node point to the new node.

23. The storage medium of claim 22, wherein a new edge is made from a destination node of the lowest-weight edge in order to ensure that a new copy operation occurs before a consumer of the new copy operation.

24. A client device, comprising:

memory for storing a current version of an operating system; and

a processor coupled to the memory, the processor being configured to:

transmit a request to a remote device regarding an update for the current version of the operating system, the request including a version number identifying the current version of the operating system;

receive a differential update file from the remote device, the differential update file including a magic

number indicator showing the differential update file is an actual update file, a new version number identifying the new version of the operating system, and a protocol buffer including an ordered list of operations;

verify the magic number;

extract the ordered list of operations from the protocol buffer; and

perform a differential update by executing the ordered list of operations in order to update the current version of the operating system to the new version of the operating system.

25. The client device of claim 24, wherein the processor is operable to perform the differential update of the operating system to the new version of the operating system without saving the differential update file in the memory while the differential update file is being streamed to the client device.

FIG. 1  
100

Magic Number	Version Number	Protocol Buffer Offset	Protocol Buffer Length	Protocol Buffer	Data Blobs	EOF
--------------	----------------	------------------------	------------------------	-----------------	------------	-----

FIG. 2

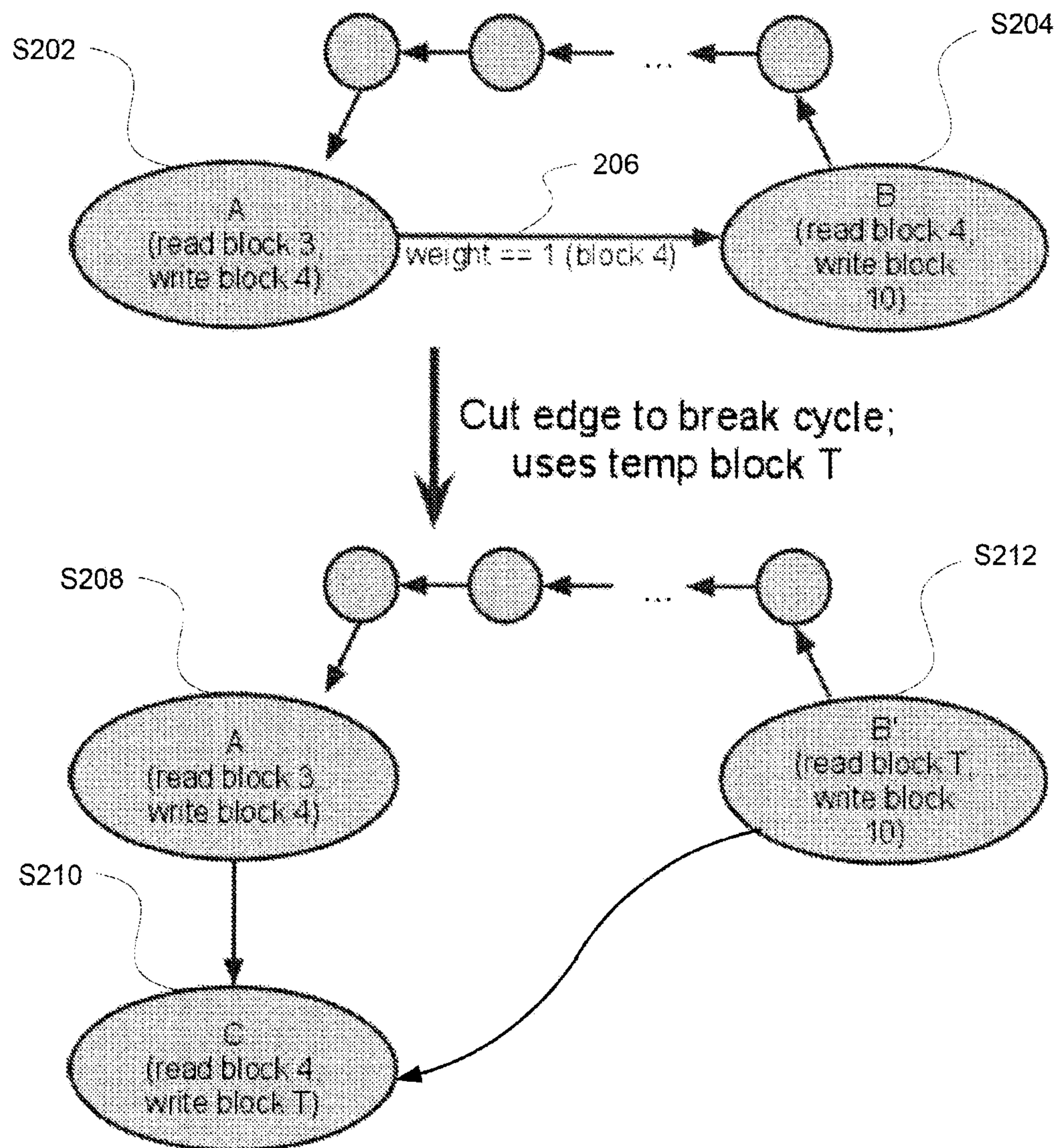
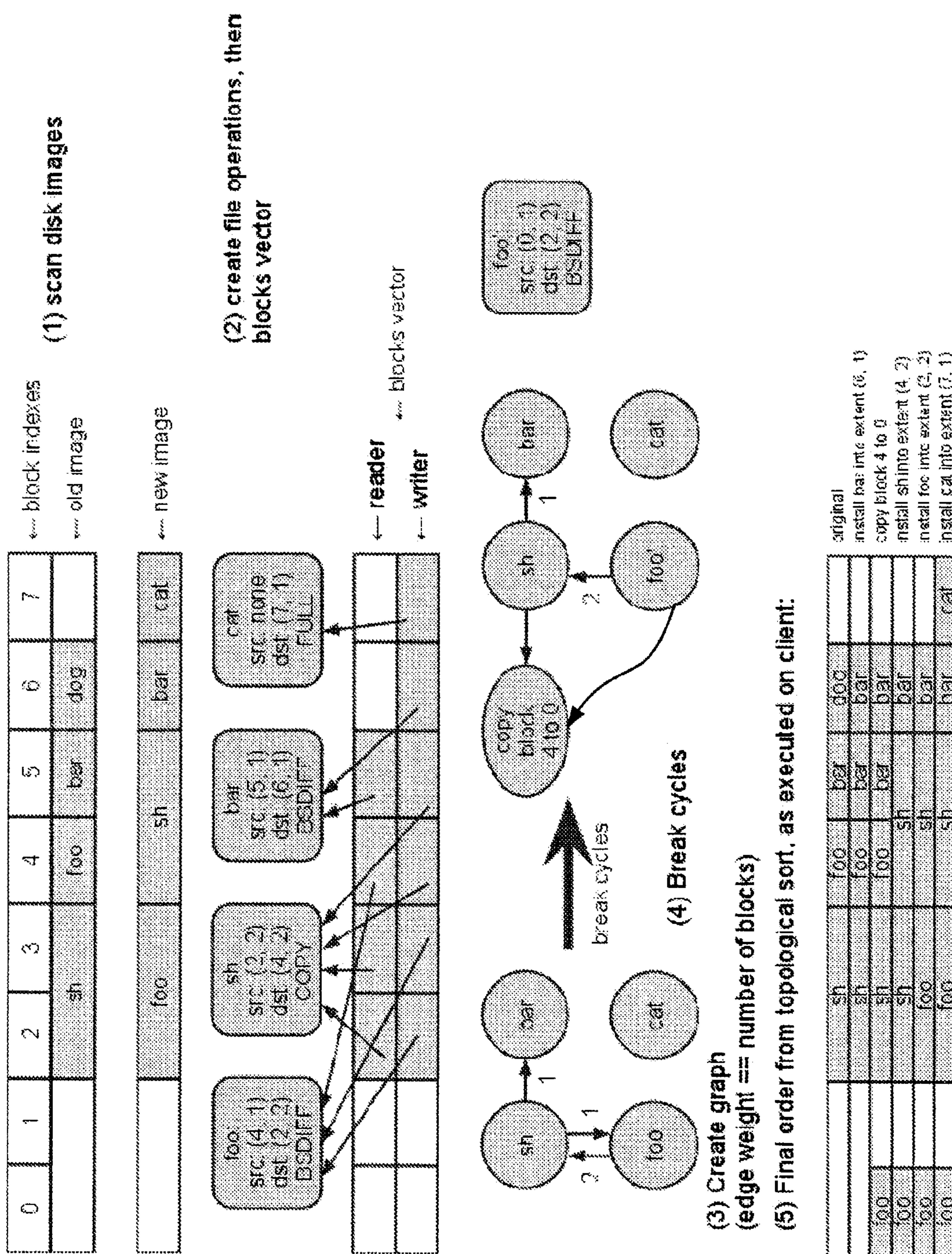
200

FIG. 3



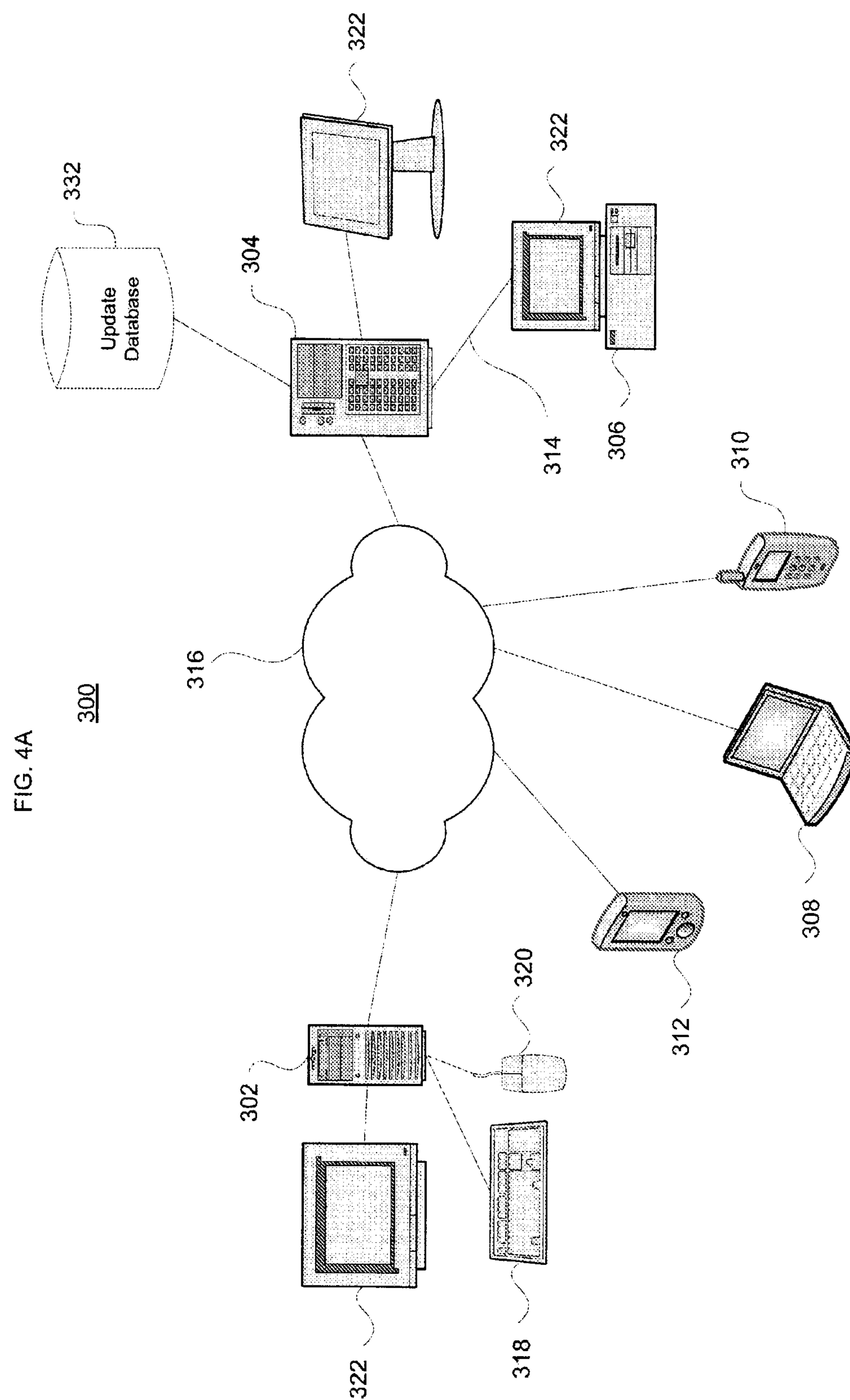


FIG. 4B

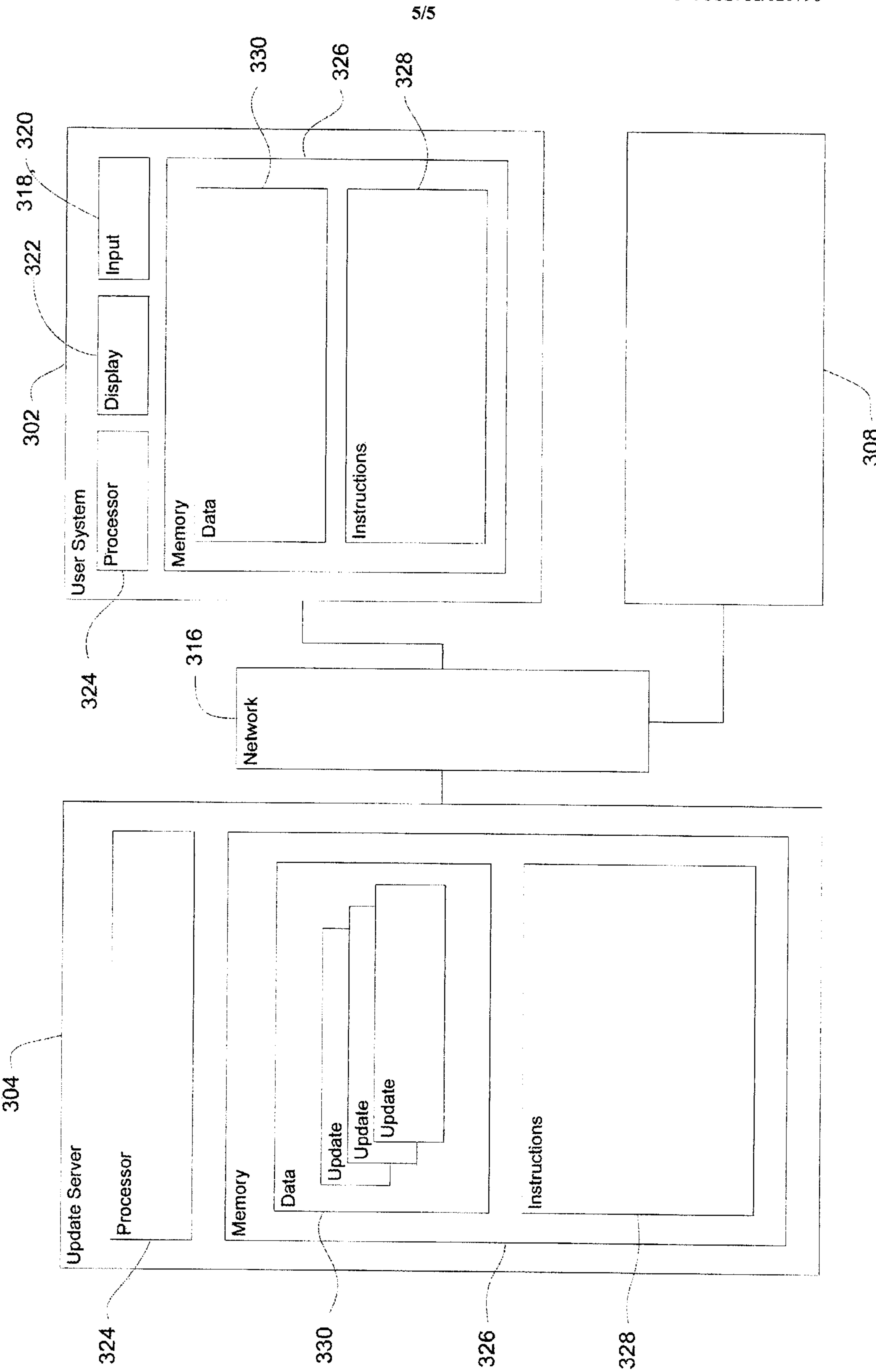


FIG. 2

200

