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

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
5 June 2008 (05.06.2008)

PCT

(10) International Publication Number
WO 2008/067226 A1

(51) International Patent Classification:
G06F 7/24 (2006.01)

(21) International Application Number:
PCT/US2007/085357

(22) International Filing Date:
21 November 2007 (21.11.2007)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
11/566,122 1 December 2006 (01.12.2006) US

(71) Applicant *(for all designated States except US)*: **NEC LABORATORIES AMERICA, INC.** [US/US]; 4 Independence Way, Suite 200, Princeton, New Jersey 08540 (US).

(72) Inventors: **DUBNICKI, Cezary**; 1614 Mulberry Court, Monmouth Junction, New Jersey 08852 (US). **LICHOTA, Krzysztof**; A1. Ken 26/8, PL-02-797 Warszawa (PL). **KRUUS, Erik**; 261 Hamilton Road East, Hillsborough, New Jersey 08844 (US). **UNGUREANU, Cristian**; 147 Hamilton Avenue, Princeton, New Jersey 08540 (US).

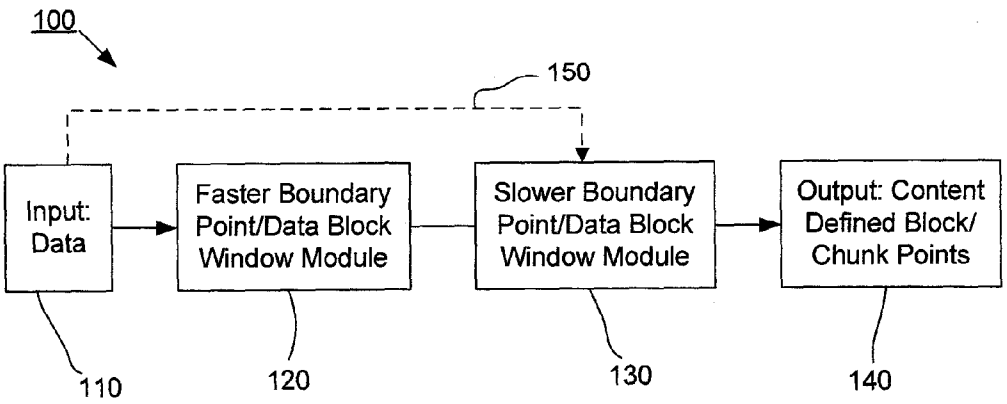
(74) Agent: **KOLODKA, Joseph**; 4 Independence Way, Suite 200, Princeton, New Jersey 08540 (US).

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

(84) Designated States *(unless otherwise indicated, for every kind of regional protection available)*: ARIPO (BW, GH, GM, KE, 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, LV, MC, MT, 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:
— *with international search report*

(54) **Title:** METHODS AND SYSTEMS FOR DATA MANAGEMENT USING MULTIPLE SELECTION CRITERIA



(57) **Abstract:** Systems and methods for data management and data processing are provided. Embodiments may include systems and methods relating to fast data selection with reasonably high quality results, and may include a faster data selection function and a slower data selection function. Various embodiments may include systems and methods relating to data hashing and/or data redundancy identification and elimination for a data set or a string of data. Embodiments may include a first selection function is used to pre-select boundary points or data blocks/windows from a data set or data stream and a second selection function is used to refine the boundary points or data blocks/windows. The second selection function may be better at determining the best places for boundary points or data blocks/windows in the data set or data stream. In various embodiments, data may be processed by a first faster hash function and slower more discriminating second hash function.

WO 2008/067226 A1

METHODS AND SYSTEMS FOR DATA MANAGEMENT USING MULTIPLE SELECTION CRITERIA

[0001] This patent application is related to U.S. Patent Application No. (TBD), titled METHODS AND SYSTEMS FOR QUICK AND EFFICIENT DATA MANAGEMENT AND/OR PROCESSING to Cezary Dubnicki, Erik Kruus, and Cristian Ungureanu, also filed on December 1, 2006, which is hereby incorporated herein by reference for all purposes.

[0002] This disclosure may contain information subject to copyright protection, for example, the various exemplary C++ codes and pseudocodes presented herein. The copyright owner has no objection to the facsimile reproduction by anyone of the patent disclosure or the patent as it appears in the U.S. Patent and Trademark Office files or records, but otherwise reserves all copyright rights whatsoever.

BACKGROUND

FIELD OF THE INVENTION

[0003] The present invention relates to the field of data processing and data management and, more specifically, to methods and systems related to quick data processing for applications such as data hashing and/or data redundancy elimination.

DESCRIPTION OF RELATED ART

[0004] Every day more and more information is created throughout the world and the amount of information being retained and transmitted continues to compound at alarming rates, raising serious concerns about data processing and management. Much of this information is created, processed, maintained, transmitted, and stored electronically. The mere magnitude of trying to manage all this data and related data streams and storage is staggering. As a result, a number of systems and methods have been developed to process data more quickly and to store and transmit less data by eliminating as much duplicate data as possible. For example, various systems and methods have been developed to help reduce the need to store, transmit, etc., duplicate data from the various electronic devices such as computers, computer networks (e.g., intranets and the Internet), mobile devices such telephones and PDA's, hardware storage devices, etc. Further, there is a need to encrypt data using cryptography, particularly during e.g., data transmission. For example, systems and methods have been developed that provide for strong

(i.e. cryptographic) hashing, and such methods may be incorporated quite naturally within applications that use data hashing to accomplish data redundancy elimination over insecure communication channels.

[0005] In various electronic data management methods and systems, a number of methodologies have been developed to hash data and/or to eliminate redundant data from, for example, data storage and data transmission. These techniques include various data compression, data hashing, and cryptography methodologies. Some exemplary techniques are disclosed in various articles including Philip Koopman, 32-Bit Cyclic Redundancy Codes for Internet Applications, Proceedings of the 2002 Conference on Dependable Systems and Networks, 2002; Jonathan Stone and Michael Greenwald, Performance of Checksums and CRCs over Real Data, IEEE/ACM Transactions on Networking, 1998; Val Henson and Richard Henderson, An Analysis of Compare-by-Hash, Proceedings of the Ninth Workshop on Hot Topics in Operating Systems, Lihue, Hawaii, May 2003, pp. 13-18; and Raj Jain, A Comparison of Hashing Schemes for Address Lookup in Computer Networks, IEEE Transactions on Communications, 1992. There are also a number of U.S. patents and patent publications that disclosed various exemplary techniques, including U.S. Patent Pub. Nos. 2005/0131939, 2006/0047855, and 2006/0112148 and U.S. Patent Nos. 7,103,602, and 6,810,398.

[0006] However, the known techniques lack certain useful capabilities. Typically the better performing selection techniques (e.g., high data redundancy elimination) use too much processing time (take too long) and very fast data selection techniques may lack the desired degree of data elimination. For example, there are a number of hashing function approaches, including whole file hashing, fixed size data block hashing, and content-defined data chunk hashing. However, none of these techniques are reasonably fast (using only a amount of computation time) and have the ability to identify most of the data redundancies in a data set (e.g., have high data redundancy elimination).

[0007] Therefore, there is a need for a data selection technique that has reasonable performance and is fast. For example, a hashing and/or data redundancy identification and elimination system and method is needed that can quickly perform data hashing and/or data redundancy identification and elimination while still identifying most of the redundant data in a data set. There is also a need for systems and methods that more quickly determine appropriate break points or boundaries for determining data blocks or chunks in a content defined hash function.

SUMMARY

[0008] The present invention is directed generally to providing systems and methods for data management and data processing. For example, embodiments may include systems and methods relating to fast data selection with better performing selection results (e.g., high data redundancy elimination), and may include a faster data selection process and a slower data selection process. Further, exemplary systems and methods relating to a data hashing and/or data redundancy identification and elimination for a data set or a string of data are provided. The present invention may be more robust and may provide systems and methods which combine some of the speed of a fast hash with the more robust performance characteristics of a hash that determines a more appropriate set of break points or boundaries. The invention may be a computer implemented invention that includes software and hardware for improving data processing speed without notably reducing the quality of the data processing results.

[009] In various embodiments of the present invention, for example, systems and methods are provided which may include a first selection function used to pre-select boundary points or data blocks/windows from a data set or data stream and may include a second selection function that may refine the boundary points or data blocks/windows from a data set or data stream. The first selection function may be faster to compute (e.g., take less processing time to roll, that is, produce a hash value for a subsequent data block/window from said data set or data stream) than the second selection function. The second selection function may be better at determining appropriate places for boundary points or data blocks/windows in the data set or data stream. The second selection function may be, but need not be, rollable. One exemplary first selection function may be a boxcar sum based function using, for example, moving windows. This is a particularly fast data selection process. One exemplary second selection function may be a Rabin fingerprint function. This first selection function is particularly fast and this second selection function has particularly good bit-randomizing characteristics. In various embodiments the first selection function and the second selection function may be used to generate a plurality of content-defined boundary points, block break points or chunk points for determining at what point a data set or data stream should be segregated into various data blocks, chunks, or windows. In various embodiments, the content defined boundaries may be used for determining the data block, chunk, or window sizes to which a hash function may be applied and a resulting hash value may be produced. In various embodiments, the resulting hash value may be compared to one or more stored hash values to determine if the data block, chunk, or window is

entirely duplicate data or whether the new hash value should be stored as a unique data block, chunk, or window.

[0010] In various embodiments, a content-defined technique for data hashing may be provided. The content-defined technique may include a processing system having a faster hash function module (e.g., taking less processing time because there are few calculations) and a slower hash function module (e.g., taking more processing time because there are more calculations). The faster hash function module may receive data from, for example, a data stream, and may pre-select one or more data boundary, break, block, or chunk points at which the data stream could be broken into distinct blocks or chunks of data. The faster hash function may be somewhat lower performance when determining the best data boundary, break, block, or chunk points. The faster hash function may be, for example, a boxcar sum function that uses, for example, a rolling or moving window. The slower hash function module may perform hashing on only those data blocks or chunks that have been pre-selected by the faster hash function. The slower hash function module may be better at determining the best data boundary, break, block, or chunking points for achieving a particular objective (e.g., identifying more redundancies). The slower hash function may be, for example, a Rabin fingerprint function, a, SHA-1 hash, a CRC32c hash, etc., and may be rolling or non-rolling.

[0011] Still further aspects included for various embodiments will be apparent to one skilled in the art based on the study of the following disclosure and the accompanying drawings thereto.

BRIEF DESCRIPTION OF THE DRAWINGS

[0012] The utility, objects, features and advantages of the invention will be readily appreciated and understood from consideration of the following detailed description of the embodiments of this invention, when taken with the accompanying drawings, in which same numbered elements are identical and:

[0013] Fig. 1 is an exemplary data management or processing system using multi-mode data boundary point determination, according to at least one embodiment;

[0014] Fig. 2 is an exemplary data management or processing method using multi-mode data boundary point determination, according to at least one embodiment;

[0015] Fig. 3 is an exemplary illustration of how data management or processing using multi-mode data boundary point determination system(s) and method(s) operates, according to at least one embodiment;

[0016] Fig. 4 is a detailed exemplary method for data management or processing that uses multi-mode data boundary point determination for identification of previously encountered data in a data set or stream, according to at least one embodiment;

[0017] Fig. 5 is an exemplary hashing technique using content-defined data blocks or chunks, according to at least one embodiment;

[0018] Fig. 6 is an exemplary first selection function using a boxcar sum technique, according to at least one embodiment;

[0019] Figs. 7a and 7b provide exemplary illustrations of data windows, blocks or chunks for an exemplary hashing technique using content-defined data blocks or chunks, according to at least one embodiment;

[0020] Fig. 8 is an exemplary functional block diagram of a computing device, according to at least one embodiment; and

[0021] Fig. 9 is an exemplary functional block diagram illustrating a network, according to at least one embodiment.

DETAILED DESCRIPTION

[0022] The present invention is directed generally to systems and methods for data management and data processing. Various embodiments may include systems and methods relating to a multi-mode data selection techniques. More specifically, various embodiments may include systems and methods relating to fast data selection with reasonably high quality results, and may include a faster data selection process and a slower data selection process. In various embodiments of the present invention, for example, systems and methods are provided which may include a first selection function used to pre-select boundary points or data blocks/windows from a data set or data stream and may include a second selection function that may refine the boundary points or data blocks/windows from a data set or data stream. The first selection function may be faster to compute (e.g., take less processing time) than the second selection function. The second selection function may be better at determining the best places for boundary points or data blocks/windows in the data set or data stream. One exemplary first selection function may be a boxcar sum based function using, for example, moving windows. This is a particularly fast data selection process. One exemplary second selection function may be a Rabin fingerprint function that may be rolling or non-rolling. Other selection functions have the faster/slower characteristics are equally applicable.

[0023] The invention may be a computer implemented invention that includes software and hardware for improving data processing speed without notably reducing the quality of the data processing results. In at least one embodiment, the system(s) and method(s) provided herein may be implemented using a computing device, and may be operational on one or more computer(s) within a network. Details of exemplary computing device(s) and network(s) are described in some detail in, Fig. 8 and Fig. 9. Prior reference to those examples may prove helpful in developing a better appreciation for various details of the invention.

[0024] In any case, for ease of understanding the present invention will be explained in more detail for use with hashing functions and/or data redundancy identification and /or data duplication elimination. However, one skilled in the art would appreciate that the present invention may be applicable to other data management and processing systems and methods including computers with a string of data to process or store, wireless communications that have data to transmit, Internet and Intranet applications, data encryption techniques, etc. In particular, the exemplary embodiments used herein to explain the present invention relate primarily to data hashing and data duplication elimination.

[0025] In data hashing and/or data redundancy identification and/or elimination embodiments, the present invention may include a faster hashing function and a slower hashing function. The fast hashing function may be for chunk, block or hash point pre-selection. The slower hashing function may be applied to determine which of the pre-selected chunk, block or hash points are better at identifying the most data duplication. Hashing functions for a data set or a string of data may be used to identify identical sections of data in large amounts of data. There are three general techniques for hashing: whole file content hashing, fixed size block hashing, and content-defined hashing. Whole file content hashing operates to create a hash value or check sum for entire files of data (e.g., a complete text or image file). Fixed size data block hashing may operate to create hash values or check sums for predetermined fixed size portions (e.g., 1000 bits) of various data files. Content-defined data block hashing may operate to create hash values or check sums based on variable size data blocks whose size is determined by a selected data content standard (e.g., break the data into a data chunk after X numbers of consecutive 1's or 0's are found in the data).

[0026] In the next few paragraphs, the speed of the SHA-1 may be almost the same in all three cases; for whole-file, fixed-size, and variable-size chunking. In each case, every input byte is in a unique chunk, and every such chunk must have the SHA-1 evaluated. However, one of the main things that may differ between whole-file, fixed-size, variable-size chunking is the

number of such SHA-1 calculations. Nevertheless, for the SHA-1 evaluations, the total processing time (e.g., CPU time) will be almost the same, since the bulk of the processing time will mostly be in the SHA-1 calculation itself, and not in overhead related to storing one or several SHA-1 hash values.

[0027] In the case of whole file hashing, hashing may be performed by applying a hashing function to all the data of entire files. For example, a SHA-1 hashing function might be used and applied to an entire data file. The SHA-1 hashing function is computationally complex and may be slow relative to some other hashing functions. Regardless, in this case, for purposes of identifying and eliminating duplication, the least amount of data duplication is found and eliminated because when a single bit of data changes in a file, the resulting hash value will be different than previously saved and the full amount of data associated with the revised file will need to be transmitted or saved (e.g., when one letter in a text file is changed, the entire data representation of the text file and it's hash value will change so that it will not be a duplicate of a previous version of the same text file). On the other hand the hashing is quick because the hashing function need only be operated once for an entire file of data.

[0028] As noted above, a fixed size data block hashing function performs hashing on portions or blocks of the entire data found in a whole file (e.g., a single text file may be broken up into 10 same sized data blocks of 10K bits), and data blocks may be set at a non-overlapping fixed size. Once again, a SHA-1 hashing function might be applied to each of a fixed size set of blocks (e.g., 10K bits) that make up a whole file (e.g., 100K bits). In this case, more duplication may be found because the block of data hashed each time is smaller, and a single bit change somewhere in a whole file will only result in a change in one of the multiple blocks that make up a whole file (e.g., 9 of the 10 10K bit blocks will be duplicates). However, a single byte insertion may hinder duplicate detection for a large number of blocks (e.g., the blocks following the block containing the first insertion may be rendered non-duplicate). The smaller the block, the better redundancy detection, but a slightly slower process because the hashing function, for example SHA-1, must be run more times for the same amount of data found in the whole data file.

[0029] Finally, using the content defined data chunk hashing, hashing may be performed by applying a fairly slow and somewhat better performance (e.g., more accurate and discriminating calculation) to identify and generate a value for various chunks of data that are defined by their content. In this case, both localized insertion and localized replacement modifications may result in changes to a single content-defined chunk, thereby increasing the amount of duplication found when compared to fixed-size chunking. One such hashing function

may include a combination of Rabin fingerprinting and SHA-1 hashing function. The Rabin fingerprinting may be applied multiple times for overlapping data windows (e.g., sliding window) of the data in the data file to determine where in the data file the chunk boundaries should be set, based on a predetermined boundary point criteria (e.g., the value of the function equals a predetermined number of bits or bytes ending in X number of 0's or 1's), then the SHA-1 hashing function may be applied to each of the determined data blocks (whose size varies based on the underlying data being analyzed). Again, each byte of input may enter into some SHA-1 hash calculation, as noted before. However, the Rabin fingerprinting presents an additional calculation burden (e.g. processing time) when compared to fixed size chunking. Although this approach is very good at identifying many more data redundancies, both of these functions can be time consuming and in combination make hashing and/or data redundancy identification and elimination very time consuming. In fact, the Rabin fingerprinting function may be particularly time consuming for identifying where in particular the various data block cuts or hash points should be in attempting to optimize the redundancy data identification and/or data elimination.

[0030] For the purposes of chunking, one means to achieve the desired degree of data elimination is to employ a selection function that produces chunks from real-world data with a content-defined, deterministic procedure in a manner that attains a chunk size distribution which matches closely the distribution expected given random data input. One way to accomplish this is to employ a hash function followed by a selection criterion with known probabilities of occurring. Typically, a single hash function is selected that has an excellent ability to "scramble" the bits within any data window. In this fashion, the output value exhibits little correlation with the actual data bits within the input window. With such a bit-randomizing hash value, H, simple selection criterion that have been used include, for example, insisting that a certain number of bits be equal to a predetermined value, or requiring that the value of H ^ (H-1) > threshold, H > threshold, or H and mask == value. Coupling a bit-randomizing hash with a selection procedure criterion may yield a selection function that selects a "random-looking" subset of input windows. The common hope is that such selection functions have excellent chances of producing a random-like chunk size distribution when presented with real-life (e.g., non-random) data inputs.

[0031] Various selection functions that are expected to have random-like chunk size distribution in the face of real-life data may be referred to herein as having good "bit-scrambling" ability. One performance goal for the present invention may involve two aspects: one is easily understood, namely, speed of operation; the other, somewhat abstract, is the aforementioned bit-scrambling behavior. The present invention may provide a mechanism for data processing

applications to greatly increase the speed of operation, while still retaining a measure of control over how much bit-scrambling ability may possibly be compromised. Selection functions satisfying the abstract bit-scrambling goal well may result in applications better able to fulfill non-abstract goals particular to their domain. For example, in various applications these goals may be related to duplicate detection, including: better duplicate detection, better potential duplicate detection, better resemblance detection, less data transmitted, increased bandwidth, decreased storage requirements, etc. Use of the present invention within such applications may result in them fulfilling, for example, non-abstract goals at greater speed. Some specific applications of the present invention may focus on data storage system(s) as an example, in which instance the concrete performance goals may be speed and amount of duplicate elimination. Hence, hash functions that may yield better performance (in our sense) on real-world data may tend to have little bitwise correlation between the value at on instant or window and the value at the next instant or window. Desirable hash functions may be highly likely to randomly influence all bits of the hash value when the window is rolled from one time in the input data stream to the next.

[0032] An example of a selection function which may perform well with random data but fails in real life is H='count the number of non-alphabetic characters (not in [A-Za-z]) within a 30-byte window' coupled with a selection rule 'H=0'. For random data input, H will be zero randomly, with probability (204/256)**30 ~= 0.0011, so one in 908 windows will produce a selection, and an exponential distribution of chunk sizes averaging around 908 bytes is to be expected. Unfortunately, if storing an archive of news clippings, the number of occurrences of 30-letter words is likely to be much rarer than this expectation, and few useful chunk points are likely to result.

[0033] Hash functions that may yield "better" performance on real-world data will tend to have little bitwise correlation between the value at one instant or window and the value at the next instant or window. Desirable hash functions may be highly likely to influence all bits of the hash value when the window is rolled from one time in the input data stream to the next. Desirable hash functions are also likely to affect many bits randomly when a single bit within an input window changes. Similar notions correspond to more conventional definitions of good hash function characteristics, and functions fulfilling these properties may result in good bit-scrambling ability for the purposes of various applications using the present invention. In any case, it may be commonly assumed that such hash functions will perform well even when presented with non-random, real-world data.

[0034] The boxcar function, for example, may be particularly bad in scrambling the least significant bits of its hash value over bytes of input, the least significant bit being trivially related to only the parity of bit 0 of the input data windows. However, the speed with which the boxcar function may evaluate a hash for successive windows of input data may make it particularly attractive for use in the present invention. Furthermore, the selection criterion for the boxcar hash may be modified in some embodiments to avoid usage of bit zero to accomplish the selection function. The companion patent mentioned above, U.S. Patent Application titled METHODS AND SYSTEMS FOR QUICK AND EFFICIENT DATA MANAGEMENT AND/OR PROCESSING provides an example of a large dataset for which the application's performance goal of speed and duplicate elimination was greatly improved by the present invention: a large increase in speed was measured, while the amount of duplicate elimination was verified on a 1.1 Terabyte real-life dataset to be virtually identical to that obtained with several embodiments using, for example, Rabin fingerprinting or multiplicative linear congruential generator (MLCG) hashes only to accomplish content-defined chunking.

[00035] Referring to Fig. 1, an exemplary data management or processing system 100 using multi-mode data boundary point/data block determination is provided, according to at least one embodiment of the present invention. In this exemplary system, input data 110 may be provided to a Faster Boundary Point/Data Block Window Module 120. This input data 110 may be, for example, a sequence of files or packets of data in binary 1's and 0's that may need to be processed, transmitted or stored in one or more electronic devices such as a microprocessor, a storage disk, a memory, a computer system, etc. The module 120 may be able to more quickly process the determination of hash, cut, boundary or break points in the input data 110 because it may use a method, for example a boxcar summation process or function, that takes less processing time than traditional processes for determining good hash, cut, boundary or break points. Although, module 120 might not provide the best selection of hash, cut, boundary or break points in the data, its speed will make up for the lesser quality hash, cut, boundary or break points (e.g., may result in less duplicate data being identified). In various embodiments, the module 120 may pre-select a subset of all possible hash, cut, boundary or break points. Some exemplary easily rollable hash functions may include a boxcar sum function, an successive multiply-by-constant and add-byte-value (MLCG) function, a rolN-xor function, etc. The Faster Boundary Point/Data Block Window Module 120 may be coupled to a Slower Faster Boundary Point/Data Block Window Module 130. Module 130 may take relatively more processing time per iteration than module 120, however, it is able to make better possible hash, cut, boundary or

break point determinations (e.g., may result in more duplicate data being identified). In various embodiments, module 130 may utilize a CRC32c function, a Rabin fingerprint function, a SHA-1 function, etc., that may incorporate rolling or non-rolling windows. A table of various cut, break, hash, block point functions is illustrated in Table I below, showing various speeds achieved during simulation.

name	pseudo-code (non-rolling)	speed MB/s
boxcar	hash += b	360
MLCG	hash = hash * A + b	270
rolN-xor	hash = ROL(hash,N) ^ b	280
rolN-xor[]	hash = ROL(hash,N) ^ A[b]	175
xor	hash ^= A[b]	170
xAdler	s1 +=b; s2 +=s1 ; hash=s1 ^s2 ; hash^=hash> >6;	160
Rabin	hash = ((hash< <8)|b) ^ A[hash> >N]	145

Table I

[0036] Some simple Hash Functions are provided above in Table I. Hash is assigned a constant initial value. For each byte in the window, hash is modified by adding the next byte value b. The hash functions listed here have fast rolling versions. Here A and N are hash-specific constants and ROL is a rotate-left operation. The xAdler hash is a simplified version of the Adler(/Fletcher) hash. Speed values are representative of fairly optimized code with a maximal amount of compile-time constants, and should be viewed as a reasonable indication of the hash function speed. The speed tests measure rolling the hash and checking for terminal zero bits on several hundred megabytes of in-memory random data. As can be observed in Table I, algorithm speeds roughly follow the number of external lookups required, with boxcar, MLCG, and rolN-xor hashes being the fastest, with zero table lookups. The rolling versions (see below) or xor, rolN-xor[] and Rabin hashes require two lookups of form A[b].

[0037] Specific versions of C++-like code for various exemplary non-rolling and corresponding rolling versions of hash functions are illustrated below. In the exemplary code, *ws* is the window size and *buf* may be an input data buffer of bytes. The function *calc(buf)* is a non-rolling calculation of checksum *csum*, while *roll(buf)* is a rolling version that will update a checksum whose value is *calc(buf)* to the value at *calc(&buf[1])* in fewer operations. It may be assumed that a global table of constant random numbers may be available as *rndTable[256]* for the algorithms requiring pre-translation of input bytes into more random 32-bit quantities.

Several algorithms may roll more quickly if the window size is a compiler-time constant. Actual code may use various optimizations, such as compiler optimization options, to achieve faster speed.

[0038] An exemplary Boxcar C++-like pseudocode for non-rolling *calc* and rolling *roll* routines is shown below.

```
1 void Boxcar::calc( unsigned char const* const buf ) {
2    csum = std::accumulate( buf, buf+ws, 0xffffc03fU );
3 }
4 void Boxcar::roll( unsigned char const* &buf ) {
5    csum += buf[ws] – buf[0];
6    ++buf;
7 }
```

[0039] An example of MLCG C++-like pseudocode for non-rolling *calc* and rolling *roll* routines, using a multiplicative linear congruential generator module 2^{32}, is shown below. With this choice, modulo operations may not be required on 32-bit machines as overflow of 32-bit arithmetic operations perform this function implicitly.

```
1 MLCG::MLCG( uint32_t windowsize )
2    : ws( windowsize )
3    , init( 4329005U ) // seed value
4    , mult( 741103597U ) // a large prime for pseudorandom sequences
5    , mult_ws( 1UL ) // mult^ws
6    , magic( init * (1U–mult) ) // precalc. for rolling version
7 {
8    for(uint32_t i=0U; i<ws; ++i) mult_ws *= mult;
9    // (mult_ws is actually calculated more efficiently)
10   magic *= mult_ws;
11 }
12 void MLCG::calc( unsigned char const* const buf ) {
13    csum = init;
14    for( unsigned char const *p=buf, *pEnd=&p[ws]; p!=pEnd; ++p){
15       csum = csum * mult + *p;
16    }
17 }
18 void MLCG::roll( unsigned char const* &buf ) {
19    csum = csum * mult – mult_ws * buf[0] + buf[ws] + magic;
20    ++buf;
21 }
```

[0040] An exemplary rolNxor C++-like pseudocode for non-rolling *calc* and rolling *roll* routines, using a rotate-left function *rol32* by 5 bits and assuming a 64-byte window for simplicity (so that no fixups rol operations are needed for the *roll* functions, is show below.

```
1  RolNxor::RolNxor(uint32_t  windowsize, uint32_t rolbits)
2    : ws ( windowsize )
3    , N( rolbits )
4    , M( ((ws-1)*N) & 0x1fU )
5    , init( 0x356a356aU ) // some constant value
6  {}
7  void RolNxor::calc( unsigned char const*  const buf) {
8    csum = init;
9    for( unsigned char const  *p=buf, *pEnd=&p[ws]; p!=pEnd; ++p){
10       csum = rol32 (csum,N)  ^  *p;
11   }
12 }
13 void RolNxor::roll( unsigned char const*  &buf ) {
14    csum = rol32(csum^ rol32(buf[0],M), N) ^ buf[ws];
15    ++buf;
16 }
```

[0041] An exemplary rolNxor[] C++-like pseudocode for non-rolling *calc* and rolling *roll* routines, using a rotate-left function *rol32* by 5 bits and assuming a 64-byte window for simplicity (so that no fixups rol operations are needed for the *roll* function), is show below.

```
1  RolNxorTable::RolNxorTable(uint32_t  windowsize, uint32_t rolbits)
2    : ws ( windowsize )
3    , N( rolbits )
4    , M( ((ws-1)*N) & 0x1fU )
5    , init( 0U ) // some constant value
6  {}
7  void RolNxorTable::calc( unsigned char const*  const buf) {
8    csum = init;
9    for( unsigned char const  *p=buf, *pEnd=&p[ws]; p!=pEnd; ++p){
10       csum = rol32 (csum,N)  ^ rndTable[ *p];
11   }
12 }
13 void RolNxorTable::roll( unsigned char const*  &buf ) {
14    csum = rol32(csum ^ rol32(rndTable[buf[0]],M), N) ^ rndTable[buf[ws]];
15    ++buf;
16 }
```

[0042] An exemplary xor hash C++-like pseudocode for non-rolling *calc* and rolling *roll* routines, identical to rolNxor[] except without the rotation operation, is shown below.

```
1 void XorTable::calc( unsigned char const* const buf ) {
2    csum = init;
3    for( unsigned char const *p=buf, *pEnd=&p[ws]; p!=pEnd; ++p){
4       csum ^= rndTable[*p];
5    }
6 }
7 void XorTable::roll( unsigned char const* &buf ) {
8    csum ^= rndTable[buf[0]] ^ rndTable[buf[ws]];
9    ++buf;
10 }
```

[0043] An exemplary xAdler C++-like pseudocode for non-rolling *calc* and rolling *roll* routines, using for example, a modified version of the Adler (/Fletcher) hash with no table lookup, is shown below. The routines demonstrate a *roll* function that rolls internal state, while producing a checksum value in another distinct step, provided by the *scramble* function. Alternatively, *rndTable[]* could have been used, in conjunction with a more standard Adler algorithm.

```
1 xAdler::xAdler( uint32_t windowsize )
2  : ws( windowsize )
3  , magic( 1493U ) // to increase range of affected bits slightly,
4                   // value is related to expected size of ws
5  , s1( 0U ) // s1 and s2 are internal state
6  , s2( 0U ) // used to produce a checksum
7  {}
8 // hash is some way to generate csum from the internal state
9 uint32_t scramble( uint32_t x, uint32_t y ) {
10    x ^= y;
11    x ^= (x>>6);
12    return x;
13 }
14 void xAdler::calc( unsigned char const* const buf ) {
15    s1 = s2 = 0U;
16    for( unsigned char const *p=buf, *pEnd=&p[ws]; p!=pEnd; ++p){
17       s1 += *p + magic;
18       s2 += s1;
19    }
20    // add a final bit-scramble
21    csum = scramble(s1,s2);
22 }
23 void xAdler::roll( unsigned char const* &buf ) {
24    s1 += buf[ws] - buf[0];
25    s2 += s1 - ws * ( buf[0] + magic );
26    ++buf;
27    csum = scramble(s1,s2);
```

```
28 }
```

[0044] An exemplary Rabin C++-like pseudocode for non-rolling *calc* and rolling *roll* routines for a simple implementation of a CRC is shown below. Tables of constants T[256] and U[256] may be calculated following, for example, A. Broder, Sequences II: Methods in Communications, Security, and Computer Science pp. 143_152 (1993).

```
1 Rabin::Rabin( uint32_t windowsize, uint32_t poly )
2   : ws( windowsize )
3       , p( poly ) // The irreducible generating polynomial
4 {
5       // precompute tables T[256] and U[256] of constants related
6       // to a generating polynomial. T describes how to append
7       // a new byte quickly. U describes how to add/remove a
8       // leading byte quickly (U depends on the window size).
9 }
10 // adding a byte modifies the old crc value in this way:
11 uint32_t append8 ( uint32_t crc, unsigned char byte ) {
12      return T[ ((crc>>24) ^ byte) & 0xff ] ^ (crc << 8);
13 }
14 void Rabin::calc( unsigned char const * const buf ) {
15   uint32_t csum=0ULL;
16   for( unsigned char const *p=buf, *pEnd=&p[ws]; p!=pEnd; ++p){
17     csum = append8( csum, *p );
18   }
19 }
20 void Rabin::roll( unsigned char const* &buf ) {
21    csum = append8( csum ^ U[buf[0]], buf[ws] );
22 }
```

[0045] When the simple hash functions in Table I are used in chunking algorithms, the actual speed may be substantially less, because data may be supplied in small chunks and extra buffer management may be needed. Furthermore, a chunking application may also typically require a strong hash function to be calculated for the entire chunk.

[0046] Further, in various embodiments the methods used by module 120 and module 130 may be content-defined hashing and output Content Defined Block/Chunk Points 140. In general, rolling window functions are faster than non-rolling window functions. As mentioned earlier, in general, content based hashing techniques are better at identifying more duplicates than file or fixed block size hashing techniques. In any case, the content defined block/chunk points 140 may be used to improve the processing time while maintaining a reasonable quality level for processing the input data. In various embodiments, these content defined block/chunk points 140

may be used for identifying duplicate data, improving encryption, improving data transmission, etc. It should be noted that although module 130 may be fed by the output of module 120, the processes performed in these modules may occur to some extent in parallel or simultaneously (e.g., use in dual processor systems). Module 130 may in some embodiments have direct recourse, additionally, to input data 110. For example, if it is determined that exceptionally the module 120 has failed to meet predetermined criteria, module 130 could embark upon a presumably slower and less desirable process of processing the input data via an alternate mechanism for content-defined chunking, as indicated by the dashed line 150. Such an alternate mechanism may be provided by a technique using, for example, known functions to determine a set of chunk points to be used as output for step 140.

[0047] Referring to Fig. 2, an exemplary data management or processing method 200 using multi-mode data boundary point/data block determination is provided, according to at least one embodiment of the present invention. In this embodiment, at step 210, a data stream or data is provided as an input to the process. Next, at step 220, pre-selection of boundary points/data chunk windows is performed. In various embodiments, the pre-selection may be performed using a fast hash function. Some exemplary fast hash functions may include a boxcar sum function, a rol-5-xor function, a successive multiple-by-constant and add-byte-value function, etc. Then, at step 230, a refined subset of boundary points/data chunk windows may be identified using a slower but better deterministic function, for example, a quality hash function that identifies hash, cut, boundary, or chunk points that will maximize a preferred characteristic such as identification of duplicate data in the input data. Some exemplary processes for slower higher quality hashing may include, for example, a Rabin fingerprint function, a SHA-1 function, etc., that may incorporate rolling or non-rolling windows. Finally, in step 240, the process may output a content defined boundary point(s)/data chunk windows that may be used in determining, for example, hash values for determining data duplication and/or duplication elimination in some of the embodiments of the present invention.

[0048] Referring to Fig. 3, an exemplary conceptual illustration 300 of data management or processing using multi-mode data boundary point determination system(s) and method(s) operates is provided, according to at least one embodiment of the present invention. In this illustration, it is shown that the first faster selection function (305) and the second slower selection function (310) may operate together to more quickly identify (processing more megabytes per second) the desirable hash, cut, boundary or chunk points and/or windows 340 from a large set of possible data points/windows 320. The large set of possible hash, cut,

boundary or chunk points and/or windows 320 may include, for example, data points or chunks A (319a), B (312a), C (317a), D (311a), E (315a), F (316a), G (321a), H (313a), I (314a), and J (318a). The first faster selection function 305 may process the large set of possible data points/windows 320 to quickly select a reasonably good pre-selected sub-group of desirable hash, cut, boundary or chunk points and/or windows 330. In this example, data points and/or data windows that are pre-selected into sub group 330 include data points and/or windows B (312b), D (311b), E (315b), H (313b), and I (314b). The processing speed of the first faster selection function 305 may be, for example, approximately 2 times faster than the second slower selection function 310. However, the data hash, cut, boundary, or break points and/or data windows it selects may have a lesser performance or may be less desirable than those that would be selected by the second slower selection function 310, if only the second slower selection function 310 analyzed the data set 320. On the other hand, the first faster selection function 305 may be capable of providing a reasonable number of data hash, cut, boundary, or break points and/or data windows 320 so that statistically these points are highly likely to contain many of the data break points and/or windows that have the highest quality of most desirable characteristics. Thus, the second slower selection function 310 may process the data points and/or windows including B (312b), D (311b), E (315b), H (313b), and I (314b) and select group 340 including data points and/or windows including B (312c) and D (311c) which have high quality characteristics, for example, identify most of the desirable data hash, cut, boundary, or break points and/or data windows. This process may go on repeatedly until all the data in a data set or data stream has been processed.

[0049] Referring now to Fig. 4, an exemplary method for data management or processing 400 that uses multi-mode data boundary point determination for identification of previously encountered data in a data set or stream is provided, according to at least one embodiment of the invention. First, in step 405 input data may be provided for processing. This input data 405 may be, for example, a sequence of files or packets of data in binary 1's and 0's that may need to be processed, transmitted or stored in one or more electronic devices such as a microprocessor, a storage disk, a memory, a computer system, etc. Next, in step 410, a first selection function may be performed. The first selection function may be a faster selection function that identifies where one or more logical break or boundary points may be made in the data set or data stream based on a predetermined data processing objective. In various embodiments, the first selection process 410 may be a reasonably fast hashing function for identifying hash, cut, boundary or break points in the data set or data stream, but may offer

reduced bit scrambling ability. The first selection may identify a subset of all possible hashing, cut, or break points that may be input to the second selection function. A faster first selection function may be defined as a function that is able to give a checksum faster based on megabytes of input data processed per second. Some exemplary first selection functions 410 may include a boxcar sum function, a rol-5-xor function, a successive multiple-by-constant and add-byte-value function, etc., some of which have been described in more detail above and some are described in more detail below. Then, in step 415, a second selection function may be performed having better performance at selecting the preferred hash, cut, or boundary break points in the data set or data stream. In various embodiments, this second selection function may be slower than the first selection function but provide better discriminating and/or accuracy in identifying the preferred (based on a desired characteristic, for example, duplication identification and/or elimination) hash, cut, boundary or break points in the data set or data stream. Some exemplary second selection function 415 may include a CRC32c function, a Rabin fingerprint function, etc, as described above and may be described in more detail below. In some embodiments, the second selection function may only consider the subset of hash, cut, boundary or break points in the data set or data stream identified by the first selection criteria. In this way, compared to a single-mode selection technique the process may be performed more quickly and take less processing time.

[0050] In other embodiments, a second selection function may notice that the output of the first selection function fails to satisfy predetermined criteria and may provide (within the second selection module) an alternate mechanism to produce the chunk points. Such an alternate mechanism may operate much more slowly, and may preferably operate only in exceptional circumstances. For example, if it is noticed that every window of input is selected by the first selection function, leading to a large number of chunks of size 1, or that the first selection function is producing chunks at a statistically improbably low rate (compared to expectations based on random inputs), then the second selection function may decide that the first selection function was insufficient. In this case, the alternate policy invoked may have recourse to the original data, (shown as dashed line 417) to produce final chunk or break points using, for example, a chunking technique using a hash function of better bit-scrambling ability than provided by the first (fast) selection function.

[0051] Next, one or both of steps 420 or 435 may be performed. At step 420, the most recent underlying data contained in the chunk may be identified along with previously encountered underlying data so that they may be compared. At step 420, the most recent

underlying data may be compared to all previously encountered underlying data to see if there are any matches. For example, in various embodiments new underlying data may be compared with previously stored or transmitted underlying data. If, at step 420, there is a match, then at step 425 the most recent underlying data that matched previously encountered underlying data is not outputted. For example, in various embodiments the duplicate may be detected. In that case, then the duplicate underlying data may be eliminated. On the other hand, if at step 420 the most recent underlying data does not equate to any of the previously encountered underlying data, then at step 430 the new underlying data may be outputted (e.g. stored or transmitted).

[0052] At step 435, a value for the selected data chunk(s) or block(s) may be generated that is indicative of the underlying data contained in the data chunk(s) or block(s). For example, a hash value may be generated to represent the particular data contained in the data chunk(s) or block(s). One exemplary function for generating the value may be, for example, a SHA-1 function. Next, at step 440, the most recently generated value from step 435 may be identified along with previously generated values so that they may be compared. The most recently generated value is compared to all previously generated and encountered values to see if there are any matches. For example, in various embodiments a newly generated hash value may be compared with previously generated and/or stored or transmitted hash values. If, at step 440, there is a match, then at step 445 the most recently generated value that matched a previously generated value may not be outputted. For example, in various embodiments the duplicate may be detected. In that case, then the duplicate data and its generated value may be eliminated, and a location indicator related to the prior stored hash value showing which location data is stored at so as to regenerate the eliminated data. On the other hand, if at step 440 the newly generated value, for example a hash value, does not equate to any of the previously encountered values, then at step 450 the newly generated value (e.g., hash value) and location indicator may be outputted and the data which it represents may be outputted (e.g. stored or transmitted). In the case of a hash value, it may be stored in a hash table for future reference.

[0053] Referring now to Fig. 5, an exemplary hashing technique using content-defined data blocks or chunks is provided, according to at least one embodiment of the present invention. Content-defined hashing functions operate by using a predetermined rule for determining at what point in a data set or stream that a hash, cut, boundary, or chunk point is to be made. This rule, for example X number of consecutive 0's or 1's, when applied to the data set or data stream will result in random hash, cut, boundary, chunk, or break points and variable length data chunks or data windows (e.g., C_1 525, C_2 530, C_3 535, and C_4 540 are each a different length) that may be

hashed to produce a hash value. As another example, the content-defined data chunks may be based on a Rabin function in which fingerprints (unique representations of the underlying data based on a predetermined calculation) for each sliding window frame may be calculated. If the value of this calculation matches, for example, the x least significant bits of a constant, then a hash, cut, boundary, chunk, or break point may be determined and marked (e.g., selected) for later use in hashing a data block or window. The diagram 500 of Fig. 5 shows a data stream 502 that has been partially broken into chunks C_1 525, C_2 530, C_3 535, and C_4 540 using hash, cut, boundary, chunk, or break points. This may be accomplished by using a sliding window approach and adding a new byte 510 into a sliding window may be accomplished in two parts. The window area of the data steam may be portion 520, that may be for example a 64-byte window. At each step in the sliding window process, the oldest position of the sliding window may be subtracted and a new one added. The value for the oldest byte b_{i-64} 505 in a first window 515 may be subtracted from the fingerprint and the value of the new byte b_i 510 may then be added to the data fingerprint to form a second window 518. This approach may be made possible by having the fingerprint commutative over addition as in the simple boxcar function, or by more complicated mechanisms specific to the window size and the fingerprint calculation chosen for the sliding window frames. The process will continue over time as the new data 555 is provided to the process over time as shown by arrow 560. When subtracting the oldest bytes over time, speed and performance of the calculation may be improved by using pre-computed tables. As noted above, this Rabin fingerprint may be used as a second selection function to find better performance break points by virtue of its better bit-scrambling ability, and may be slower than the first selection function. However, there are alternative functions which may also have excellent bit-scrambling ability and may be used as the second selection function. For example, the SHA-1 function may be used, or specific versions of Rabin fingerprinting, such as the CRC32c hash, may be used.

[0054] Referring now to Fig. 6, an exemplary first selection function using a boxcar sum technique is illustrated 600, according to at least one embodiment. In this example, a boxcar sum function is used to determine hash, cut, boundary, chunk, or break points in a data set or data stream. In this case, again a sliding window approach may be used. Further, to make the process fast, the old data blocks that are subtracted and the new data blocks that are added may simply be a predetermined number of bytes or bits of data and the calculation may be simply a sum of adding up all the data values in each of the data blocks such that: Boxcar Sum = $\sum$ N-bytes from i=1 to n. As such, to calculate a new boxcar sum, the earliest group of data sum may be

subtracted and the most recent group of data summation may be added. As shown, a data set or data stream 605 is provide for hashing and is comprised of groups of data a, b, c, d, and e. Data groups a, b, c, d, and e may be of equal bit or byte length, for example, 64 bits. The first window 610 may be made up of data groups a, b, and c. Data group "a" may have a sum total value (e.g., the sum of each value of each of the bits in group a) of, for example, 15. Data group "b" may have a sum total value of, for example, 10. Data group "c" may have a sum equal to 5. Data group "d" may have a sum total value of, for example, 10. Data group "e" may have a sum total value of, for example, 5. Given these values for the various data groups in the data stream or data set, the boxcar sum of window 610 may be 15+10+5 = 30. Then, the next boxcar sum may be calculated as the prior calculated sum minus the earliest group of data a 640, plus the newest group of data d 650: 30-15+10 = 25. Then, the next boxcar sum may be calculated as the prior calculated sum minus the earliest group of data b 6550, plus the newest group of data e 660: 25-10+5 = 20. If the function is set to make hash, cut, boundary, chunk, or break points where the sum is equal to 25, than the hash, cut, boundary, chunk, or break point subset would in this short example includes points 670. As can be seen, this may be a particularly simple and fast way to perform a first selection function for hashing of data, however, the quality of the hash points may be somewhat lower due to the lack of sophistication. As such, the boxcar sum approach can develop a pre-selection subset of potential hash, cut, boundary, chunk, or break points.

[0055] Referring to Figs. 7a and 7b, exemplary illustrations of data windows, blocks or chunks are provided for an exemplary hashing technique using content-defined data blocks or chunks, according to at least one embodiment of the present invention. Fig. 7a shows chunking 700 at various points in the data stream where the fingerprint or hash value for various windows equals a predetermined value. Various functions may be applied with rolling, moving or sliding windows. This example may be indicative of either a faster or a slower selection function involving a rolling or sliding window, such as the boxcar sum or the Rabin fingerprint functions, and the criterion of equality to a predetermined value could be replaced with another criterion having some desired probability of occurrence. In this example, various windows (705a – 735a) in a sliding window function are illustrated in bold and organized as a plurality of groups of data in the data set or stream having data groups a, b, c, d, e, f, g, h, i, and j. At various points in time, a new window is formed and evaluated against a predetermined value to determine if a hash, cut, boundary, chunk, or break point should be made. In this example, the predetermined value results in a hash, cut, boundary, chunk, or break point as identified at the end 740 of the second window 710a. As such, a hash value may be made of the data in the sum of data blocks a, b, c,

and d. Then after performing hashing at two more sliding time windows, third window 715a and fourth window 720a, a hash, cut, boundary, chunk, or break point is identified at the end 750 of the fourth window 720a. As such, a hash value may be made of the data in the sum of data blocks e and f. Then after calculating the value of the data in the windows of two more sliding time windows, fifth window 725a and sixth window 730a, a hash, cut, boundary, chunk, or break point is identified at the end 760 of the sixth window 730a. As such, a hash value may be made of the data in the sum of data blocks g and h. As shown, the process continues to seventh window 735a without finding another value match and resulting hash, cut, boundary, chunk, or break point.

[0056] Fig. 7b shows chunking 765 at various points in the data stream where the fingerprint or hash value for various windows equals a predetermined value. This example illustrates how the hash, cut, boundary, chunk, or break point will change due to the introduction of a new data group z has been inserted within the earlier indicated data set or stream having data groups a, b, c, d, e, f, g, h, i, and j. As can be seen from this illustration, the introduction of a new data group may result in the elimination of one hash, cut, boundary, chunk, or break point that occurred before in the fourth window 720b; in this case no hash, cut, boundary, chunk, or break point is selected at 780. In this example, various windows (705b – 735b) in a sliding window function are illustrated in bold and organized as a plurality of groups of data in the data set or stream having data groups a, b, c, d, e, f, g, h, i, and j, with new data group z inserted between e and f. At various points in time, a new window is formed and evaluated against a predetermined value to determine if a hash, cut, boundary, chunk, or break point should be made. In this example, the predetermined value results in a hash, cut, boundary, chunk, or break point is identified at the end 770 of the second window 710b. As such, a hash value may be made of the data in the sum of data blocks a, b, c, and d. Then after performing hashing at two more sliding time windows, third window 715b and fourth window 720b, the new data group z is encountered and a hash, cut, boundary, chunk, or break point is not identified at the end 780 of the fourth window 720b. Then after calculating the value of the data in the windows of three more sliding time windows (because the sliding window covers three data groups), fifth window 725b, sixth window 730b, and seventh window 735b, a hash, cut, boundary, chunk, or break point is identified at the end 790 of the seventh window 735b. As such, a new hash value may be made of the data in the sum of data blocks e, z, f, g and h. As shown, the insertion of a new data group z may change the output of the selection function at up to 3 locations because the change in the underlying data is limited to only a single data group and the sliding window includes three data

groups. In Fig. 7, it is assumed that the effect was to make all three affected windows inactive during the selection criteria "equal to a predetermined value". Based on this example for a sliding window functions, one can appreciate how a change in data or a different data set can result in varying hash, cut, boundary, chunk, or break points. In various embodiments, once a hash, cut, boundary, chunk, or break point is identified, in the case of a faster selection function then a subset of hash, cut, boundary, chunk, or break points may be determined using, for example, a boxcar sum function with a rolling, sliding or moving window as described above, then a slower but better performing second selection function may be used. For example, a Rabin fingerprint function may then be performed using the subset of hash, cut, boundary, chunk, or break points pre-selected as its starting point, so as to reduce processing time while keeping a reasonable quality for hash, cut, boundary, chunk, or break points selection.

[0057] A particularly good example of a hashing function for the second selection function is the CRC32c. The CRC32c hashing function is a particular implementation of a Rabin fingerprint and may be found in, for example, J. Stone, R. Stewart and D. Otis, "Stream Control Transmission Protocol (SCTP) Checksum Change", The Internet Society, RFC 3309, Sept. 2002, pages 1-17. This may illustrate the CRC32c standard and gives a sample implementation that describes this hash function. The CRC32c recommendation actually is a specific implementation of a hash based on polynomial division, so it may somewhat be considered a particular instance of a Rabin hash. The CRC32c is typically a 32-bit code and may specify a specific polynomial and method, more or less. These functions may be modified for C++ language and may be modified to supports 64-bit polynomials. The code for the second function may be generic (e.g., off-the-shelf) or hardwired. If it is "generic" it may turn out to be somewhat slower.

[0058] As noted above, in various embodiments the present invention may be used for data identification and duplicate data elimination. As such, subsequent to determining preferred hash, cut, boundary, chunk, or break points, a hash value for the determined chunk of data may be produced and compared with previously stored has values. In various embodiments, the present invention may be particularly applicable to data duplication elimination. Further, as also noted above, the present invention may be equally applicable to various electronic systems including wireless networks, the Internet, intranets, computer systems and networks with a string of data that is processed, stored, and/or transmitted and the use of hashing can prove useful. A description is provided below of some of the various systems upon which the present invention may operate.

[0059] As noted, in various embodiments, the system(s) and method(s) provided herein may be implemented using a computing device, for example, a personal computer, a server, a mini-mainframe computer, and/or a mainframe computer, etc., programmed to execute a sequence of instructions that configure the computer to perform operations as described herein. In various embodiments, the computing device may be, for example, a personal computer available from any number of commercial manufacturers such as, for example, Dell Computer of Austin, Texas, running, for example, the Windows™ XP™ and Linux operating systems, and having a standard set of peripheral devices (e.g., keyboard, mouse, display, printer). Fig. 8 is a functional block diagram of one embodiment of a computing device 800 that may be useful for hosting software application programs implementing the system(s) and method(s) described herein. Referring now to Fig. 8, the computing device 800 may include a processing unit 805, communications interface(s) 810, storage device(s) 815, a user interface 820, operating system(s) instructions 835, application executable instructions/API 840, all provided in functional communication and may use, for example, a data bus 850. The computing device 800 may also include system memory 855, data and data executable code 865, software modules 860, and interface port(s). The Interface Port(s) 870 may be coupled to one or more input/output device(s) 875, such as printers, scanner(s), all-in-one printer/scanner/fax machines, etc. The processing unit(s) 805 may be one or more microprocessor(s) or microcontroller(s) configured to execute software instructions implementing the functions described herein. Application executable instructions/APIs 840 and operating system instructions 835 may be stored using computing device 800 on the storage device(s) 815 and/or system memory 855 that may include volatile and nonvolatile memory. Application executable instructions/APIs 840 may include software application programs implementing the present invention system(s) and method(s). Operating system instructions 835 may include software instructions operable to control basic operation and control of the processor 805. In one embodiment, operating system instructions 835 may include, for example, the XP™ operating system available from Microsoft Corporation of Redmond, Washington.

[0060] Instructions may be read into a main memory from another computer-readable medium, such as a storage device. The term "computer-readable medium" as used herein may refer to any medium that participates in providing instructions to the processing unit 805 for execution. Such a medium may take many forms, including, but not limited to, non-volatile media, volatile media, and transmission media. Non-volatile media may include, for example, optical or magnetic disks, thumb or jump drives, and storage devices. Volatile media may

include dynamic memory such as a main memory or cache memory. Transmission media may include coaxial cable, copper wire, and fiber optics, including the connections that comprise the bus 850. Transmission media may also take the form of acoustic or light waves, such as those generated during Radio Frequency (RF) and Infrared (IR) data communications. Common forms of computer-readable media include, for example, floppy disk, a flexible disk, hard disk, magnetic tape, any other magnetic medium, Universal Serial Bus (USB) memory stick™, a CD-ROM, DVD, any other optical medium, a RAM, a ROM, a PROM, an EPROM, a Flash EPROM, any other memory chip or cartridge, a carrier wave as described hereinafter, or any other medium from which a computer can read.

[0061] Various forms of computer-readable media may be involved in carrying one or more sequences of one or more instructions to the processing unit(s) 805 for execution. For example, the instructions may be initially borne on a magnetic disk of a remote computer(s) 885 (e.g., a server, a PC, a mainframe, etc.). The remote computer(s) 885 may load the instructions into its dynamic memory and send the instructions over a one or more network interface(s) 880 using, for example, a telephone line connected to a modem, which may be an analog, digital, DSL or cable modem. The network may be, for example, the Internet, and Intranet, a peer-to-peer network, etc. The computing device 800 may send messages and receive data, including program code(s), through a network of other computer(s) via the communications interface 810, which may be coupled through network interface(s) 880. A server may transmit a requested code for an application program through the Internet for a downloaded application. The received code may be executed by the processing unit(s) 805 as it is received, and/or stored in a storage device 815 or other non-volatile storage 855 for later execution. In this manner, the computing device 800 may obtain an application code in the form of a carrier wave.

[0062] The present system(s) and method(s) may reside on a single computing device or platform 800, or on multiple computing devices 800, or different applications may reside on separate computing devices 800. Application executable instructions/APIs 840 and operating system instructions 835 may be loaded into one or more allocated code segments of computing device 800 volatile memory for runtime execution. In one embodiment, computing device 800 may include system memory 855, such as 512 MB of volatile memory and 80GB of nonvolatile memory storage. In at least one embodiment, software portions of the present invention system(s) and method(s) may be implemented using, for example, C programming language source code instructions. Other embodiments are possible.

[0063] Application executable instructions/APIs 840 may include one or more application program interfaces (APIs). The system(s) and method(s) of the present invention may use APIs 840 for inter-process communication and to request and return inter-application function calls. For example, an API may be provided in conjunction with a database 865 in order to facilitate the development of, for example, SQL scripts useful to cause the database to perform particular data storage or retrieval operations in accordance with the instructions specified in the script(s). In general, APIs may be used to facilitate development of application programs which are programmed to accomplish some of the functions described herein.

[0064] The communications interface(s) 810 may provide the computing device 800 the capability to transmit and receive information over the Internet, including but not limited to electronic mail, HTML or XML pages, and file transfer capabilities. To this end, the communications interface 810 may further include a web browser such as, but not limited to, Microsoft Internet Explorer™ provided by Microsoft Corporation. The user interface(s) 820 may include a computer terminal display, keyboard, and mouse device. One or more Graphical User Interfaces (GUIs) also may be included to provide for display and manipulation of data contained in interactive HTML or XML pages.

[0065] Referring now to Fig. 9, a network 900 upon which the system(s) and method(s) may operate, is illustrated. As noted above, the system(s) and method(s) of the present patent application may be operational on one or more computer(s). The network 900 may include one or more client(s) 905 coupled to one or more client data store(s) 910. The one or more client(s) may be coupled through a communication network (e.g., fiber optics, telephone lines, wireless, etc.) to the communication framework 930. The communication framework 230 may be, for example, the Internet, and Intranet, a peer-to-peer network, a LAN, an ad hoc computer-to-computer network, etc. The network 900 may also include one or more server(s) 915 coupled to the communication framework 930 and coupled to a server data store(s) 920. The present invention system(s) and method(s) may also have portions that are operative on one or more of the components in the network 900 so as to operate as a complete operative system(s) and method(s).

[0066] While embodiments of the invention have been described above, it is evident that many alternatives, modifications and variations will be apparent to those skilled in the art. In general, embodiments may relate to the automation of these and other business processes in which analysis of data is performed. Accordingly, the embodiments of the invention, as set forth above, are intended to be illustrative, and should not be construed as limitations on the scope of

the invention. Various changes may be made without departing from the spirit and scope of the invention. Accordingly, the scope of the present invention should be determined not by the embodiments illustrated above, but by the claims appended hereto and their legal equivalents

[0067] All publications, patents, and patent applications cited herein are hereby incorporated by reference in their entirety for all purposes.

CLAIMS

We claim:

1. A method of data management, comprising the steps of:

pre-selecting a portion of a plurality of windows of data in a data stream using a first selecting function; and

selecting a subset of the portion of the plurality of windows pre-selected using a second selecting function.

2. The method of claim 1, wherein the first selecting function is faster at selecting a data block boundary than is the second selecting function.

3. The method of claim 2, wherein the first selecting function is faster at window selection than is the second selecting function.

4. The method of claim 1, wherein the first selecting function includes a boxcar sum function, an MLCG function, or a rolN-xor function.

5. The method of claim 4, wherein the first selecting function is the boxcar sum function and the boxcar sum function is coupled with a selection criterion on the value of the boxcar sum.

6. The method of claim 1, wherein the second selecting function includes a Rabin fingerprint, a SHA-1 function, or a CRC32c function.

7. The method of claim 1, wherein the first selecting function includes a rolling, sliding or moving window function.

8. The method of claim 1, wherein the windows are utilized for defining data groups for hashing and speed of determining break points in the data stream is increased.

9. The method of claim 1, further comprising the step of:

generating a value for a data chunk size determined by one or more chunk points of the subset, wherein the generated value is indicative of underlying data contained in the data chunk.

10. The method of claim 9, further comprising the step of:

comparing the generated value with one or more previously generated values to determine if the generated value equals the one or more previously generated values.

11. The method of claim 10, further comprising the steps of:

determining that there is data duplication present; and

halting further processing of the data in a data chunk determined to have duplicate data.

12. The method of claim 10, further comprising the step of:

storing the generated value if there is no data duplication.

13. The method of claim 1, further comprising the step of comparing underlying data contained in a new window of data or data chunk selected by the second selection function with previously encountered underlying data from previously defined window(s) of data or data chunk(s).

14. The method of claim 13, further comprising the steps of:

outputting the underlying data contained in the new window of data if it does not equal the previously encountered underlying data; and

not outputting the underlying data contained in the new data chunk if it does equal the previously encountered underlying data.

15. The method of claim 1, further comprising the step of generating a value for a window of data or data chunk indicative of underlying data contained in the window of data or data chunk.

16. The method of claim 15, further comprising the step of comparing the generated value with previously generated or encountered value(s).

17. The method of claim 16, further comprising the steps of:

outputting the generated value if it does not equal the previously generated or encountered value(s); and

not outputting the generated value if it does equal the previously generated or encountered value(s).

18. A method of determining boundary points in a data stream for data management, comprising:

pre-selecting a portion of a plurality of boundary points in a data stream using a first selecting function;

selecting a subset of the portion of the plurality of boundary points pre-selected using a second selecting function; and

generating a value for a chunk of data as determined by the subset of the portion of the plurality of boundary points.

19. The method of claim 18, further comprising the step of comparing the generated value with one or more stored value(s), so as to detect a duplicate or store the generated value.

20. The method of claim 19, wherein generating a value is performed by hashing.

21. The method of claim 20, further comprising the step of eliminating duplicate data found by hashing the data contained in a chunk of data defined by the boundary point(s).

22. A data processing system, comprising:

a first selection function module configured to pre-select a first set of data break points or windows; and

a second selection function module configured to select a subset of the pre-selected data break points or windows, wherein the first selection function module processes the data break points or windows faster than the second selection function module.

23. The data processing system of claim 22, wherein the first selection function module includes a boxcar sum function and the second selection function module includes a Rabin function.

24. The data processing system of claim 23, wherein the data processing system is a hashing system and outputs content-defined data block(s) or chunk point(s).

25. A data processing system, comprising:

means for pre-selecting a first set of data break points or windows; and

means for selecting and outputting a subset of the pre-selected data break points or windows, wherein the means for pre-selecting processes the data break points or windows faster than the means for selecting.

26. The data processing system of claim 25, further comprising means for generating a value for each of the data break point(s) or window(s) of the subset.

27. The data processing system of claim 26, wherein the means for pre-selecting a first set of data break points or windows performs a rolling boxcar sum function, the means for selecting and outputting a subset of the pre-selected data break points or windows performs a Rabin function, and the means for generating a value performs a SHA-1 function.

28. The data processing system of claim 27, wherein the generated value is a hash value.

29. The data processing system of claim 25, wherein the data processing system is a hashing system and outputs content-defined data block(s) or chunk point(s).

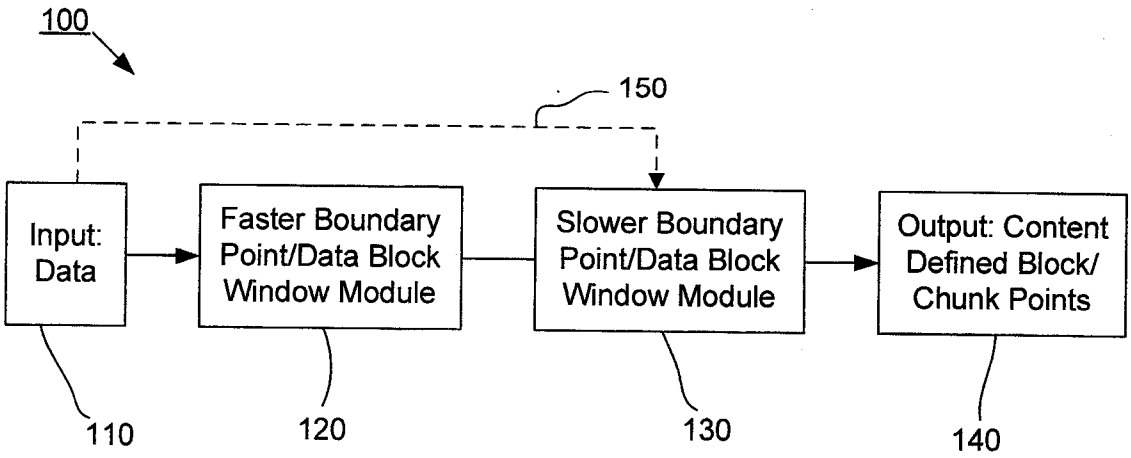

100
150
Input: Data
Faster Boundary Point/Data Block Window Module
Slower Boundary Point/Data Block Window Module
Output: Content Defined Block/ Chunk Points
110
120
130
140

Fig. 1

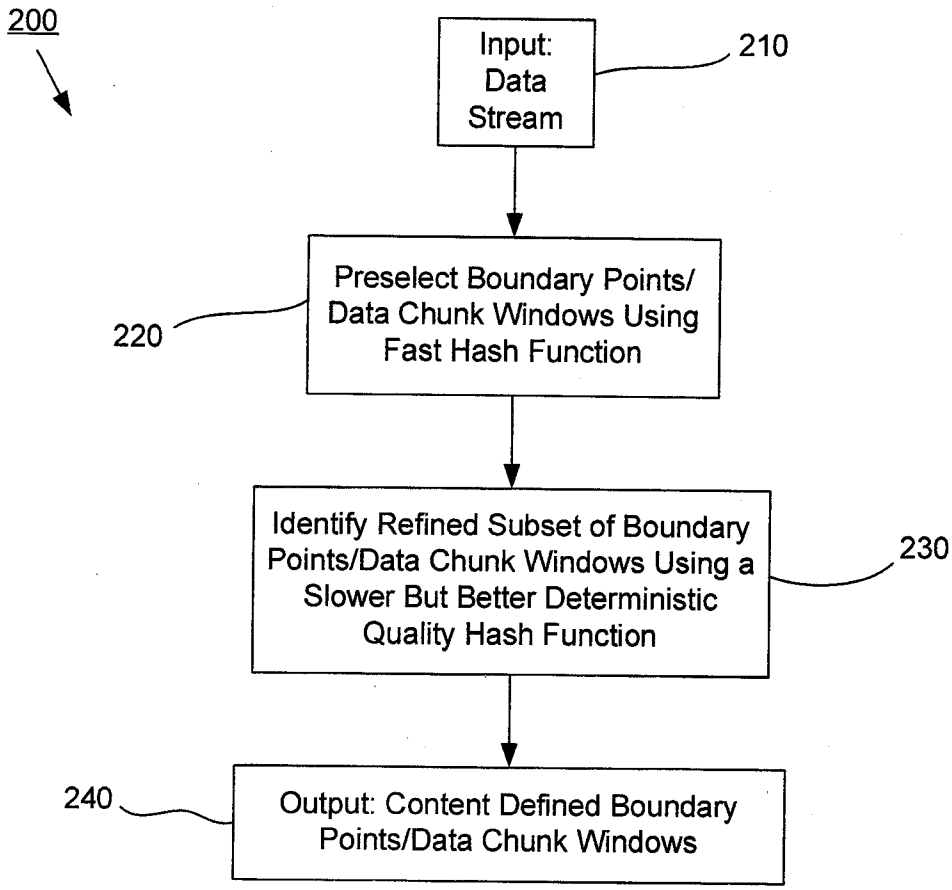

200
Input: Data Stream
210
Preselect Boundary Points/ Data Chunk Windows Using Fast Hash Function
220
Identify Refined Subset of Boundary Points/Data Chunk Windows Using a Slower But Better Deterministic Quality Hash Function
230
Output: Content Defined Boundary Points/Data Chunk Windows
240

Fig. 2

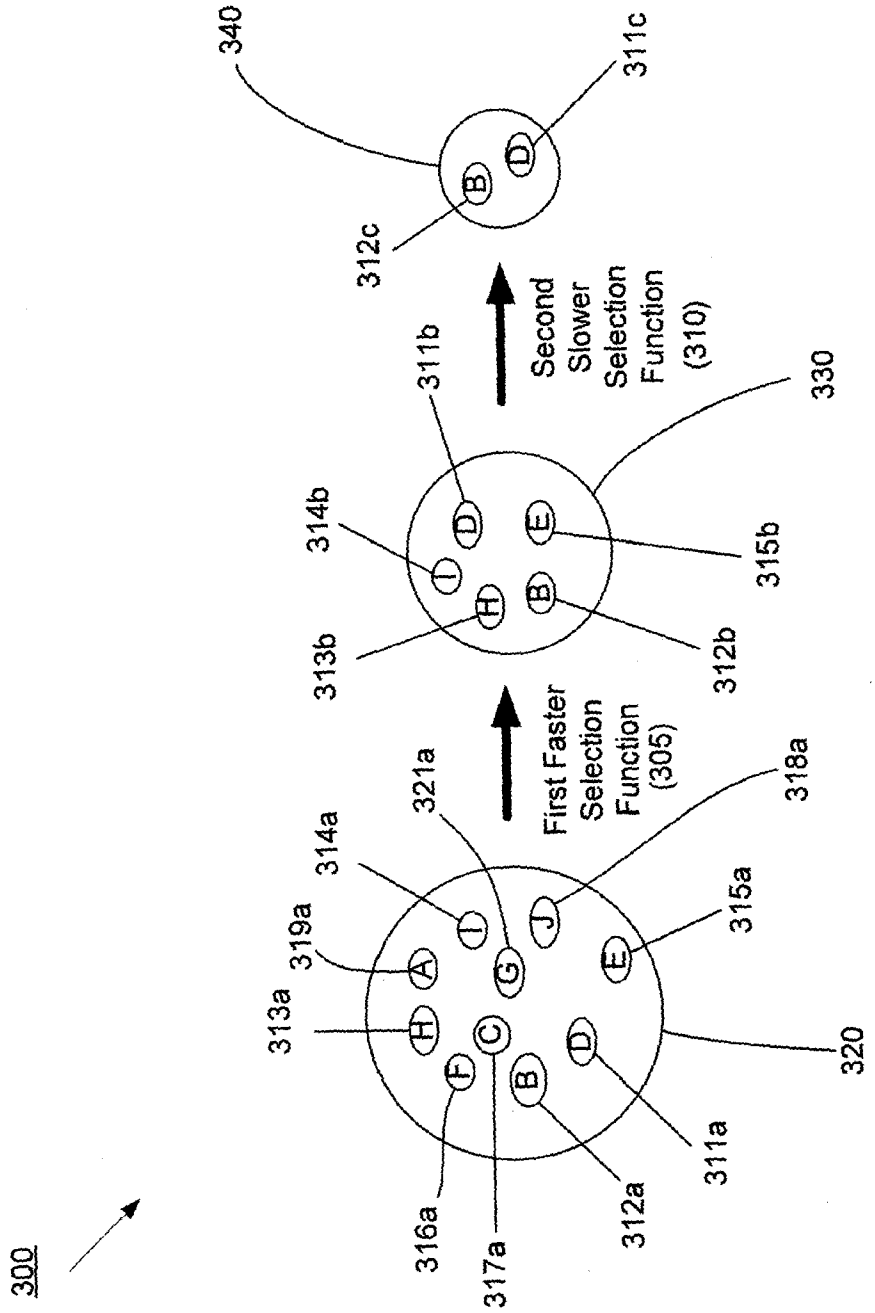

300
340
311c
312c
Second Slower Selection Function (310)
311b
330
314b
315b
313b
312b
First Faster Selection Function (305)
321a
318a
314a
315a
319a
313a
320
316a
317a
312a
311a

Fig. 3

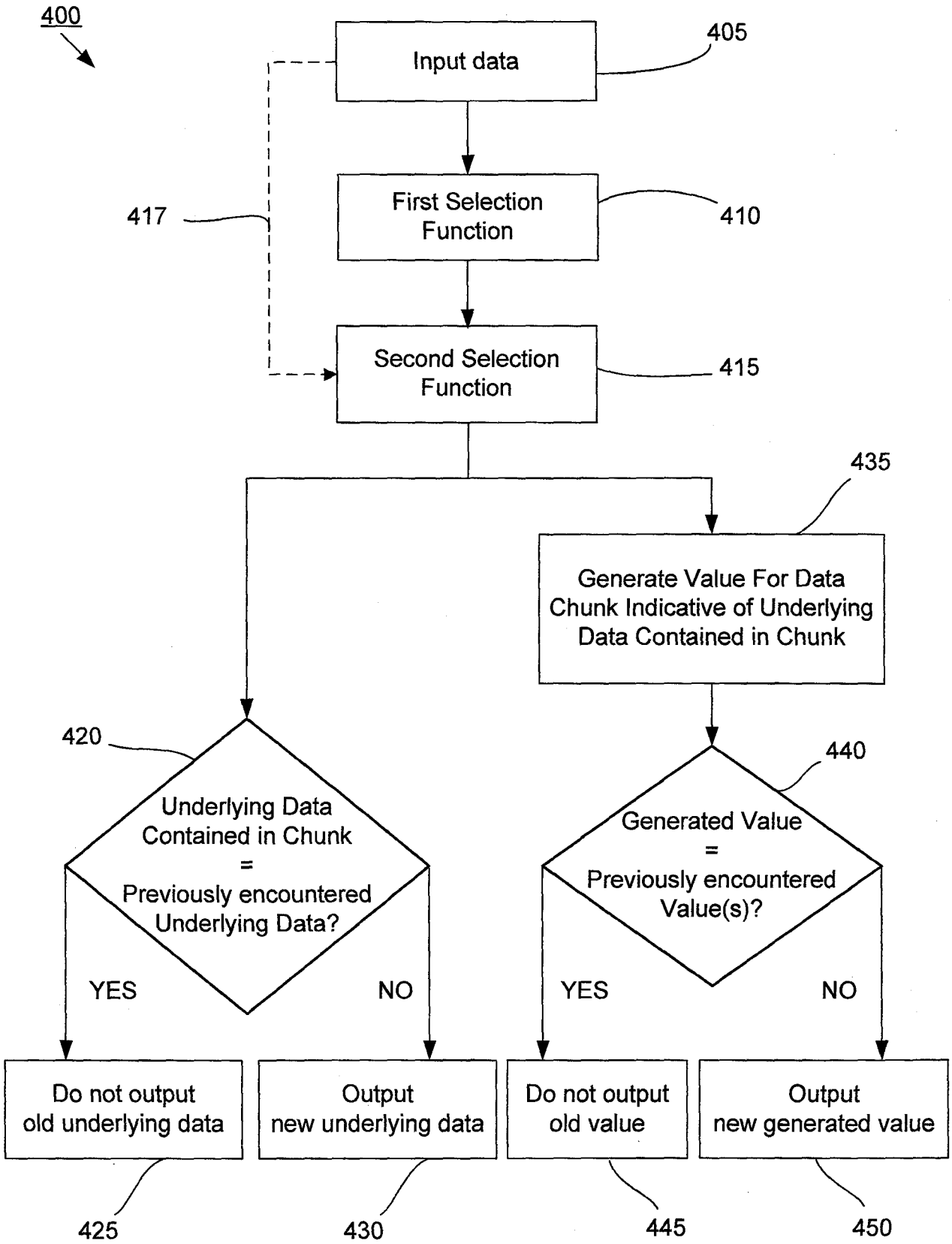

400
Input data
405
First Selection Function
410
417
Second Selection Function
415
435
Generate Value For Data Chunk Indicative of Underlying Data Contained in Chunk
420
Underlying Data Contained in Chunk = Previously encountered Underlying Data?
YES
NO
440
Generated Value = Previously encountered Value(s)?
YES
NO
Do not output old underlying data
Output new underlying data
Do not output old value
Output new generated value
425
430
445
450

Fig. 4

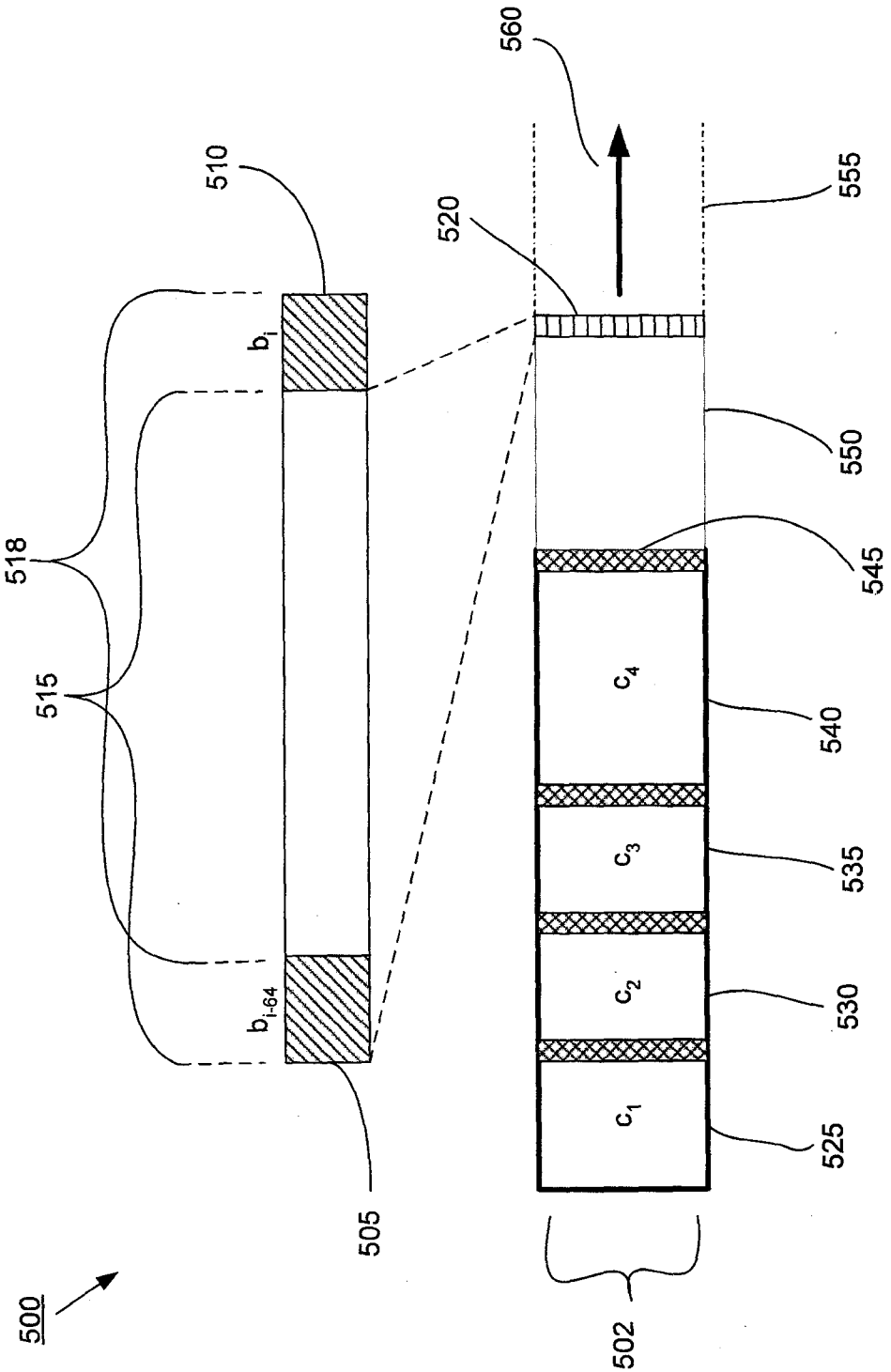

560
510
520
555
b_i
550
518
545
515
c_4
540
c_3
535
c_2
530
c_1
525
b_{i-64}
505
502
500

Fig. 5

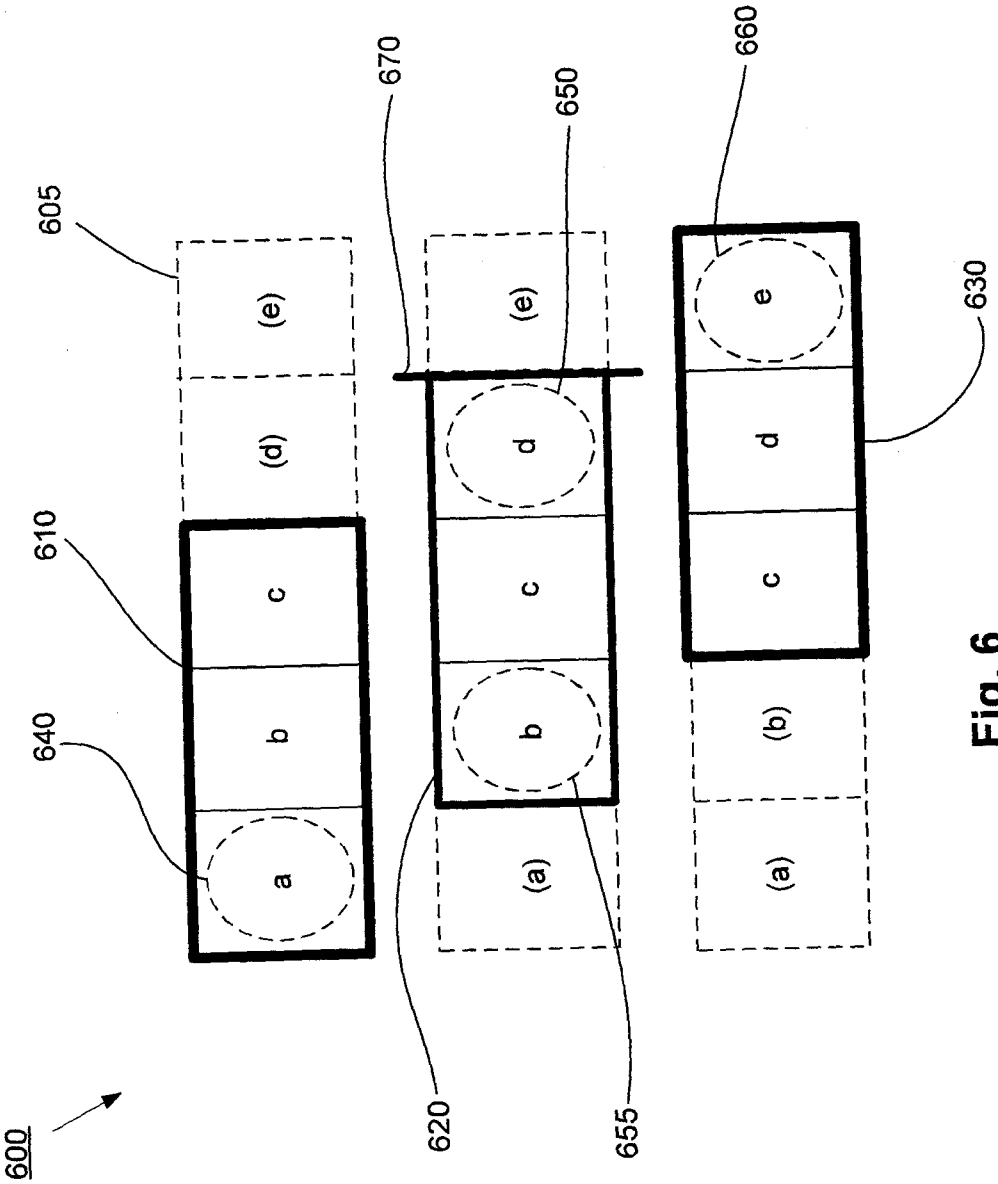


Fig. 6

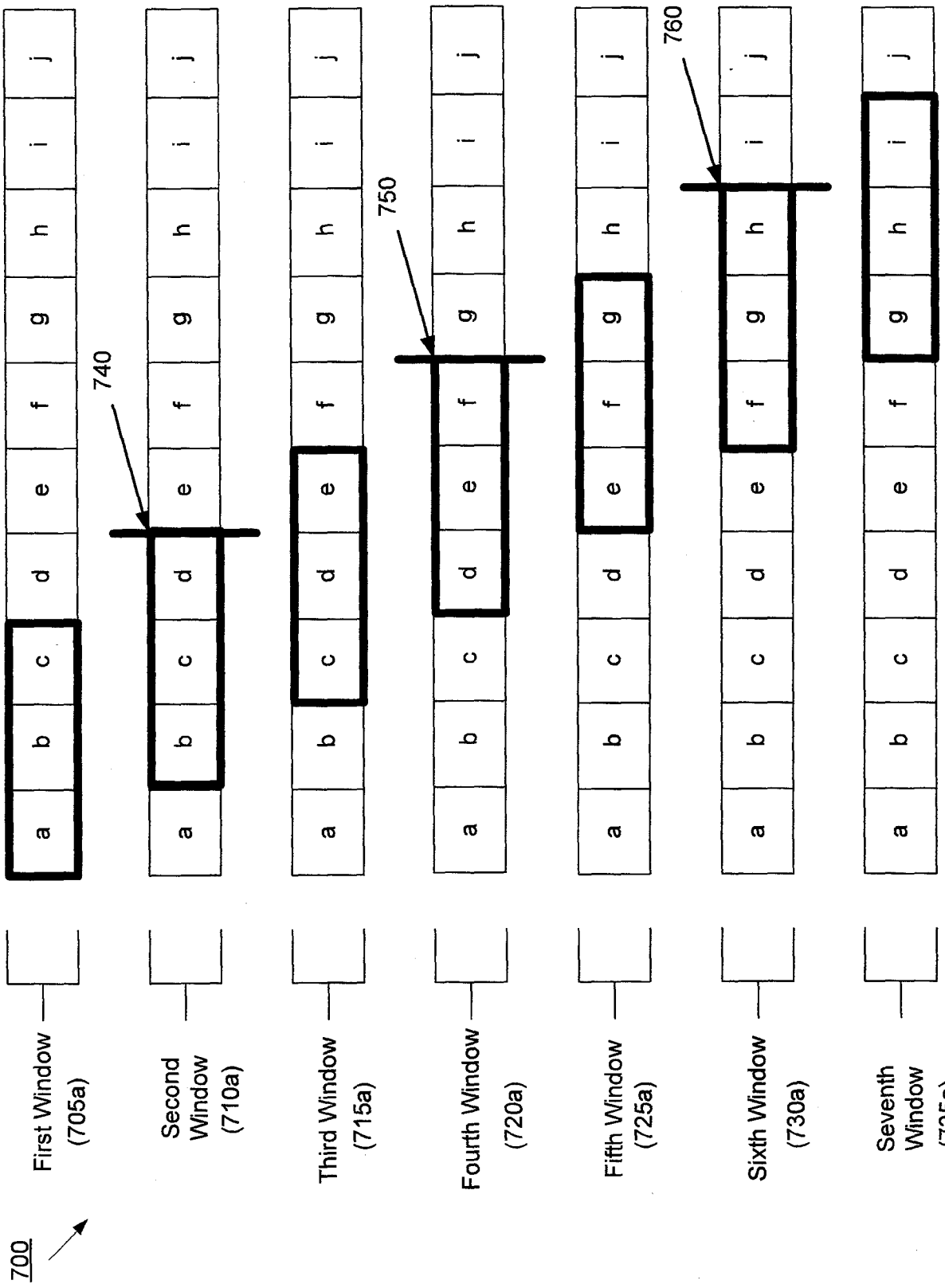

700
First Window (705a)
Second Window (710a)
Third Window (715a)
Fourth Window (720a)
Fifth Window (725a)
Sixth Window (730a)
Seventh Window (735a)
740
750
760
a b c d e f g h i j

Fig. 7a

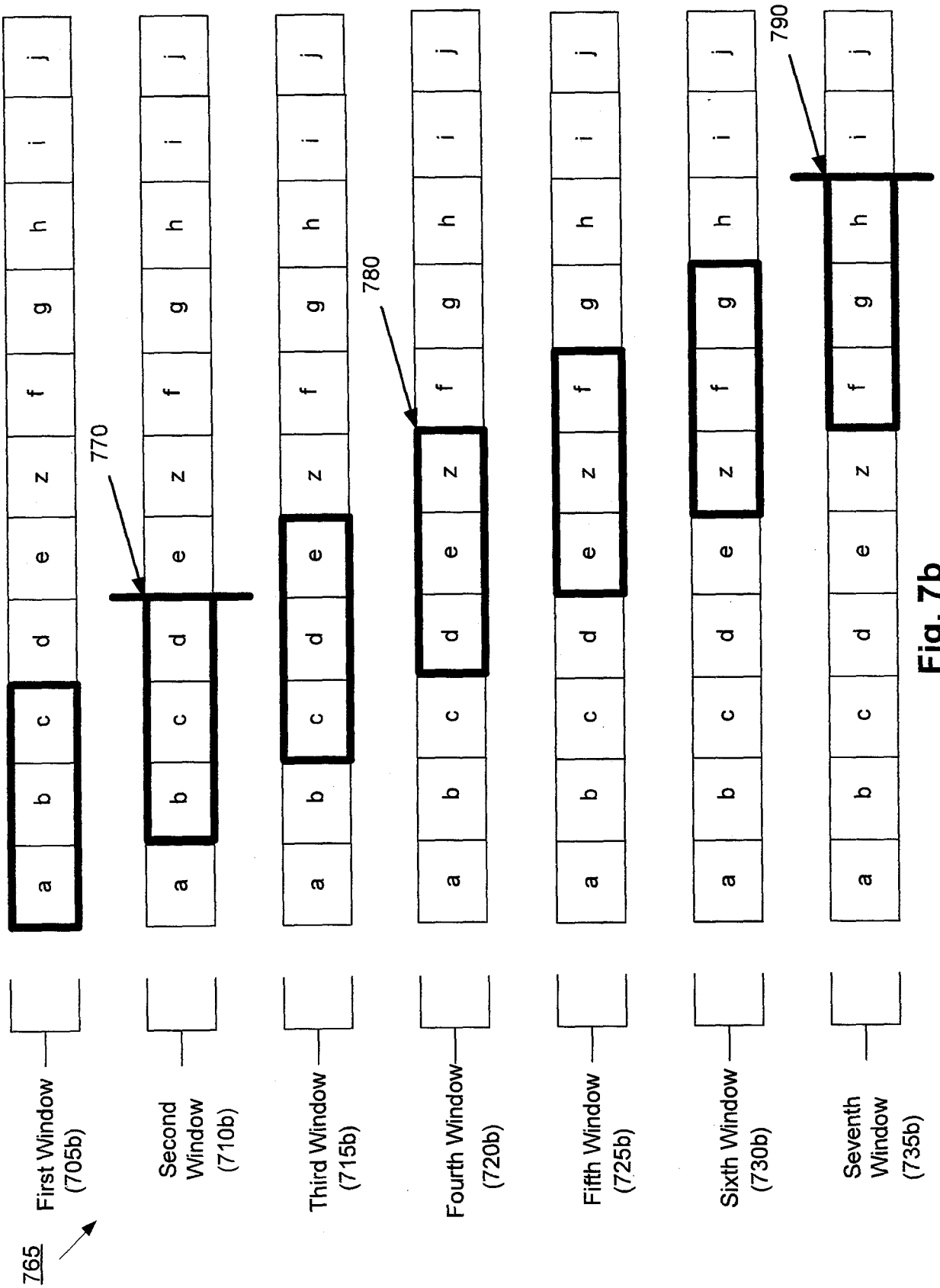

765
First Window (705b)
Second Window (710b)
Third Window (715b)
Fourth Window (720b)
Fifth Window (725b)
Sixth Window (730b)
Seventh Window (735b)
770
780
790
a b c d e z f g h i j

Fig. 7b

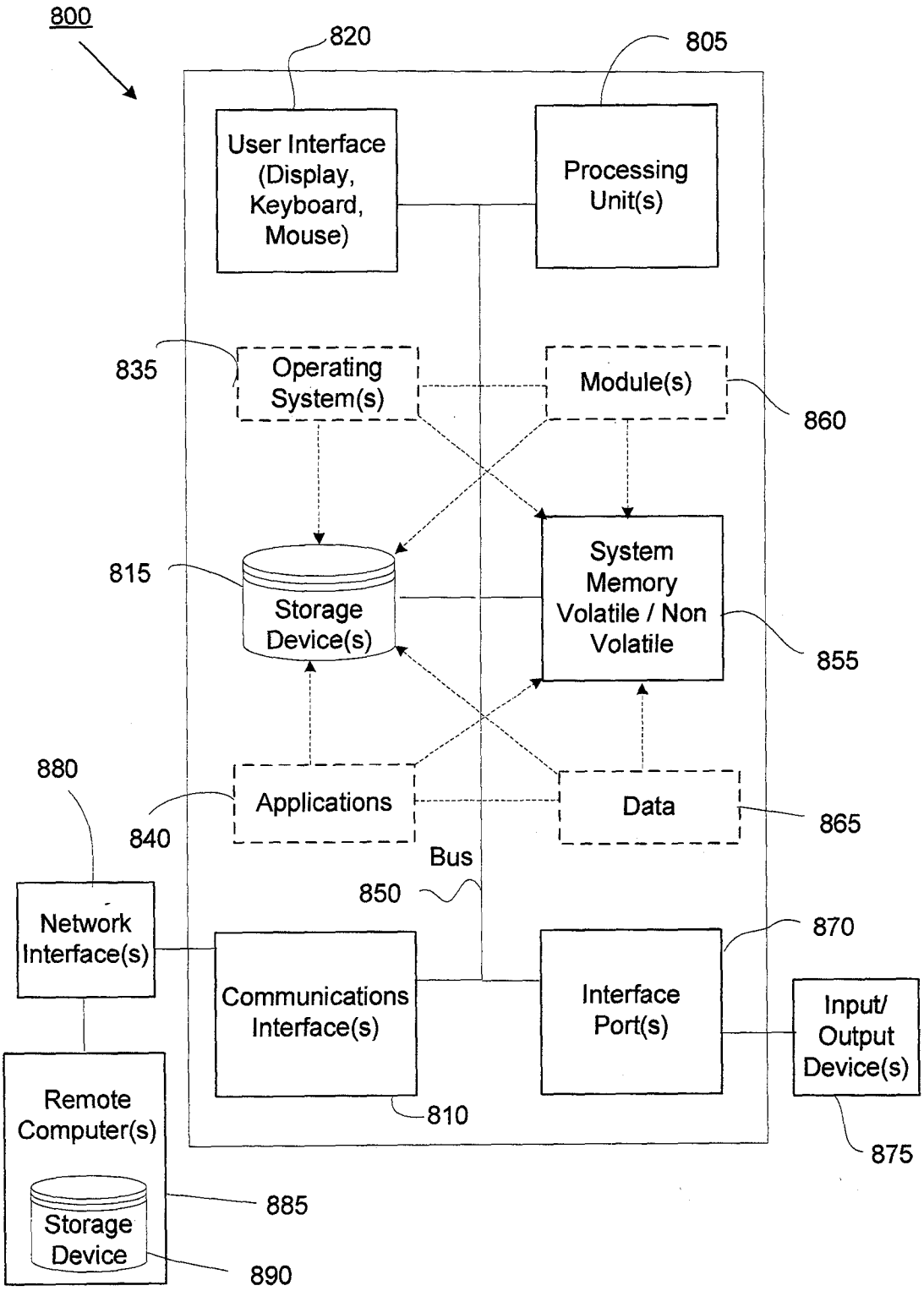

800
820
805
User Interface (Display, Keyboard, Mouse)
Processing Unit(s)
835
Operating System(s)
Module(s)
860
815
Storage Device(s)
System Memory Volatile / Non Volatile
855
880
840
Applications
Data
865
Bus
850
Network Interface(s)
Communications Interface(s)
870
Interface Port(s)
Input/ Output Device(s)
875
810
Remote Computer(s)
885
Storage Device
890

Fig. 8

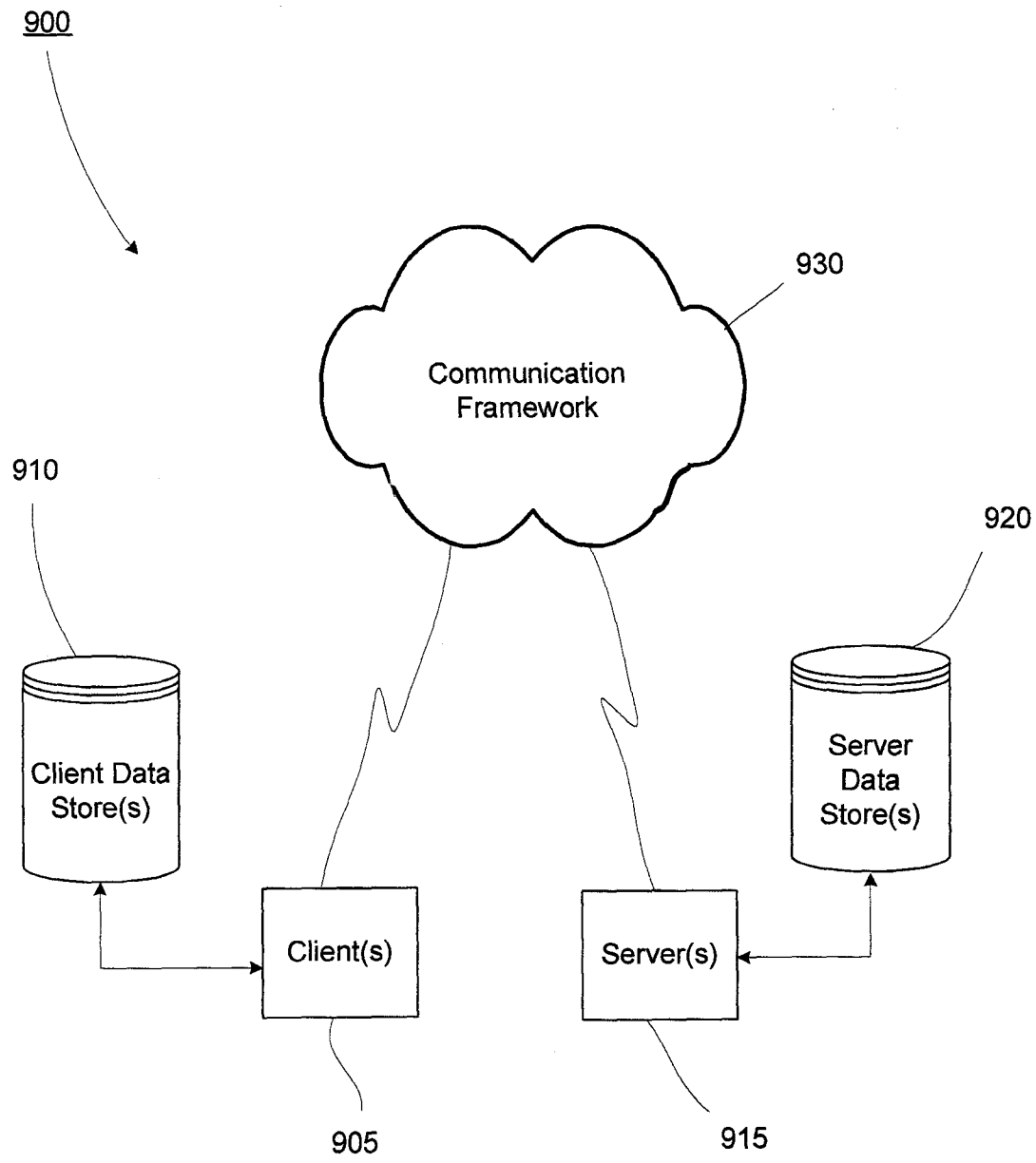

900
930
Communication Framework
910
920
Client Data Store(s)
Server Data Store(s)
Client(s)
Server(s)
905
915

Fig. 9

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US2007/085357

A. CLASSIFICATION OF SUBJECT MATTER

G06F 7/24(2006.01)i

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

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)

IPC 8 : G06F. H03M

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

Korean Utility models and applications for Utility models since 1975

Japanese Utility models and applications for Utility models since 1975

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

eKIPASS(KIPO internal) "hash", "data", "redundancy", "identification", "elimination"

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	US 20020169934 A1(OLIVER KRAPP et al) 14 November 2002 See abstract; figures 1-11.	1-29
A	US 5990810 A(ROSS NEIL WILLIAMS) 23 November 1999 See abstract; figures 4-26.	1-29
A	US 6828925 B2(STEVEN MCCANNE et al.) 7 December 2004 See abstract; figures 3-14.	1-29
A	US 20040225655 A1(GREGORY HAGAN MOULTON) 11 November 2004 See abstract; figures 3-14.	1-29

☐ Further documents are listed in the continuation of Box C.

☒ See patent family annex.

* Special categories of cited documents:

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

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

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

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

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

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

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

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

"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
29 FEBRUARY 2008 (29.02.2008)	**29 FEBRUARY 2008 (29.02.2008)**

Name and mailing address of the ISA/KR

Korean Intellectual Property Office
Government Complex-Daejeon, 139 Seonsa-ro, Seo-gu, Daejeon 302-701, Republic of Korea

Facsimile No. 82-42-472-7140

Authorized officer

HAN, Seon Kyoung

Telephone No. 82-42-481-8523

INTERNATIONAL SEARCH REPORT

Information on patent family members

International application No.
PCT/US2007/085357

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
US 20020169934 A1	14.11.2002	EP 01244221 A1	25.09.2002
		US 6889297 B2	03.05.2005
US 5990810 A	23.11.1999	AU 123295 A0	16.03.1995
		AU 239295 A0	11.05.1995
		AU 4659396 A1	04.09.1996
		WO 9625801 A1	22.08.1996
US 6828925 B2	07.12.2004	US 20040174276 A1	09.09.2004
		US 2005162288 A1	28.07.2005
		US 2006061495 A1	23.03.2006
		US 2007018858 A1	25.01.2007
		US 6961009 B2	01.11.2005
		US 7116249 B2	03.10.2006
US 20040225655 A1	11.11.2004	US 7272602 B1	18.09.2007