



- (51) **International Patent Classification:**
G06F 17/30 (2006.01)
- (21) **International Application Number:**
PCT/SG2016/050156
- (22) **International Filing Date:**
30 March 2016 (30.03.2016)
- (25) **Filing Language:** English
- (26) **Publication Language:** English
- (30) **Priority Data:**
10201502504Q 30 March 2015 (30.03.2015) SG
- (71) **Applicant:** NATIONAL UNIVERSITY OF SINGAPORE [SG/SG]; 21 Lower Kent Ridge Road, Singapore 119077 (SG).
- (72) **Inventors:** DINH, Tien Tuan Anh; c/o National University of Singapore, School of Computing, Department of Computer Science, 21 Lower Kent Ridge Road, Singapore 119077 (SG). SAXENA, Prateek; c/o National University of Singapore, School of Computing, Department of Com-

puter Science, 21 Lower Kent Ridge Road, Singapore 119077 (SG). **CHANG, Ee-Chien**; c/o National University of Singapore, School of Computing, Department of Computer Science, 21 Lower Kent Ridge Road, Singapore 119077 (SG). **OOI, Beng Chin**; c/o National University of Singapore, School of Computing, Department of Computer Science, 21 Lower Kent Ridge Road, Singapore 119077 (SG). **ZHANG, Chunwang**; c/o National University of Singapore, School of Computing, Department of Computer Science, 21 Lower Kent Ridge Road, Singapore 119077 (SG).

(74) **Agent:** SPRUSON & FERGUSON (ASIA) PTE LTD; P.O. Box 1531, Robinson Road Post Office, Singapore 903031 (SG).

(81) **Designated States** (*unless otherwise indicated, for every kind of national protection available*): AE, AG, AL, AM, AO, AT, AU, AZ, BA, BB, BG, BH, BN, BR, BW, BY, BZ, CA, CH, CL, CN, CO, CR, CU, CZ, DE, DK, DM, DO, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, GT, HN, HR, HU, ID, IL, IN, IR, IS, JP, KE, KG, KN, KP, KR,

[Continued on next page]

(54) **Title:** EXTRACTING INFORMATION FROM A DATA SET IN A DISTRIBUTED COMPUTING ENVIRONMENT

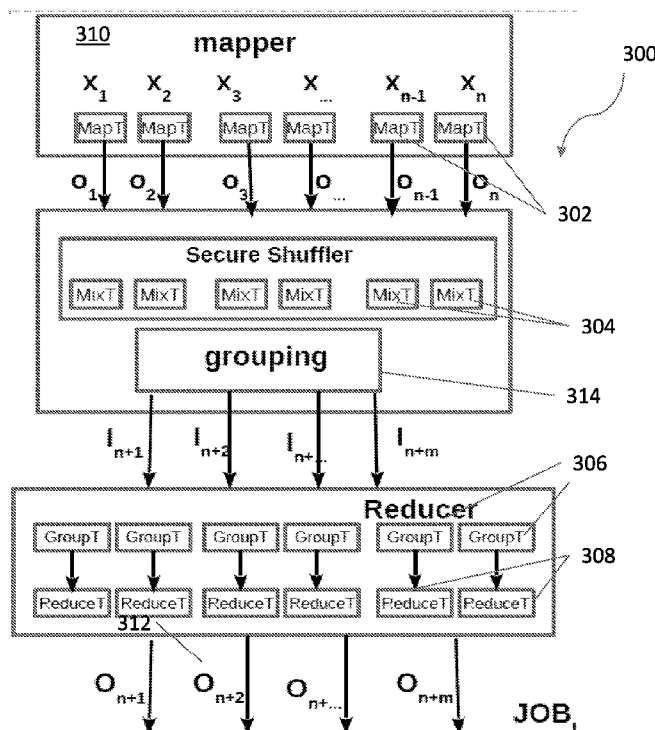


Figure 3

(57) **Abstract:** The threat of data theft in public and private clouds from insiders (e.g. curious administrators) is a serious concern. In relation to this issue, there is disclosed a method of extracting desired information from a data set in a distributed computing environment. The method comprises receiving the data set at a plurality of tuple-forming nodes, each node receiving a unique data subset of the data set, and applying, at respective tuple-forming nodes, an operation to each data subset to produce data tuples, each data tuple including a key-component by which the data tuples can be grouped. Data tuples are then sent to a plurality of first permuting nodes, wherein a substantially equal number of data tuples are received at each first permuting node, and the tuples are then randomly permuted. After random permuting, the data tuples are sent to a plurality of second permuting nodes, wherein a substantially equal number of data tuples are received at each second permuting node, and once again randomly permutes. The data tuples are subsequently sent to a plurality of grouper computing nodes, wherein each grouper computing node receives all data tuples the key-component of which is in a set of key-components, the set being unique to each respective grouper computing node, whereat the data tuples are grouped according to the key-component of each data tuple, and the data tuples are then analysed to extract the desired information.



KZ, LA, LC, LK, LR, LS, LU, LY, MA, MD, ME, MG, MK, MN, MW, MX, MY, MZ, NA, NG, NI, NO, NZ, OM, PA, PE, PG, PH, PL, PT, QA, RO, RS, RU, RW, SA, SC, SD, SE, SG, SK, SL, SM, ST, SV, SY, TH, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, ZA, ZM, ZW.

DK, EE, ES, FI, FR, GB, GR, HR, HU, IE, IS, IT, LT, LU, LV, MC, MK, MT, NL, NO, PL, PT, RO, RS, SE, SI, SK, SM, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, KM, ML, MR, NE, SN, TD, TG).

Declarations under Rule 4.17:

— of inventorship (Rule 4.17(iv))

Published:

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

(84) Designated States (unless otherwise indicated, for every kind of regional protection available): ARIPO (BW, GH, GM, KE, LR, LS, MW, MZ, NA, RW, SD, SL, ST, SZ, TZ, UG, ZM, ZW), Eurasian (AM, AZ, BY, KG, KZ, RU, TJ, TM), European (AL, AT, BE, BG, CH, CY, CZ, DE,

EXTRACTING INFORMATION FROM A DATA SET IN A DISTRIBUTED COMPUTING ENVIRONMENT

The present application claims priority from Singapore application No. 10201502504Q, filed on 30 March 2015, the entire contents of which is incorporated herein by reference.

TECHNICAL FIELD

[1] The present disclosure relates to the extraction of data from a data set in a distributed computing environment. The present disclosure also relates to a distributed computing environment used for that data extraction.

BACKGROUND

[2] New big-data analysis platforms can enable distributed computation on encrypted data utilizing trusted computing primitives available in commodity server hardware. One methodology for distributed computation on encrypted data is the MapReduce technique. One of the main issues for current MapReduce methodologies is that only individual units of distributed computation (e.g. map and reduce units) are protected. This leaves several important channels of information leakage exposed to adversaries.

[3] Some alternative design choices, such as an oblivious random access memory (ORAM) construction and analogous constructions, can achieve stronger privacy of the data during execution. However, constructions such as the ORAM construction are expensive and time-intensive processes to execute.

[4] Trusted computing primitives available today in commodity servers have been shown useful in enabling a privacy-preserving computation on an untrusted device. However, ensuring privacy-preserving execution of a distributed computation on an untrusted cloud is a challenging and open problem.

[5] It is desired therefore that there be provided a method for distributed computation of data on, *inter alia*, untrusted cloud-based systems that maintains privacy without the significant addition of overhead and consequent processing time.

SUMMARY

[6] The present invention provides a method of extracting desired information from a data set in a distributed computing environment, the method comprising:

receiving the data set at a plurality of tuple-forming nodes, each node receiving a unique data subset of the data set;

applying, at respective tuple-forming nodes, an operation to each data subset to produce data tuples, each data tuple including a key-component by which the data tuples can be grouped;

sending the data tuples to a plurality of first permuting nodes, wherein a substantially equal number of data tuples are received at each first permuting node;

randomly permuting the data tuples;

sending the data tuples to a plurality of second permuting nodes, wherein a substantially equal number of data tuples are received at each second permuting node;

randomly permuting the data tuples;

sending the data tuples to a plurality of grouper computing nodes, wherein each grouper computing node receives all data tuples the key-component of which is in a set of key-components, the set being unique to each respective grouper computing node;

grouping the data tuples according to the key-component of each data tuple; and

analysing the data tuples to extract the desired information.

[7] The tuples may be encrypted before each sending step. The tuples may be decrypted after each sending step.

[8] Encryption of data before each sending step may employ a randomized encryption scheme.

[9] The sets of key-components may be non-overlapping. The sets of key-components may comprise disjoint ranges of key components.

[10] The method may further include the step of padding the data tuples to a common tuple size, before sending the data tuples to the first permuting nodes.

[11] The permutation step performed at the second permuting nodes may be a linear-time permutation algorithm.

[12] A processing time for processes performed at the permuting nodes is padded to a common processing time, before data tuples are sent from the respective permuting node.

[13] The second permuting nodes may encrypt at least part of each data tuple using deterministic symmetric encryption. The second permuting nodes may encrypt only the key-component of each data tuple using deterministic symmetric encryption. The method may further comprise encrypting a non-key-component of each data tuple using an encryption key selected independently of the encryption key used to perform the deterministic symmetric encryption. The first set of key-components may comprise deterministically symmetrically encrypted key-components, such that all data tuples with a particular deterministically symmetrically encrypted key-component are sent to the same grouper node. The step of grouping data tuples according to the key-component of each data tuple may comprise sorting data tuples occurs on the respective key-component when deterministically encrypted.

[14] Between permuting of data tuples at the first permuting nodes and permuting of data tuples at the second permuting nodes, the data tuples may be sequentially permuted at one or more pluralities of further permuting nodes.

[15] The step of analysing the data may comprise performing a reduce operation.

[16] Processing of data tuples at each permuting node and grouper node may be synchronised.

[17] Before the grouping step, the data tuples may be sorted according to the key-component of each data tuple.

[18] The present invention also provides a distributed computing environment for extracting desired information from a data set, comprising:

a plurality of tuple-forming nodes for forming data tuples from the data set;

multiple pluralities of permuting nodes for permuting data tuples; and

a plurality of grouper nodes for grouping data tuples, each grouper node comprising a processor for analysing the data tuples to extract the desired information, wherein the tuple forming nodes are in communication with a first of the pluralities of permuting nodes, and the grouper nodes are in communication with a second of the pluralities of permuting nodes.

[19] The first and second pluralities of permuting nodes may be in communication with each other.

[20] The distributed computing environment may comprise at least one further plurality of permuting nodes such that the data tuples are sequentially permuted by each of the at least one further plurality of permuting nodes before being communicated to the second plurality of permuting nodes.

[21] The present invention further provides a computer-readable medium comprising computer program steps that, when executed by a computer, cause the computer to perform methods as taught herein.

[22] The computer-readable medium may comprise computer program steps for managing input and output interfacing between the pluralities of nodes, and computer program steps for managing job scheduling at the pluralities of nodes.

[23] The steps of forming data tuples, permuting data and analysing data may be performed in a trusted computing base (TCB), and one or both of the computer program steps for managing input and output interfacing between the pluralities of nodes, and the computer program steps for managing job scheduling at the pluralities of nodes, are performed outside the TCB.

[24] Some embodiments of the invention may provide a method for computing on encrypted data that avoids using the expensive ORAM construction, and achieves only a logarithmic factor of latency in the MapReduce system. Some embodiments may enhance an existing Hadoop implementation.

[25] Some embodiments of the invention may offer a factor of 1.3 times to 44.6 times lower overhead than existing solutions with equivalent privacy, and a total of 17% to 130% of overhead over the baseline solution which protects only individual units of the computation.

[26] While the discussion made with reference to the drawings is targeted at MapReduce, it can be generalized to other distributed dataflow frameworks such as Spark and Dryad.

BRIEF DESCRIPTION OF THE DRAWINGS

[27] Some non-limiting embodiments of the invention will now be described with reference to the drawings in which:

FIG. 1 illustrates a method of extracting desired information from a data set in a distributed computing environment;

FIG. 2 is an example of a MapReduce computation model;

FIG. 3 provides an illustrative example of a data flow employing trusted components taught herein;

FIG. 4 provides an illustrative example of a data flow in accordance with the method of the present invention using a 2-round mix network;

FIG. 5 shows a normalized break-down time for applications of the method of the present invention, the running time consisting of the time taken by the tuple-forming and grouper computing nodes, plus the time taken by the permuting nodes, along with residual running time taken by other parts of the software stack (e.g. provisioning);

FIG. 6 shows the cost of executing the operation performed by the tuple-forming nodes (i.e. mapT as discussed below) and the cost for executing the secure shuffling process afforded by the permuting nodes – input sizes (number of ciphertexts per input) for generating FIG. 6 vary from 2 to 10^6 ;

FIG. 7 is an expanded block diagram of an exemplary embodiment of a server architecture of a computer system for implementing a method of extracting desired information from a data set in a distributed computing environment; and

FIG. 8 illustrates an exemplary configuration of a server system shown in Figure 7.

DETAILED DESCRIPTION

[28] Embodiments of the present invention will be described, by way of example only, with reference to the drawings. Like reference numerals and characters in the drawings refer to like elements or equivalents.

[29] Some portions of the description which follows are explicitly or implicitly presented in terms of algorithms and functional or symbolic representations of operations on data within a computer memory. These algorithmic descriptions and functional or symbolic representations are the means used by those skilled in the data processing arts to convey most effectively the substance of their work to others skilled in the art. An algorithm is here, and generally, conceived to be a self-consistent sequence of steps leading to a desired result. The steps are those requiring physical manipulations of physical quantities, such as electrical, magnetic or optical signals capable of being stored, transferred, combined, compared, and otherwise manipulated.

[30] Unless specifically stated otherwise, and as apparent from the following, it will be appreciated that throughout the present specification, discussions utilizing terms such as “receiving”, “retrieving”, “filtering”, “providing”, “displaying”, “analysing”, “enabling”, “disabling” or the like, refer to the action and processes of a computer system, or similar electronic device, that manipulates and transforms data represented as physical quantities within the computer system into other data similarly represented as physical quantities within the computer system or other information storage, transmission or display devices.

[31] The present specification also discloses apparatus for performing the operations of the methods. Such apparatus may be specially constructed for the required purposes, or may comprise a computer or other device selectively activated or reconfigured by a computer program stored in the computer. The algorithms and displays presented herein are not inherently related to any particular computer or other apparatus. Various machines may be used with programs in accordance with the teachings herein. Alternatively, the construction of more specialized apparatus to perform the required method steps may be appropriate. The structure of a computer will appear from the description below.

[32] In addition, the present specification also implicitly discloses a computer program, in that it would be apparent to the person skilled in the art that the individual steps of the method described herein may be put into effect by computer code. The computer program is not intended to be limited to any particular programming language and implementation thereof. It will be appreciated that a variety of programming languages and coding thereof may be used to implement the teachings of the disclosure contained herein. Moreover, the computer program is not intended to be limited to any particular control flow. There are many other variants of the computer program, which can use different control flows without departing from the spirit or scope of the invention.

[33] Furthermore, one or more of the steps of the computer program may be performed in parallel rather than sequentially. Such a computer program may be stored on any computer readable medium. The computer readable medium may include storage devices such as magnetic or optical disks, memory chips, or other storage devices suitable for interfacing with a computer. The computer readable medium may also include a hard-wired medium such as exemplified in the Internet system, or wireless medium such as exemplified in the GSM mobile telephone system. The computer program when loaded and executed on such a computer effectively results in an apparatus that implements the steps of the preferred method.

[34] FIG. 1 illustrates a method 100 of extracting desired information from a data set in a distributed computing environment. The method 100 broadly comprises:

Step 102: receiving the data set at a plurality of tuple-forming nodes, each node receiving a unique data subset of the data set;

Step 104: applying, at respective tuple-forming nodes, an operation to each data subset to produce data tuples, each data tuple including a key-component by which the data tuples can be grouped;

Step 106: sending the data tuples to a plurality of first permuting nodes, wherein a substantially equal number of data tuples are received at each first permuting node;

Step 108: randomly permuting the data tuples;

Step 110: sending the data tuples to a plurality of second permuting nodes, wherein a substantially equal number of data tuples are received at each second permuting node;

Step 112: randomly permuting the data tuples;

Step 114: sending the data tuples to a plurality of grouper computing nodes, wherein each grouper computing node receives all data tuples the key-component of which is in a set of key-components, the set being unique to each respective grouper computing node;

Step 116: grouping the data tuples according to the key-component of each data tuple; and

Step 118: analysing the data tuples to extract the desired information.

[35] Step 102 involves receiving the data set at a plurality of tuple-forming nodes. The data set may be a file, image, plurality of files or images, or any other data.

[36] Each node receives a unique data subset of the data set. For example, where the input data set is a text file, a first tuple-forming node may receive a first portion of the text file – for example, the first 1,000 words, the first 1kb of the file or any other portion – a second tuple-forming node may receive a second portion of the text file – for example, the second 1,000 words, the second 1kb of the file etc – and so on. Similarly, the input data set may be a number of text files (or other files) with a unique group of one or more of the text files being sent to each tuple-forming node. In this sense, the data set is segmented into unique data subsets.

[37] Step 104 involves applying, at respective tuple-forming nodes, an operation to each data subset. The operation produces the data tuples. Each data tuple includes a key-component by which the data tuples can be grouped. For example, the method may be applied in a word count job where the intention is to count the number of occurrences of each word in the text file constituting the input data set. For each word in the data subset at

a respective tuple-forming node, the node may form an output tuple of $\langle w, 1 \rangle$ for each word w in the data subset. The output of the method should therefore be an encrypted list of tuples $\langle w, w_c \rangle$ where w_c is the number of occurrences of word w in the input data set.

[38] Step 106 involves sending the data tuples to a plurality of first permuting nodes. The tuples may be encrypted before sending. The tuple-forming nodes may use common encryption and decryption keys, or may each use a different set of encryption and decryption keys. In fact, encryption may be used before each sending operation.

[39] Encryption may employ any form of encryption. In the present context, encryption involves employing a randomised encryption scheme.

[40] Each sending operation may involve a substantially equal number of data tuples being received at each first permuting node. This may be achieved by the tuple-forming nodes collectively sending a substantially equal number of tuples to each first permuting node, or may involve each individual tuple-forming node sending a substantially equal number of tuples to each first permuting node. Distributing a substantially equal number of data tuples to each first permuting node ensures that an observer cannot determine the manner in which the tuples are distributed for shuffling. It also makes it more difficult to determine the tuple-forming node from which a tuple was sent once that tuple leaves the respective first permuting node.

[41] Where the tuples have been encrypted before sending, the first permuting nodes may decrypt the tuples upon receipt. Of course, decryption may occur after each sending step where the tuples have been encrypted. Alternatively, the first permuting nodes may shuffle the encrypted tuples in their encrypted form.

[42] Step 106 may further include padding the data tuples before sending to the first permuting nodes. The tuples may be padded to a common tuple size. The advantages of padding to a common tuple size are discussed below.

[43] Step 108: involves randomly permuting the data tuples. This is a simple permutation operation where the data is shuffled or permuted.

[44] Step 110 involves sending the data tuples to a plurality of second permuting nodes. The tuples may be encrypted before sending. As with the tuple-forming nodes, the first permuting nodes may use common encryption and decryption keys, or may each use a different set of encryption and decryption keys.

[45] Each sending operation may involve a substantially equal number of data tuples being received at each second permuting node. This may be achieved by the first permuting nodes collectively sending a substantially equal number of tuples to each second permuting node, or may involve each individual first permuting node sending a substantially equal number of tuples to each second permuting node. Again, a substantially equal

distribution of data tuples ensures that an observer cannot track a tuple from the tuple-forming nodes, through the first permuting nodes to the second permuting nodes.

[46] Where the tuples have been encrypted before sending, the second permuting nodes may decrypt the tuples upon receipt. Alternatively, the second permuting nodes may shuffle the encrypted tuples in their encrypted form.

[47] The first and second permuting nodes are used to permute or shuffle the data tuples to make it difficult to track a tuple leaving a particular tuple-forming node, through the shuffling or permuting process, to the grouper computing nodes. An observer therefore cannot determine which output (i.e. the desired information) relates to which data subset inputted to the tuple-forming nodes..

[48] Step 112: involves randomly permuting the data tuples. This may be performed by a similar operation to that used at step 108 or may involve a different random permutation operation.

[49] Step 114 involves sending the data tuples to a plurality of grouper computing nodes. The sending operation is performed so that each grouper computing node receives all data tuples the key-component of which is in a set of key-components. For example, all data tuples containing word w (i.e. the tuple key) are sent to the same grouper computing node so that the relevant grouper computing node can aggregate all the values from each tuple.

[50] A particular grouper computing node may receive all tuples with a key-component from a group of key-components. In other words, a particular grouper computing node may receive all tuples with tuples keys being words w and w_i .

[51] Since all tuples having a particular key-component are sent to a particular grouper computing node, each set of key-components are unique to each respective grouper computing node. Moreover, the sets of key-components in this embodiment are necessarily non-overlapping since no key component of a data tuples sent to a particular grouper computing node will be the same as the key component of a data tuple sent to any other grouper computing node.

[52] As an added security measure, each set of key-components comprise disjoint ranges of key-components. In so doing, an observer will not be able to determine which key-components, from all possible key-components, are analysed by a particular grouper computing node. Moreover, if an observer learns one of the key-components, that knowledge cannot be used to infer the values of the other key-components for that grouper computing node.

[53] Step 116 involves grouping the data tuples according to the key-component of each data tuple. As mentioned with reference to step 114, since all data tuples having the same key component are sent to the same grouper computing node, the output from the

respective grouper computer node will be a single tuple for each key component in cases such as the word count example – for example, for n tuples for word w $\langle w, 1 \rangle$ the grouper computing node will output $\langle w, n \rangle$.

[54] Step 118 involves analysing the data tuples to extract the desired information. In the word count example the aggregated output $\langle w, n \rangle$, the number n of each word w in the data set, is the desired information.

Exemplary Implementations

[55] The embodiments discussed below are directed to enabling privacy preserving distributed computation on an untrusted cloud. A sensitive distributed computation task in the present embodiments employs many units of computation – each unit of computation is a set comprising of tuple-forming nodes, permutation nodes and grouper computing nodes. These units of computation are scheduled to run on a multi-node cluster (i.e. a cloud). The input and output data transmitted between sub-units of computation (e.g. to/from a tuple-forming node, a permuting node and a grouper computing node) are sent over channels controlled by a provisioning system – presently the provisioning system is a cloud provisioning system. It is assumed that the provisioning system is compromised – in other words, unsecure.

[56] In these embodiments, it is assumed that each computation node comprises a central processing unit (CPU). Each CPU is capable of supporting trusted computing primitives such as Trusted Platform Modules (TPMs) or Intel Software Guarded Extensions (SGX), in the cluster.

[57] The described embodiments are directed to privacy design in a MapReduce framework but may similarly be applied to Spark, Dryad and epiC frameworks. In the MapReduce framework, computation consists of two types of sub-units of computation, namely a *map* operation and a *reduce* operation. Each of the map and reduce operations takes key-value tuples or key-component tuples as inputs. A provisioning platform schedules the sub-units of computation for execution in a cluster. The provisioning platform also provides a data channel between the sub-units of computation. The provisioning platform may be, for example, Hadoop.

[58] The present embodiments endeavour to achieve a strong level of security in the distributed execution of a MapReduce task (or job). In some cases the adversary learns nothing beyond the execution time and the number of input and output tuples of each computation unit – for MapReduce, the computation task is split into map and reduce operations. If each unit of computation is viewed as one atomic operation of a larger distributed program, the execution can be thought of as running a set of operations on data

values passed via a data channel (or a global “RAM”) under adversarial control (i.e. control of an adversary). Privacy achievable using embodiments described herein should therefore be analogous to the strong level of privacy offered by the oblivious RAM protocol in the monolithic processor case.

[59] It is similarly assumed the MapReduce provisioning platform is compromised – for example, the provisioning platform may be running malware on all nodes in the cluster. Each unit of computation (e.g. mapper or reducer) in the system on which the present embodiments are run – in other words the “baseline system” – is operated in a hardware-isolated process. Inputs and outputs of each computation unit are encrypted. The adversary therefore observes only encrypted data. Additional security is required over and above encryption since the adversary can:

- observe the pattern of data reads and writes between units;
- learn the synchronization between sub-unit of computation due to the scheduling structure of the provisioning platform; and
- have the ability to duplicate computation, or tamper with the routing of encrypted data to observe variations in the execution of the program.

[60] To achieve a high level of security all sub-units of computation could be run on a single computation node – including, in the present instance, the entire MapReduce platform. All sub-units of computation may be protected by use of existing trusted computing primitives. However, such a solution would entail little trust given the large Trusted Computing Base (TCB). It would also be unwieldy to implement. For instance, a standard implementation of the Hadoop stack is over 190,000 lines of code. The scope of exploit from vulnerabilities in such a TCB is large. Such a solution is also impractical and necessarily fails to take advantage of the power of cloud computing.

[61] There is therefore a need to balance privacy and performance. Addressing leakage channels discussed above using generic methods easily yields a solution with poor practical efficiency. For instance, hiding data read and write patterns between specific computation operations (e.g. map and reduce operations) could be achieved by an oblivious RAM (ORAM) solution. Such a solution introduces a computational slowdown proportional to polylog in the size of the intermediate data exchange. As the data set increases in size, the performance degradation can result in processing times being hundreds of times slower than processing times for non-ORAM solutions.

[62] Two observations enabling the present embodiments to maintain privacy on a distributed computing environment are that:

- on a single node, most of the MapReduce codebase can stay outside of the TCB (i.e. code performing input/output (I/O) management and scheduling related tasks). The

present embodiments have therefore been designed using four new components that integrate readily to, for example, the existing MapReduce infrastructure. The four new components are the only pieces of trusted logic in the TCB. The four new components are run in a protected environment on each computation node; and

- MapReduce computation, and computation in distributed dataflow frameworks in general, has a specific structure of data exchange and execution between sub-units of computation. The map writes the data completely before it is consumed by the reduce. Exploiting this structure enables design of a secure shuffler component or permutation node – referred to above as a first permutation node, second permutation node and so on. The secure shuffler achieves the desired security but is much less expensive than a generic ORAM solution. In particular, the secure shuffler adds only a $O(\log N)$ term to the latency.

[63] In embodiments discussed herein, based on Hadoop, 7 applications were imported from popular big-data benchmarks and evaluated on a cluster. The results confirm three findings. First, porting MapReduce jobs to the present framework (i.e. method) requires small development effort: changing less than 45 lines of code. Second, the present framework solution offers a factor of 1.3 times to 44.6 times (median 11.2 times) reduction in overhead compared to the existing solutions with equivalent privacy, and a total of 17% to 130% of overhead over the baseline solution which protects against none of the attacks in respect of which the present methods provide invulnerabilities. In some embodiments, overhead is moderately high but is balanced against high compatibility and usability with high-sensitivity big data analysis tasks (e.g. in medical, social or financial data analytics). Third, the design is scalable and adds a TCB of less than 0.16% of the original Hadoop codebase in the MapReduce example in which a HaDoop provisioning platform is used.

[64] FIG. 2 shows a dataflow 200 of a MapReduce operation, from the map to the reduce operations, via a shuffling step. In practice, a MapReduce operation consists of sequential phases of map and reduce operations. Once the map step is finished using map computation sub-units 202, the intermediate tuples are grouped by their key-components. This process of grouping is known as shuffling as is performed using shuffler units 204. All tuples belonging to one group are processed by a reduce instance 206. The reduce instances 206 expect to receive tuples sorted by their key-component. Outputs of the reduce step can be used as inputs for the map step in the next phase, creating a chained MapReduce phases.

[65] In the actual implementation, the provisioning of all map units on one cluster node is locally handled by a mapper process and, similarly, by a reducer process for reduce units.

[66] In the present embodiments, the adversary may be a malicious insider in the cloud,

aiming to subvert the confidentiality of the client's computation running on the MapReduce platform. The adversary may have complete access to the network and storage back-end of the infrastructure and can tamper with any persistent storage or network traffic. For each computation node (202, 206) in the cluster, the adversary may be able to corrupt the entire software stack. This may be achieved, for example, by installing malware.

[67] The adversary may also perpetrate passive attacks as well as active attacks. A passive or honest-but-curious attacker passively observes the computation session, behaving honestly in relaying data between computation units (202, 206), but aims to infer sensitive information from the observed data. This is a pragmatic model. The adversaries may observe data backed up periodically on disk for archival. The adversaries may also have access to performance monitoring interfaces. An active or malicious attacker (e.g. an installed malware) can deviate arbitrarily from the expected behavior and tamper with any data under its control. In this regard, there are at least two direct baseline attacks that an adversary can mount on a MapReduce computation session. First, the adversary can observe data passing between computation units. If the data is left unencrypted, this leads to a direct breach in confidentiality. Second, the adversary can subvert the computation of each map or reduce instance by tampering with its execution. To address these basic threats, a baseline system upon which the present embodiments operate is one in which each computation unit is hardware-isolated and executed privately.

[68] In the present embodiments, the baseline system guarantees the program can only be invoked on its entire input dataset, or else it aborts in its first computation or map phase. Data blocks or tuples entering and exiting a computation unit (202, 206) are encrypted with authenticated encryption. All side-channels from the computation units (202, 206) are masked. Intermediate data is similarly decrypted only in a hardware-attested computation unit, which has limited memory to securely process up to T input tuples. Note that in this baseline system, the MapReduce provisioning platform is responsible for invoking various trusted units of computation in hardware-isolated processes. The provisioning platform passes encrypted data between those trusted units of computation. However, this baseline leaks significant information.

[69] Ideally, the distributed execution of the MapReduce program should leak nothing to the adversary, except the total size of the input, total size of the output and the running time. The aforementioned baseline system fails to achieve this ideal level of privacy. It leaks two types of information:

- (a) the input and output size, and processing time of individual computation units;
- and
- (b) dataflow among the computation units.

[70] The leakage of dataflow among the computation units is significant in many applications since it reveals relationships among portions of the input. For instance, in computing Pagerank scores for an encrypted graph, each flow from a computation unit to another corresponds to an edge in the input graph. Leaking the dataflow essentially reveals the whole graph edge-structure of the graph.

[71] In line with present teachings, techniques for hiding or reducing the leakage of the input and output size and processing time include padding the input and/or output size. Similarly, introducing timing delays also masks the processing time – in other words, padding the processing time. For example, the processing time at the permuting nodes may be padded to a common processing time before data tuples are sent from the permuting nodes. Such measures may require algorithmic redesign of the application, use of specialized programming languages or hardware. Such measures may lead to large overheads for applications where the worst case running time is significantly larger than the average case. The present embodiments focus on eliminating leakage of dataflow information. The present embodiments also provide a neat formulation clearly capturing information that might be revealed.

[72] In addition, the present embodiments specify an admissible leakage or leakage tolerance Ψ . The admissible leakage Ψ captures the input and output size and running time of each trusted computation unit (i.e. set of tuple-forming, permutation and grouper computing nodes) invoked in the system. In this sense, the execution is defined as a protocol between trusted components and the adversary, and the intent is to achieve privacy modulo- Ψ .

[73] Consider an honest execution of a program on input data set $I = \langle x_1, x_2, \dots, x_n \rangle$. For a given MapReduce phase, let there be n map computation units. Let the map computation units be labeled such that the unit with label i takes x_i as its input. Since the tuples generated by the map computation units are to be shuffled, and divided into groups according to the key-components, let K be the set of unique key-components and let $\pi : [n+1; n+m] \rightarrow K$ be a randomly chosen permutation, where $m = |K|$. Next, m reduce computation units are to be invoked. These reduce computation units are labeled starting from $n+1$, such that the computation unit i takes tuples with key-component $\pi(i)$ as its input. Similarly, let I_i, O_i, T_i be the respective input size (measured by number of tuples), output size, and processing time of the computation unit i respectively, and call $\Psi_i = \langle I_i, O_i, T_i \rangle$ the IO-profile of computation unit i . The profile Ψ of the entire execution on input I is the sequence of Ψ_i for all computation units $i \in [1, \dots, n+m]$ in the execution protocol. If an adversary A can initiate the above protocol and observe Ψ , then the adversary has access to Ψ .

[74] Now considering execution of the program on the same input $I = \langle x_1, x_2, \dots, x_n \rangle$ under a MapReduce provisioning protocol by an adversary A . A semi-honest adversary A can obtain information on the value of the input (i.e. the data set or tuples sent to each permuting node or grouper computing node), output (i.e. the encrypted tuples or the desired data, encrypted or otherwise) and processing time of every trusted instance, including information on trusted instances other than the sub-unit of computation. If the adversary is malicious, it can further tamper with the inputs and invocations of the instances.

[75] The adversary may therefore be able to control 6 parameters:

- (C1) the start time of each computation instance;
- (C2) the end time of each instance;
- (C3) the encrypted tuples passed to its inputs;
- (C4) the number of computation instances;
- (C5) order of computation units executed; and
- (C6) the total number of map-reduce phases executed.

[76] Since the adversary A can obtain “more” information and tamper with the execution, then the present methods take into account the consideration of whether adversary A can gain more knowledge than an adversary having access only to Ψ . Using the standard notions of indistinguishability and adversaries, the following is defined:

Privacy modulo- Ψ – a provisioning protocol for a program is modulo- Ψ private if, for any adversary A executing, for example, the MapReduce protocol or other relevant protocol, there is an adversary $\tilde{A}$ with access only to Ψ , such that the output of A and $\tilde{A}$ are indistinguishable.

[77] This definition states that the output of the adversaries can be directly seen as deduction made on the information available. The fact that all adversaries have output indistinguishable from the one which knows Ψ suggests that no additional information can be gained by any adversary beyond that implied by knowledge of Ψ . This definition follows the scenario proposed by Canetti (R. Canetti – Universally composable security: A new paradigm for cryptographic protocols – *IEEE symposium on Foundations of Computer Science*, 2001), which facilitates universal composition. Hence, if a protocol is private module- Ψ for one map-reduce phase or cycle of the method, then an entire sequence of phases executed is private module- Ψ . The proposed method involves use of a sequence of sub-units of computation each executing tuple-forming, permuting and grouping phases, the sub-units being the tuple-forming nodes, permuting nodes and grouper computing nodes respectively. In the application of embodiments of the present method to a MapReduce protocol, a sequence of map (tuple-forming), shuffle (permuting), and reduce

(grouping) phases are used – where each phase starts only after the previous phase has completed. The phases may thus be carried out sequentially.

[78] This ensures universal composition can be applied.

[79] If a developer restructures the original computation to make the I/O-profile the same for all inputs, then leakage due to Ψ is zero. In this case, privacy modulo- Ψ implies the execution becomes completely oblivious. Therefore, the developer can consider using orthogonal techniques to mask timing latencies. Thus trace paths and I/O patterns can be hidden to achieve ideal privacy, if the performance considerations permit so.

[80] In order to achieve ideal privacy in some example embodiments, it is assumed the underlying hardware of the baseline system sufficiently protects each computation unit from malware and snooping attacks. The range of threats against which protection is given varies based on the non-negligible advantage in a distinguishing game underlying trusted computing hardware. For instance, traditional TPMs protect against software-only attacks but not against physical access to RAM via attacks such as cold-boot. More recent trusted computing primitives, such as Intel SGX, encrypt physical memory and therefore offer stronger protection against adversaries with direct physical access. Therefore, the methods of the example embodiments do not focus on the specifics of how to protect each computation unit, as it is likely to change with changes in the hardware platform. Instead the present methods can be implemented in any virtualization-assisted isolation that protects user-level processes on a malicious guest operating system (OS).

[81] In order to achieve ideal privacy it is also assumed information leakage via side-channels (e.g. cache latencies, power) from a computation unit is minimal.

[82] Finally, to enable arbitrary computation on encrypted data, decryption keys need to be made available to each hardware-isolated computation unit. This provisioning of a client's keys to the cloud requires a set of trusted administrator interfaces and privileged software. In order to achieve ideal privacy, it is also assumed such trusted key provisioning exists.

[83] Since the sub-units of computation are maintained in the TCB, the result of the above analysis of privacy modulo- Ψ is that the provisioning protocol must be private modulo- Ψ . To this end, FIG. 3 shows a data flow 300 of embodiments of the present invention. Filled or shaded components are trusted. Input (i.e. the data set), intermediate and output tuples are encrypted – in other words, the tuples sent from tuple-forming nodes 302 to the first permuting nodes 304 and so on through the data flow 300. The original map and reduce operations are replaced with the tuple-forming nodes (hereinafter referred to as “mapT”) and grouper computing nodes (hereinafter referred to as “reduce”). New components of the

embodiment of FIG. 2 are the mixer nodes which use mixT, and another trusted component called groupT of the TCB to an existing MapReduce platform.

[84] The computation proceeds in phases, each consisting of a map step (performed using tuple-forming nodes), a shuffle step (performed using first permuting nodes, second permuting nodes and so on where additional sets of permuting nodes are used per step 120, for example, between the first set of permuting nodes and the second set of permuting nodes), and a reduce step (performed using grouper computing nodes). In the dataflow 300 of Fig. 3 the four new trusted components are depicted. These four new TCB components are mapT 302 (tuple-forming node), mixT 304 (permuting node), groupT 306 and reduceT 308 (grouper computing node). Two of these, mapT 302 and groupT 306 correspond to the execution of map and reduce sub-units of computation. They ensure that output tuples from the mapper 310 and reducer 312 respectively are encrypted and each tuple is of the same size. The reduceT 308 and mixT 304 implement the critical role of secure shuffling or permuting.

[85] Notably, the present new TCB components enable integrity checks to defeat active attacks to be distributed. This results in minimal global synchronization. The shuffler in the platform is responsible for grouping tuples, and invoking reduce units on disjoint ranges of tuple-keys. In other words, the method may comprise grouping permuted tuples after the respective random permutation steps at process 314, before sending the data tuples on to the next plurality of permuting nodes or to the grouper computing nodes. On each cluster node, reduceT 308 checks the grouped order and the expected range of tuples received using the trusted groupT 306 component. The outputs of the reduceT 308 units are then again fed back into the next processing phase.

[86] The present methods can enable a major part of the software stack, which deals with job scheduling and I/O operations, to be left outside of the TCB. The present design makes no change to the grouping and scheduling algorithms. They can in fact be outside the Trusted Computing Base (TCB) as shown in FIG. 2. Therefore, the design is conceptually simple and requires no intrusive changes to be implemented over existing MapReduce implementations and the implementations of other protocols.

[87] For any given execution, it is useful to ensure that each computation step in a phase is private modulo- Ψ . For example, if the map step, the shuffle step, and the reduce step of the operation can be individually made private modulo- Ψ , by the property of serial composability, the entire phase and a sequence of phases can be shown to be private module- Ψ .

[88] One significant challenge to achieving such privacy is performing secure shuffling. Consider the naive approach in which the entire shuffler is moved into the platform TCB of

each cluster node. Consider then the grouping step of the shuffler, often implemented as a distributed sort or hash-based grouping algorithmic process. The grouping process can only process a limited number of tuples locally at each mapper, so access to intermediate tuples must go to the network during the grouping process. Here, network data access patterns from the shuffler leak information. For example, if the shuffler were implemented using a standard merge sort implementation, the merge step leaks the relative position of the pointers in sorted sub-arrays as it fetches parts of each sub-array from the network incrementally. This can be removed using ORAM implementations, but these add significantly to overhead.

[89] In MapReduce and other dataflow frameworks, the sequence of data access patterns is fixed: it consists of cycles of tuple writes followed by reads. For example, the reduce units start reading and processing their inputs only after the map units have finished. Some embodiments of the present methods employ *inter alia* two significant steps: firstly, intermediate encrypted tuples are rewritten with re-randomized tuple keys. This is done such that there is no ability to link the re-randomized tuples and the original encrypted map output tuples. This step can be realized by a secure mix network. The privacy of the computation in the present case reduces directly to the problem of secure mixing. The total latency added by a solution according to the present embodiments in an additive term of $O(\log M)$ in the worst case. In the MapReduce context, since the shuffle step is based on sorting which already admits $O(N \log M)$ overhead, the present method asymptotically retains the complexity of the original framework.

[90] The present embodiments can employ multiple phases of the method as a whole. The present embodiments can also employ multiple shuffling or permutation steps. As such, privacy can be maintained using a cascaded mix network (or cascaded-mix) to securely shuffle tuples. For example, the procedure may consist of a cascading of k intermediate steps – with reference to dataflow 400 of FIG. 4, two intermediate (or permuting) steps 402, 404 are performed. The present methods may employ k identical steps (mixing steps) each employing a number of trusted computation units called mixT units (permutation sub units), the execution of which can be distributed over multiple nodes called mixers or permuting nodes. Each mixT in step 402 takes a fixed amount of T tuples that it can process in memory. That mixT of step 402 passes exactly the same number of encrypted tuples to all mixT units in the subsequent step 404. Therefore, in each step of the cascade, the mixer utilizes N/T mixT units for mixing N tuples. At $k = \log_T^N$, the network ensures the strongest possible unlinkability between input and output tuples. That is, the output distribution is statistically indistinguishable from a random distribution.

[91] Each permuting node or mixT sub-unit decrypts the tuples it receives from the previous step. It then randomly permutes the tuples. Random permutation may employ a linear-time algorithm. The mixT then re-encrypts the permuted tuples with one or more new randomly chosen symmetric encryption keys. The keys are known only to mixT sub-units. The keys can be derived using a secure key-derivation function from a common secret. The processing time of mixT sub units may be padded. The padding may be so that the processing time becomes constant. The padding incurs low overhead since the re-encryption time has low variance over different inputs. The re-encrypted tuples may then be grouped before being sent on to the groupT sub-units 306. Grouping of the re-encrypted tuples may be grouping into groups comprising sets of discontinuous, unique key-components.

[92] FIG. 4 may be deemed to represent a distributed computing environment 400 for extracting desired information from a data set. The environment may be a cloud-based environment. The distributed computing environment 400 comprises a plurality of tuple-forming nodes (mapT 406) for forming data tuples from the data set. The environment 400 further comprises multiple pluralities of permuting nodes 408, 410 for permuting data tuple. The environment 400 further comprises a plurality of grouper nodes 412 for grouping data tuples, each grouper node comprising a processor for analysing the data tuples to extract the desired information.

[93] The tuple forming nodes 406 are in communication with a first plurality 408 of the pluralities of permuting nodes (408, 410), and the grouper nodes 410 are in communication with a second plurality 410 of the pluralities of permuting nodes (408, 410). It will be understood that further pluralities of permuting nodes may be provided between the pluralities of permuting nodes 408, 410. In either case, the pluralities of permuting nodes 408, 410 are in communication with each other either firstly – in the case where no further pluralities of permuting nodes are given – or indirectly through one or more further pluralities of permuting nodes.

[94] Let W represents the number of input and output tuples of cascaded-mix with k steps. Intuitively, when k is sufficiently large, an semi-honest adversary who has observed the execution does not gain more knowledge than W . In this regard, cascaded-mixing is private module- W under semi-honest adversarial attack, given that the underlying encryption scheme is semantically secure.

[95] After the mixing step, secure grouping is performed. The shuffler can group the randomized tuple keys. As mentioned above, the grouping scheme or algorithm may be outside the TCB. The output of the cascaded-mix is fed into the grouping scheme or algorithm. This scheme or algorithm combines all tuples with the same tuple-key. The

algorithm or scheme forwards the combined tuples to reducers. To avoid the circumstance in which the outputs of the cascaded-mix are probabilistically encrypted, the grouping step may be performed in a trusted component. In embodiments of the present methods, a further step may be added in the cascade. That further step may be added at the end of the cascade – for example, for a cascade comprising k steps, embodiments of the present method may include a $(k+1)^{\text{th}}$ step. The $(k+1)^{\text{th}}$ step accommodates the requirement for subsequent grouping. The further step may comprise using deterministic symmetric encryption F_s to encrypt the key-component of the output tuples. In fact, deterministic symmetric encryption of at least a part of each tuple may be performed at one or more sets of the permuting nodes – for example, the second permuting nodes may encrypt only the key-component of each data tuple (i.e. the ‘word’ being counted in a word counting operation). Using deterministic encryption ensures that encrypted key-component is constant. Accordingly, the tuples can be sent to the correct grouper computing node without decryption. For added security, the non-key-component of each tuple (e.g. the value accompanying the word) may be encrypted using an encryption key selected independently of the encryption key used to perform the deterministic symmetric encryption. Encryption may be performed with a secret key s . The $\langle a; b \rangle$ may be encrypted to a ciphertext of the form $\langle F_s(a), E(a, b) \rangle$, where $E(-)$ is a probabilistic encryption scheme. This ensures that the two shuffled tuples with the same tuple-keys have the same ciphertext for the key-component of the tuple. This enables subsequent grouping to be performed by the groupT sub units without decrypting the tuples.

[96] The secret key s may be randomized in each invocation of the embodiments of the present methods employing the further step. This can be used to randomize the ciphertexts across two map-reduce phases or jobs.

[97] What the adversary gains by observing the last or further step of mixing is the tuples groups which are permuted using $F_s(-)$. Thus, if $F_s(-)$ is a pseudorandom function family, the adversary can only learn about the size of each group, which is already implied by Ψ . Embodiments of the present methods are therefore private modulo- Ψ (under semi-honest adversarial attack), assuming that the underlying private-key encryption is semantically secure, and $F_s(-)$ is a pseudorandom function family.

[98] A malicious adversary may deviate from the behavior set out above by mounting active attacks using the 6 parameters under its control. To this end, some embodiments of the present method may provide a directed acyclic graph (DAG). The vertices of the DAG may comprise trusted computation units. The edges of the DAG may denote the flow of encrypted data blocks. According to FIGs. 3 and 4 the present embodiments provide four kinds of trusted computation sub-units or vertices in the DAG: mapT (tuple forming node),

mixT (permuting node), groupT and reduceT (grouper computing node). At a high-level, an integrity checking mechanism works by ensuring that nodes at the j^{th} level (by topological sorted order) in the DAG check the consistency of the execution at level $j-1$. The method may, in this instance, comprise determining whether an adversary deviates or tampers with the execution or outputs from level $j-1$. The method may they include the step of aborting the execution. The method may therefore similarly comprise processing all tuples output from level $i-1$ at level i . This step ensures that a computation in step i starts only after outputs of previous steps are passed to it. This implicitly synchronizes the start of the actual computation units at step i . Under this constraint, it can be shown that the start-end time of each computation node only allows the adversary to delay an entire step, or distinguish the outputs of units within one step, which is already implied by Ψ .

[99] The present methods may also be private modulo- Ψ under attack from a malicious adversary. Such privacy may assume the underlying authenticated-encryption is semantically secure (confidentiality). Such privacy may also assume security under a chosen message attack (integrity). Such privacy may further assume $F_s(-)$ is a pseudorandom function family. Given a malicious adversary A that executes the present methods, an adversary $\tilde{A}$ may be simulated only has access to Ψ in the following manner. To simulate A , the adversary $\tilde{A}$ needs to fill in information not present in Ψ . For the output of a trusted unit of computation, the simulation simply fills in random tuples, where the number of tuples is derived from Ψ . The timing information can likewise be filled-in. Whenever adversary A deviates from the protocol and feeds a different input to a trusted instance, the simulation will expect the instance, halt and fill in the information accordingly. Note that the input to A and the input constructed for the simulation $\tilde{A}$ could have the same DAG of program execution, although the encrypted tuples are different. Suppose there is a distinguisher that distinguishes A and $\tilde{A}$, then there are two cases to consider, namely that the two DAG's are either the same or different. If there are non-negligible chances that they are the same, then distinguisher can be constructed to contradict the security of the encryption, or $F_s(-)$. If there are non-negligible chances that they are different, a valid authentication tag can be forged. Hence, the outputs of A and $\tilde{A}$ are indistinguishable.

[100] Note: nearly all the integrity checks can be distributed across the cluster, with checking of invariants done locally at each trusted unit of computation. Therefore, integrity checking mechanism described above can largely bundle the integrity metadata with the original data. No global synchronization is necessary, except for the case of the groupT units (i.e. a grouping sub-unit of each permuting node. The mixT may be the other sub-unit of a permuting node) consuming data processed by an untrusted grouping step. In this regard, the method may involve synchronizing processing of tuples at the first permuting nodes and/or second permuting nodes and/or any other sets of permuting nodes, and may

also involve synchronizing between permuting nodes and grouper computing nodes. That synchronization may comprise global synchronization. This ensures that the ordering of the grouped tuples received by the designated reduceT is preserved. The method may thus involve synchronizing the groupT units (i.e. grouping sub-units of each permuting node). This ensures each reducer (grouper computing node) processes a distinct range of tuple-keys, and that all the tuple-keys are processed by at least one of the reduce units.

[101] To accommodate the DAG instance – in other words where the DAG corresponds to a program execution – the method may comprise assign a unique instance identifier to each node. The method may instead comprise assigning a unique instance identifier to each sub-unit of computation. In a practical application, the provisioning system shall assign three identifiers. Let the vertex i at the level j be designated identifier (i,j) , and the total number of units at level j be $|V_j|$.

[102] The method may further comprise sending on or both of each designated identifier and a total number of units $|V_j|$, to each node. In other words, when a computation instance is spawned (i.e. the present method commences), its designed instance identifier (i,j) and the total number of units $|V_j|$ are passed by the provisioning system to each node. They may be passed as auxiliary input parameters. Each vertex with identifier (i,j) is an operation of type mapT (i.e. tuple-forming node), groupT, mixT (a groupT and a mixT instance together forming a permuting node) or reduce (grouper computing node). This may be denoted by the function $OpType(i,j)$. For integrity checking, each vertex may emit a tagged-block as output which can be checked by trusted components in the next stage.

Specifically, the tagged block may be a 6-tuple $B = \langle O, LvlCnt, SrcID, DstID, DstLvl, DstType \rangle$, where:

- O is the encrypted output tuple-set;
- LvlCnt is the number of units at source level;
- SrcID is the instance id of the source vertex;
- DstID is instance id of destination vertex or NULL
- DstLvl is the level of the destination vertex; and
- DstType is the destination operation type.

[103] The method may comprise each vertex with identifier (i,j) fetching the tagged-blocks from all vertices at the previous level, denoted by the multiset B . The method may then involve performing one or more of the following consistency checks on B (each consistency check may be performed in isolation of the other checks and, similarly, any combination of such checks may be performed):

1. The LvlCnt for all $b \in B$ are the same (say $\ell(B)$).
2. The SrcID for all $b \in B$ are distinct.

3. For set $S = \{ \text{SrcID}(b) \mid b \in 2B, jS_j = \text{`}(B) \}$.
4. For all $b \in 2B$, $\text{DstLvl}(b) = j$.
5. For all $b \in 2B$, $\text{DstID}(b) = (i,j)$ or NULL.
6. For all $b \in 2B$, $\text{DstType}(b) = \text{OpType}(i,j)$.

[104] In line with the consistency checks, the method may comprise checking tagged-blocks from all units in the previous level are read. The method may similarly involve checking tagged-blocks from all units in the previous level are distinct. These checks result from consistency checks 1 to 3. Thus, it is ensured the adversary has not dropped or duplicated any output tuple. The method may involve checking the computation nodes are ordered sequentially. This is achieved by consistency check 4. In other words, the adversary cannot misroute data bypassing certain levels. The method may involve checking that execution progresses in the expected order — for instance, the map or tuple-forming step is followed by a mix or permutation step, subsequently followed by a group step, and so on. This can be achieved using consistency check 6. The method may comprise the step of checking if the source vertex wishes to fix the recipient identifier of a tagged-block by identifier. The method may comprise setting it verifiably enforcing it by setting it to a non-NULL value. This is achieved by consistency check 5.

[105] The method may further comprise encrypting each tagged-block. The encryption may involve standard authenticated encryption, protecting the integrity of all metadata in the respective tagged block. The method may involve each mixT (permutation node) reading the output metadata of all mapT (tuple-forming node). Thus, each mixT knows the total number of tuples N generated in the entire map step, by summing up the counts of encrypted tuples received. The method may further comprise each mixT (permutation node) independently determining the total number of permuting nodes in the system as $N=T$, where T is the pre-configured number of tuples that each mixT can process securely without invoking disk accesses, for example hundreds of millions of tuples. This method may perform these steps in a decentralized manner. Thus no co-ordination between mixT units is required. A mapT unit (tuple-forming node) invoked with identifier (i,j) simply emits tagged-blocks, with the following structure: $\langle -, |V_j|, (i,j), \text{NULL}, j+1, \text{mixT} \rangle$.

[106] The method may involve each mixT (or permuting node) re-encrypting and permuting a fixed number (T) of tuples. In a k -step cascaded mix network, at any step s ($s < k-1$) the mixT outputs $T=m$ tuples to each one of the m mixT units in the step $s+1$. To ensure this, the method may involve each mixT (or permutation node) adding metadata to its tagged-block output so that it reaches only the specified mixT unit for the next stage. To do so, the DstType field may be used which is set to type $\text{mixTs}+1$ by the mixer at step s . Thus, each mixT node knows the total number of tuples being shuffled N (encoded in OpType), its step

number in the cascaded mix, and the public value T . From this, the method may further comprise each mixT determining the correct number of cascade steps to perform. The method may also comprise aborting the execution if the adversary tries to avoid any step of the mixing.

[107] The method may further involve the last mix step (i,j) (i.e. that which sends the tuples to the grouper computing nodes) writing the original tuple as $\langle Fs(k), (k,v,tctr) \rangle$, where the second part of this tuple is protected with authenticated-encryption. The value tctr is called a tuple-counter, which makes each tuple globally distinct in the running of the method. Specifically, the permutation nodes in the last permutation step encode the value $(i,j;ctr)$ where ctr is a counter unique to the instance (i,j) . The assumption here is that all such output tuples will be grouped by the first component, and each group will be forwarded to reducers with no duplicates. To ensure that the outputs received are correctly ordered and untampered, the mixer sub-unit (mixT) in a respective permutation node may send a special tagged-block to the respective grouper sub-unit (groupT) in the respective node. This tagged-block may comprise the count of tuples corresponding to $Fs(k)$ generated by the immediately preceding mixer sub-unit (mixT unit with id (i,j)). With this information each grouper sub-unit (groupT) can locally check that, for each received group corresponding to $g=Fs(k)$, the count of distinct tuples $(k; _ ; i,j;ctr)$ it receives tallies with that specified in the tagged-block received from mixT node (i,j) , for all blocks in B .

[108] Furthermore the method may involve synchronizing grouper sub-units (groupT units) to identify any overlap between tuple-key ranges. This requires an additional exchange of tokens between groupT units containing the range of group keys and tuple-counters that each unit processes.

[109] There may be a one-to-one mapping between groupT units and reduceT units. This enables the groupT to check the correctness of the tuple group before forwarding to the reduce with which it is mapped. This communication is analogous to that between mixT units.

[110] The present methods may be implemented differently depending on the underlying architectural primitives available. For instance, the present methods could be implemented using Intel SGX, using the mechanisms of VC3 to provide a baseline system. A trusted-hypervisor approach may alternatively be employed to implement the baseline system. The trusted hypervisor approach may minimize the performance overheads from the baseline system. Intel TXT may be used to securely boot a trusted Xen-4.4.3 hypervisor kernel, for example, ensuring its static boot integrity.

[111] All data input and output by map units (i.e. tuple-forming) and reduce units (grouper computing nodes) may be encrypted with AES-GCM utilizing 256-bit keys, for example.

The hypervisor may load, verify and execute TCB components within its address space. The rest of the software stack may run in ring 3 and invoke the units by making hypercalls. Note that the TCB components can be isolated as user-level processes in the future, but this is only meaningful if the processes are protected by stronger solutions such as Intel SGX or other systems.

[112] In performance in a real cluster under real workloads, in which 7 data intensive jobs were imported from the standard benchmark to a system implementing the present methods, an embodiment of the present method was implemented with fewer than 25% of the lines of code modified based on original HaDoo jobs, added fewer than 500 lines of code to the TCB (less than 0.16% of the entire Hadoop software stack), and added 17% to 130% of overhead in running time to the baseline system. The embodiment was benchmarked against another system offering the same level of privacy, in which encrypted tuples are sent back to a trusted client. The embodiment tested ultimately proved up to 44.6 times faster than the benchmark solution.

[113] The selected benchmark solution was the HiBench suite. The 7 benchmark applications are listed in Table 1 and cover a wide range of data-intensive tasks:

- compute intensive (KMeans, Grep, Pagerank);
- shuffle intensive (Wordcount, Index), database queries (Join, Aggregate); and
- iterative (KMeans, Pagerank),

the size of the encrypted input data being between 1 GB and 2.5 GB in these case studies.

As shown in Table 1, the number of application hypercalls consists of both mapT and reduce invocation, and the number of platform hypercalls include groupT and mixT invocations.

<i>Job</i>	<i>LoC changed (vs. Hadoop job)</i>	<i>TCB increase (vs. Hadoop codebase)</i>	<i>Input size (vs. plaintext size)</i>	<i>Shuffled bytes</i>	<i># App hypercalls</i>	<i># Platform hypercalls</i>
Wordcount	36 (15%)	370 (0.14%)	2.1G (1.06x)	4.2G	3277173	35
Index	28 (24%)	370 (0.14%)	2.5G (1.15x)	8G	3277173	59
Grep	13 (13%)	355 (0.13%)	2.1G (1.06x)	75M	3277174	10
Aggregate	16 (18%)	395 (0.15%)	2G (1.19x)	289M	18121377	12
Join	30 (22%)	478 (0.18%)	2G (1.19x)	458M	11910647	14
Pagerank	42 (29%)	429 (0.15%)	2.5G (4x)	2.6G	1750000	21
KMeans	11 (7%)	400 (0.12%)	1G (1.09x)	11K	13000004	8

Table 1 – summary of porting effort and TCB increase for various implementations of the present method, and the application runtime cost factors.

[114] Different applications have different amount of shuffled data, ranging from small sizes (75MB in Grep, 11K in KMeans) to large sizes (4.2GB in Wordcount, 8GB in Index). The implementation of the present solution used the Xen-4.3.3 64-bit hypervisor compiled with a trusted boot option. The rest of software stack of the present method was run on Ubuntu 13.04 64-bit version. The experiments were conducted in a cluster of commodity servers

equipped with 1 quad-core Intel CPU 1.8GHz, 1TB hard drive, 8GB RAM and 1GB Ethernet cards. The setup was varied to have between 1 to 4 compute nodes (running mappers and reducers) and between 1 to 4 mixer nodes for implementing a 2-step cascaded mix network. The results presented below are from running with 4 compute nodes and 4 mixers each reserving a 100MB buffer for mixing, averaged over 10 executions.

[115] Regarding the overheads and cost: Table 2 shows that a total overhead of between 17% and 130% over the baseline system was observed. That overhead is simply involved in encrypting inputs and outputs of map/reduce units (tuple-forming nodes and permuting nodes), and utilizes none of the present privacy-enhancing techniques. It can also be seen that in all applications except for Grep and KMeans, running time is proportional to the size of data transferred during shuffling (shuffled bytes column in Table 1). To understand the cost factors contributing to the overhead the time taken by the secure shuffler was measured, by the mapT and reduceT units, and by the rest of the Hadoop system which comprises the time spent on I/O, scheduling and other book-keeping tasks. This relative cost breakdown 500 is detailed in FIG. 5. From FIG. 5 it is observable that the cost of the secure shuffler 502 is significant in the overall processing time of the job. Therefore, there is incentive to reduce the overheads of shuffling, by avoiding the generic ORAM solution, and that reduction is critical to reducing the overall overheads.

[116] The two main benchmarks which have high overheads of over 100%, namely, Wordcount and Index, incur this cost primarily due to the cost of privacy-preserving shuffling (i.e. permuting and/or permuting and reducing) of a large amount of data. In addition, the total cost of the both the permutation nodes and other trusted components is comparable to that of the existing Hadoop cost. This evidences that the present methods preserve the asymptotic complexity of Hadoop while significantly improving privacy.

[117] Apart from the baseline system, a further point of comparison is previously proposed systems that send encrypted tuples to the user for private computation. Systems such as Monomi and AutoCrypt employ homomorphic encryption for computing on encrypted data on single servers. For operations that cannot be done on the server using partially homomorphic encryption, such Monomi-like systems forward the data to a trusted set of servers (or to a client's private cloud) for decryption.

[118] This is referred to as the download-and-compute approach. To estimate the performance of a Monomi-like system for applications involving the present applications achieving a privacy equivalent to the present methods, it is assumed that the system on which the job is implemented uses Paillier, ElGamal and randomized search schemes for homomorphic computation, but not order-preserving encryption (OPE) or deterministic schemes – these leak more information than the baseline system of the present disclosure.

Operations were then run that fall outside the allowed homomorphic operations, including shuffling, as a separate network request to a trusted client. Network requests were batched into one per MapReduce step and it was assumed the network round trip latency to the client was only 1ms – this is optimistic since the average round trip delay in the same data center is 10 to 100ms.

[119] Table 1 shows that to achieve the improved level of privacy of the present methods all jobs have TCB increases of fewer than 500 lines of code – in other words, 0.16% Hadoop codebase. In fact, when using Intel SGX enclaves it may be possible to eliminate the TCB altogether, and retain only the new trusted components or sub-unit of computation.

[120] As provided by Table 2, the download-and-compute approach is slower than the present methods by a factor of 1.3 to 44:6 (Table 2), with the median benchmark running slower by a factor of 11.2. The overheads are low for case-studies where most of the computation can be handled by homomorphic operations, but most of the benchmarks require conversions between homomorphic schemes (thereby requiring decryption).

<i>Job</i>	<i>Baseline (vs. no encryption)</i>	<i>M²R (% increase vs. baseline)</i>	<i>Download-and-compute (× M²R)</i>
Wordcount	570 (221)	1156 (100%)	1859 (1.6×)
Index	666 (423)	1549 (130%)	2061 (1.3×)
Grep	70 (48)	106 (50%)	1686 (15.9×)
Aggregate	125 (80)	205 (64%)	9146 (44.6×)
Join	422 (211)	510 (20%)	5716 (11.2×)
PageRank	521 (334)	755 (44%)	1209 (1.6×)
KMeans	123 (71)	145 (17%)	6071 (41.9×)

Table 2 – overall running time (s) of the present methods when compared with other systems: (1) the baseline system protecting computation only in single nodes, (2) the download-and-compute system that does not use trusted primitives but instead sends the encrypted tuples back to trusted servers when homomorphic encrypted computation is not possible.

[121] The dominant costs reported in respect of this comparison are largely complementary to the costs incurred by the specifics of the underlying platform. A micro-benchmark comparison test was conducted to evaluate the cost of context-switches and the total time spent processing in the trusted components to explain this aspect. Using some of the present methods the cost of each hypercall (switch to trusted logic) is small (13ms), and the execution of each trusted component is largely proportional to the size of its input data as shown in FIG. 6. The time taken by the trusted computation grows near linearly with the input data-size, showing that the constant overheads of context-switches and specifics of the other platform do not contribute to the reported results significantly. This implies that simple optimizations such as batching multiple trusted code invocations

would not yield any significant improvements, since the overheads are indeed proportional to the total size of data and not the number of invocations. The total number of invocations (via hypercalls) for app-specific trusted logic (mapT, reduceT) is proportional to the total number input tuples. This account for less than half a second of overhead even for millions of input tuples. The number of invocations to the other components (mixT and groupT) is much smaller (8 to 59 invocations) and the each invocation operates on large inputs of a few gigabytes. Therefore, the dominant cost is not that of context-switches, but that of the cost of multi-step shuffling operation itself and the input/output (I/O) overheads.

[122] The present methods achieve stronger privacy than previous platforms that propose to use encrypted computation for big-data analysis. In the examples discussed above, the adversary can observe an admissible amount of information, captured by Ψ , in the computation but hides everything else. It is possible to quantitatively analyze the increased privacy in information-theoretic terms, by assuming the probability distribution of input data. However, Table 3 provides a qualitative description highlighting how much privacy is gained by the techniques introduced in by the present platform over the baseline system. For instance, consider the two case studies that incur most performance overhead (Wordcount, Index). In these examples, merely encrypting the map/reduce tuples leaks information about which file contains which words. This may allow adversaries to learn the specific keywords in each file in the dataset. On the present platform, this leakage is reduced to learning only the total number of unique words in the complete database and the counts of each. In other words, information about individual files is hidden.

[123] Similarly, the present platform hides which records are in which group for database operations. For Pagerank, the baseline system leaks the complete input graph edge structure, giving away which pairs of nodes have an edge. The present platform reduces this leakage to only the number of graph vertices.

[124] The implementations of some of the embodiments of the present methods discussed above use a trusted hypervisor based on Xen for isolated computation.

[125] FIG. 7 is a simplified block diagram of an exemplary network-based system or distributed computing environment 700 for extracting desired information from a data set in a distributed computing environment. System 700 is a client/server system that may be utilized for the processing of data. More specifically, in the example embodiment, system 700 includes a server system 702 that may provide the provisioning platform, and at least one client computer system or terminal 704 each of which may comprise a node or sub-unit of computation. Presently the system 700 includes a plurality of client sub-systems, also referred to as client computer systems 704, though this term is also intended to encompass the circumstance where a client computer system 704 is the computer system of a host

(e.g. provisioning platform host), connected to server system 702. Client systems 704 may be interconnected to the Internet through a variety of interfaces including a network, such as a local area network (LAN) or a wide area network (WAN), dial-in-connections, cable modems and special high-speed ISDN lines. Client systems 704 could be any device capable of interconnecting to the Internet including a personal computer (PC), a web-based phone, personal digital assistant (PDA), or other web-based connectable equipment.

[126] A database server 706 is connected to database 708, which contains information such as the input data set, data subsets, encryption keys, decryption keys and so forth. In one embodiment, centralized database 708 is stored on server system 702 and can be accessed by potential users (e.g. parties desiring to run Wordcount and other MapReduce processes) at one of client systems 704 by logging onto server system 702 through one of client systems 704. In an alternative embodiment, database 708 is stored remotely from server system 702 and may be non-centralized.

[127] The database 708 may also be a non-transitory computer readable medium storing or embodying a computer program for extracting desired information from a data set in a distributed computing environment. The program may include at least one code segment executable by a computer to instruct the computer to perform a method as described herein, for example with reference to FIG. 1. The program may further include computer program steps for managing input and output interfacing between the pluralities of nodes, and computer program steps for managing job scheduling at the pluralities of nodes. The steps of forming data tuples, permuting data and analysing data of the method of FIG. 1 may be performed in a trusted computing base (TCB), and one or both of the computer program steps for managing input and output interfacing between the pluralities of nodes, and the computer program steps for managing job scheduling at the pluralities of nodes, are performed outside the TCB.

[128] FIG. 8 illustrates an exemplary configuration of a computing device 800, similar to server system 700 (shown in FIG. 7). Computing device 800 includes a processor 802 for executing instructions. Instructions may be stored, for example, in a memory area 804 or other computer-readable media. Processor 802 may include one or more processing units (e.g., in a multi-core configuration).

[129] Processor 802 may be operatively coupled to a communication interface 806 such that server computing device 800 is capable of communicating with a remote device such as user computing device 704 (shown in FIG. 7) or another server computing device 800. Thus processor 802 can be distributed across multiple computing systems to achieve cloud-based processing of data sets. For example, communication interface 806 may

receive data sets, provisioning information, encryption and decryption keys, may output desired information and so forth, via the Internet.

[130] Processor 802 may also be operatively coupled to storage device 808. Storage device 808 is any computer-operated hardware suitable for storing and/or retrieving data. In some embodiments, storage device 808 is integrated in server computing device 800. For example, server computing device 808 may include one or more hard disk drives as storage device 808. In other embodiments, storage device 808 is external to server computing device 800 and may be accessed by a plurality of server computing devices 800. For example, storage device 808 may include multiple storage units such as hard disks or solid state disks in a redundant array of inexpensive disks (RAID) configuration. Storage device 808 may include a storage area network (SAN) and/or a network attached storage (NAS) system.

[131] In some embodiments, processor 800 is operatively coupled to storage device 808 via a storage interface 810. Storage interface 810 is any component capable of providing processor 802 with access to storage device 808. Storage interface 810 may include, for example, an Advanced Technology Attachment (ATA) adapter, a Serial ATA (SATA) adapter, a Small Computer System Interface (SCSI) adapter, a RAID controller, a SAN adapter, a network adapter, and/or any component providing processor 802 with access to storage device 808.

[132] In operation, the processor 802, coupled to a memory device (including memory device 804 and storage device 808), is configured to receive a data set and instance of the present method (i.e. the method according to FIG. 1) a host access terminal. The processor 802 is then configured to:

- receive the data set at a plurality of tuple-forming nodes, each node receiving a unique data subset of the data set;

- apply, at respective tuple-forming nodes, an operation to each data subset to produce data tuples, each data tuple including a key-component by which the data tuples can be grouped;

- send the data tuples to a plurality of first permuting nodes, wherein a substantially equal number of data tuples are received at each first permuting node;

- randomly permute the data tuples;

- send the data tuples to a plurality of second permuting nodes, wherein a substantially equal number of data tuples are received at each second permuting node;

- randomly permute the data tuples;

send the data tuples to a plurality of grouper computing nodes, wherein each grouper computing node receives all data tuples the key-component of which is in a set of key-components, the set being unique to each respective grouper computing node;

group the data tuples according to the key-component of each data tuple; and

analyse the data tuples to extract the desired information.

[133] The computer system 800 may be instructed by a computer program embodied on a non-transitory computer readable medium, such as memory device 804 or storage device 808. The program stored on the device 804, 808 would include at least one code segment, and most likely many thousands of code segments, executable by a computer to instruct the computer to perform the requested operations.

[134] Similarly, the program may be stored remotely. To this end, the computer system may constitute a client computer system of a network-based system for performing the above methods.

[135] Many modifications and variations of the present teachings will be apparent to the skilled person in light of the present disclosure. All such modifications and variations are intended to fall within the scope of the present disclosure. Moreover, to the extent possible, features form one of the embodiments described herein may be used in one or more other embodiments to enhance or replace a feature of the one or more other embodiments. All such usage, substitution and replacement is intended to fall within the scope of the present disclosure.

CLAIMS

1. A method of extracting desired information from a data set in a distributed computing environment, the method comprising:
 - receiving the data set at a plurality of tuple-forming nodes, each node receiving a unique data subset of the data set;
 - applying, at respective tuple-forming nodes, an operation to each data subset to produce data tuples, each data tuple including a key-component by which the data tuples can be grouped;
 - sending the data tuples to a plurality of first permuting nodes, wherein a substantially equal number of data tuples are received at each first permuting node;
 - randomly permuting the data tuples;
 - sending the data tuples to a plurality of second permuting nodes, wherein a substantially equal number of data tuples are received at each second permuting node;
 - randomly permuting the data tuples;
 - sending the data tuples to a plurality of grouper computing nodes, wherein each grouper computing node receives all data tuples the key-component of which is in a set of key-components, the set being unique to each respective grouper computing node;
 - grouping the data tuples according to the key-component of each data tuple; and
 - analysing the data tuples to extract the desired information.
2. A method according to claim 1, wherein the tuples are encrypted before each sending step.
3. A method according to claim 1, wherein the tuples are decrypted after each sending step.
4. A method according to claim 2, wherein encryption of data before each sending step employs a randomized encryption scheme.
5. A method according to claim 1, wherein the sets of key-components are non-overlapping.
6. A method according to claim 1, wherein the sets of key-components comprise disjoint ranges of key components.

7. A method according to claim 1, further including the step of padding the data tuples to a common tuple size, before sending the data tuples to the first permuting nodes.
8. A method according to claim 1, wherein the permutation step performed at the second permuting nodes is a linear-time permutation algorithm.
9. A method according to claim 1, wherein a processing time for processes performed at the permuting nodes is padded to a common processing time, before data tuples are sent from the respective permuting node.
10. A method according to claim 2, wherein the second permuting nodes encrypt at least part of each data tuple using deterministic symmetric encryption.
11. A method according to claim 10, wherein the second permuting nodes encrypt only the key-component of each data tuple using deterministic symmetric encryption.
12. A method according to claim 11, further comprising encrypting a non-key-component of each data tuple using an encryption key selected independently of the encryption key used to perform the deterministic symmetric encryption.
13. A method according to claim 12, wherein the first set of key-components comprises deterministically symmetrically encrypted key-components, such that all data tuples with a particular deterministically symmetrically encrypted key-component are sent to the same grouper node.
14. A method according to claim 1, wherein, between permuting of data tuples at the first permuting nodes and permuting of data tuples at the second permuting nodes, the data tuples are sequentially permuted at one or more pluralities of further permuting nodes.
15. A method according to claim 10, wherein the step of grouping data tuples according to the key-component of each data tuple comprises sorting data tuples occurs on the respective key-component when deterministically encrypted.
16. A method according to claim 1, where the step of analysing the data comprises performing a reduce operation.

17. A method according to claim 1, wherein processing of data tuples at each permuting node and grouper node is synchronised.

18. A method according to claim 1, wherein, before the grouping step, the data tuples are sorted according to the key-component of each data tuple.

19. A distributed computing environment for extracting desired information from a data set, comprising:

- a plurality of tuple-forming nodes for forming data tuples from the data set;
- multiple pluralities of permuting nodes for permuting data tuples; and
- a plurality of grouper nodes for grouping data tuples, each grouper node comprising a processor for analysing the data tuples to extract the desired information, wherein the tuple forming nodes are in communication with a first of the pluralities of permuting nodes, and the grouper nodes are in communication with a second of the pluralities of permuting nodes.

20. A distributed computing environment according to claim 19, wherein the first and second pluralities of permuting nodes are in communication with each other.

21. A distributed computing environment according to claim 19, comprising at least one further plurality of permuting nodes such that the data tuples are sequentially permuted by each of the at least one further plurality of permuting nodes before being communicated to the second plurality of permuting nodes.

22. A computer-readable medium comprising computer program steps that, when executed by a computer, cause the computer to perform the method according to claim 1.

23. A computer-readable medium according to claim 22, comprising computer program steps for managing input and output interfacing between the pluralities of nodes, and computer program steps for managing job scheduling at the pluralities of nodes.

24. A computer-readable medium according to claim 23, wherein the steps of forming data tuples, permuting data and analysing data are performed in a trusted computing base (TCB), and one or both of the computer program steps for managing input and output interfacing between the pluralities of nodes, and the computer program steps for managing job scheduling at the pluralities of nodes, are performed outside the TCB.

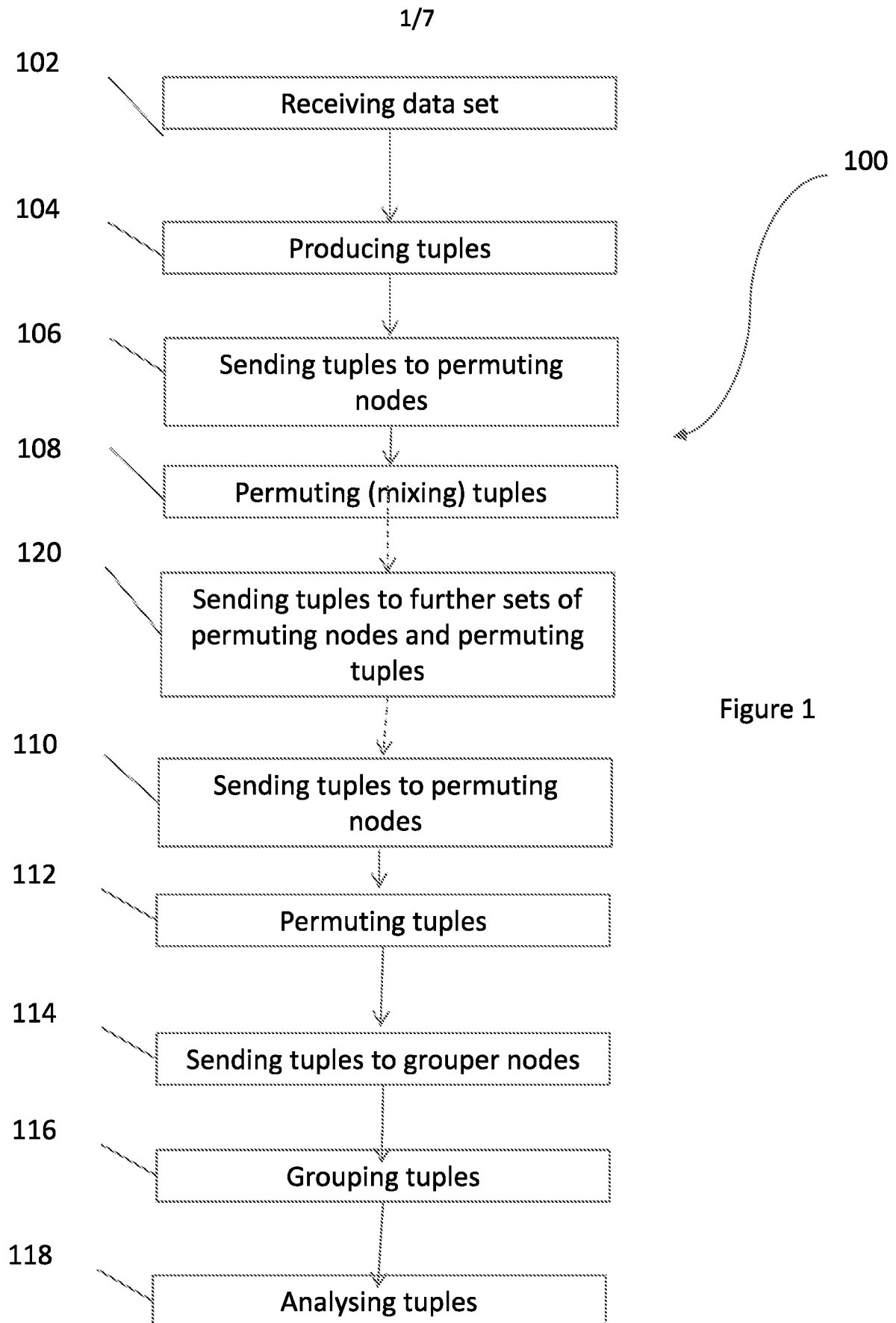
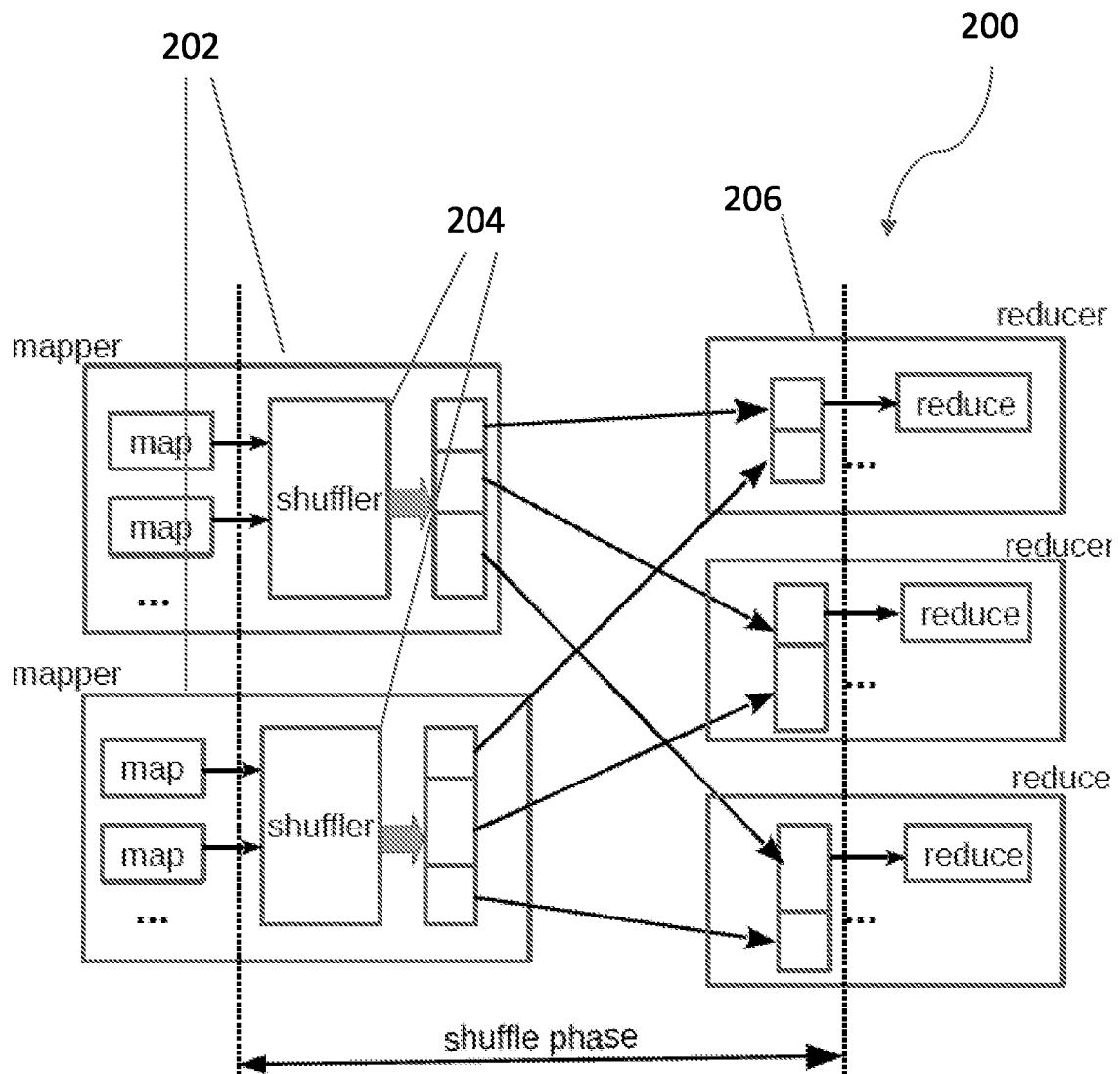


Figure 1

2/7

Figure 2



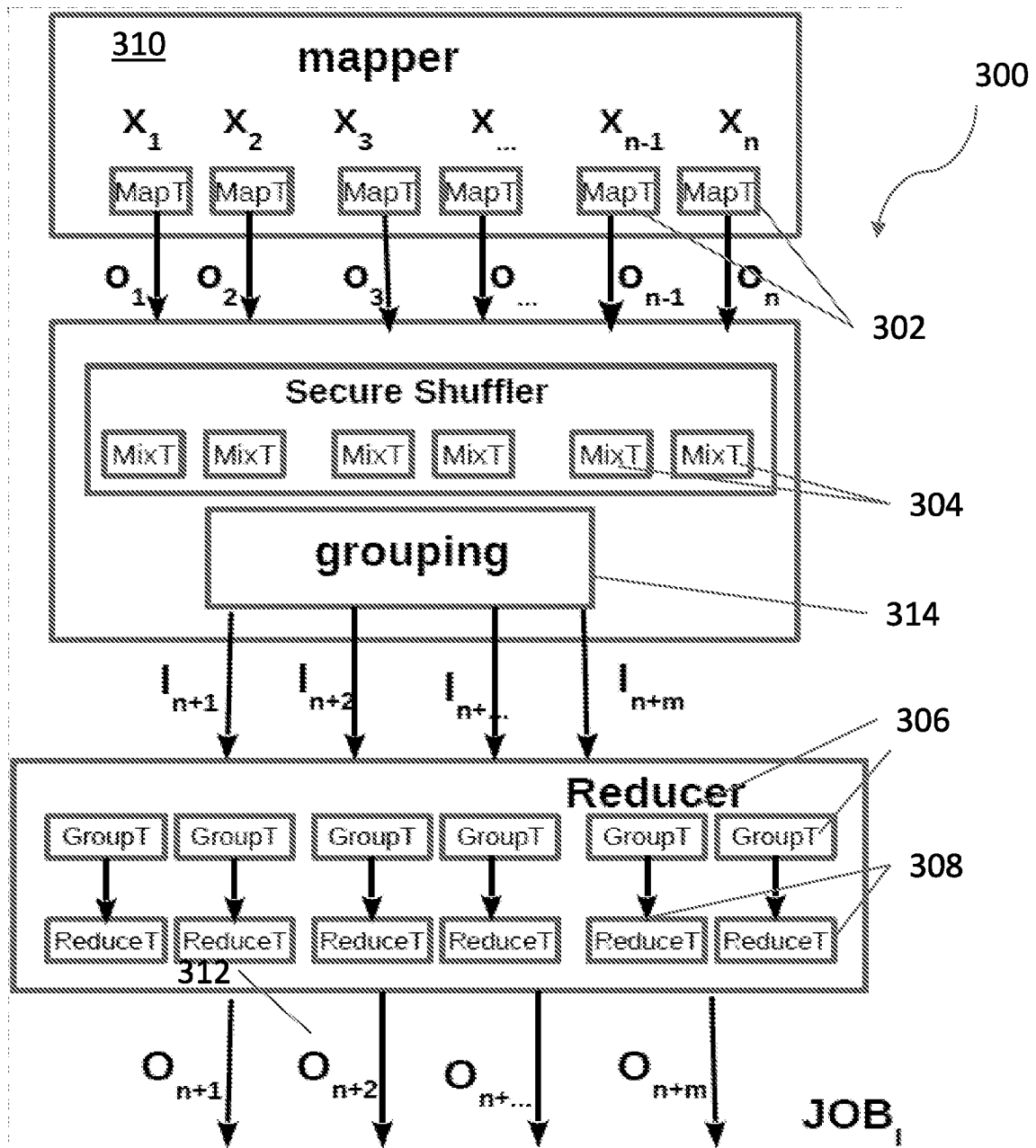


Figure 3

4/7

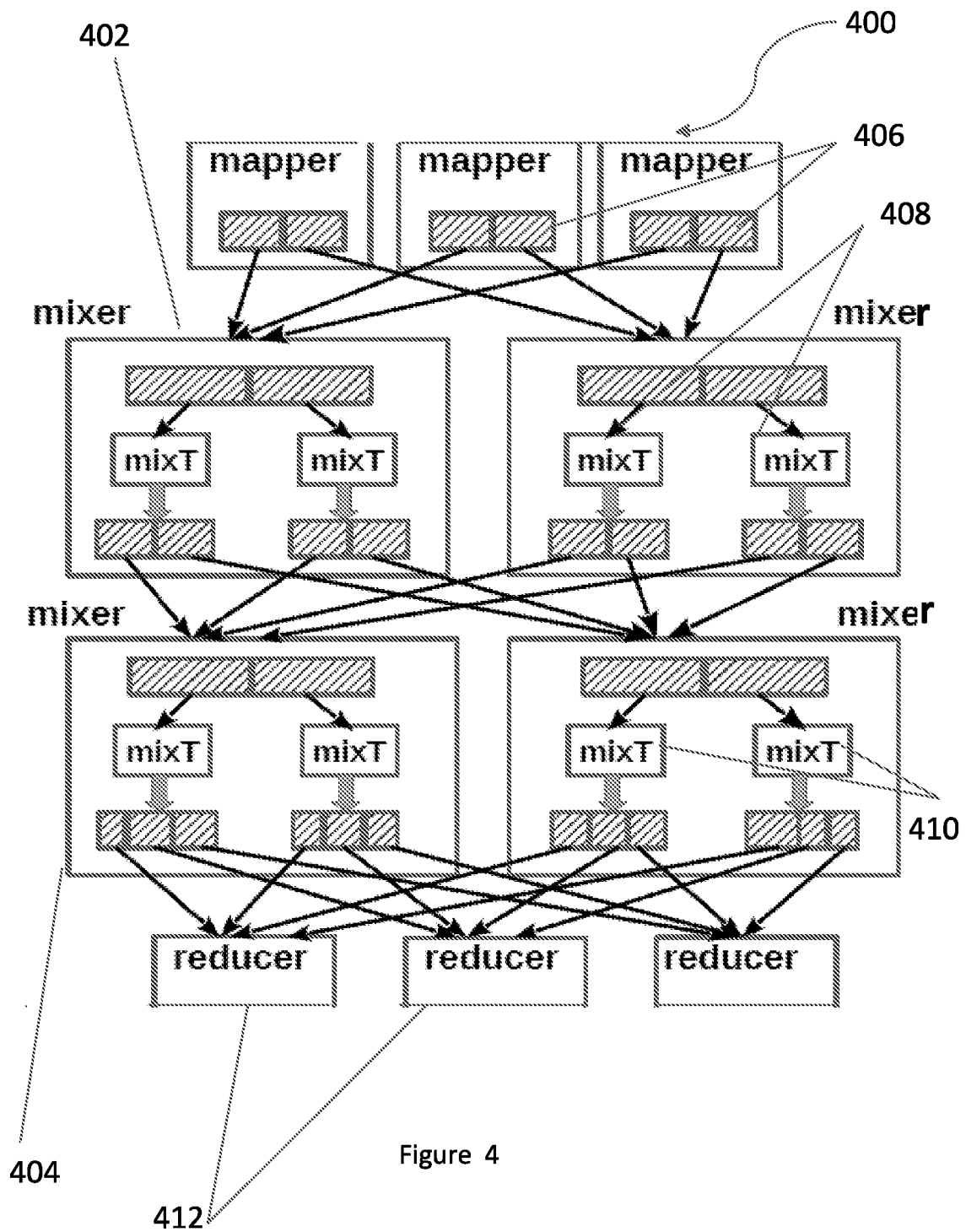
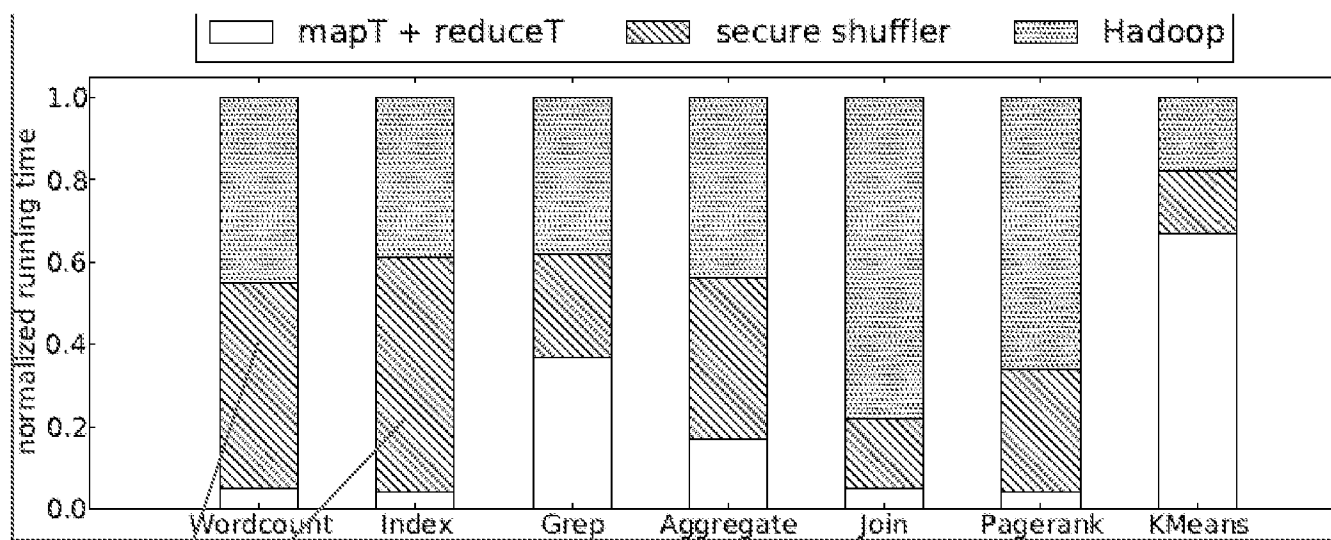


Figure 4

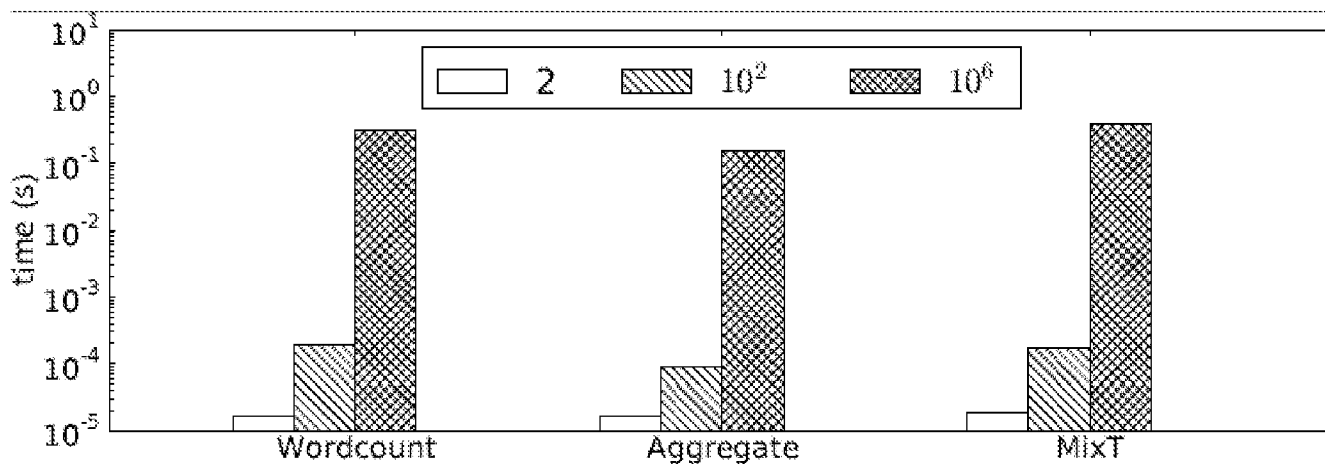


502

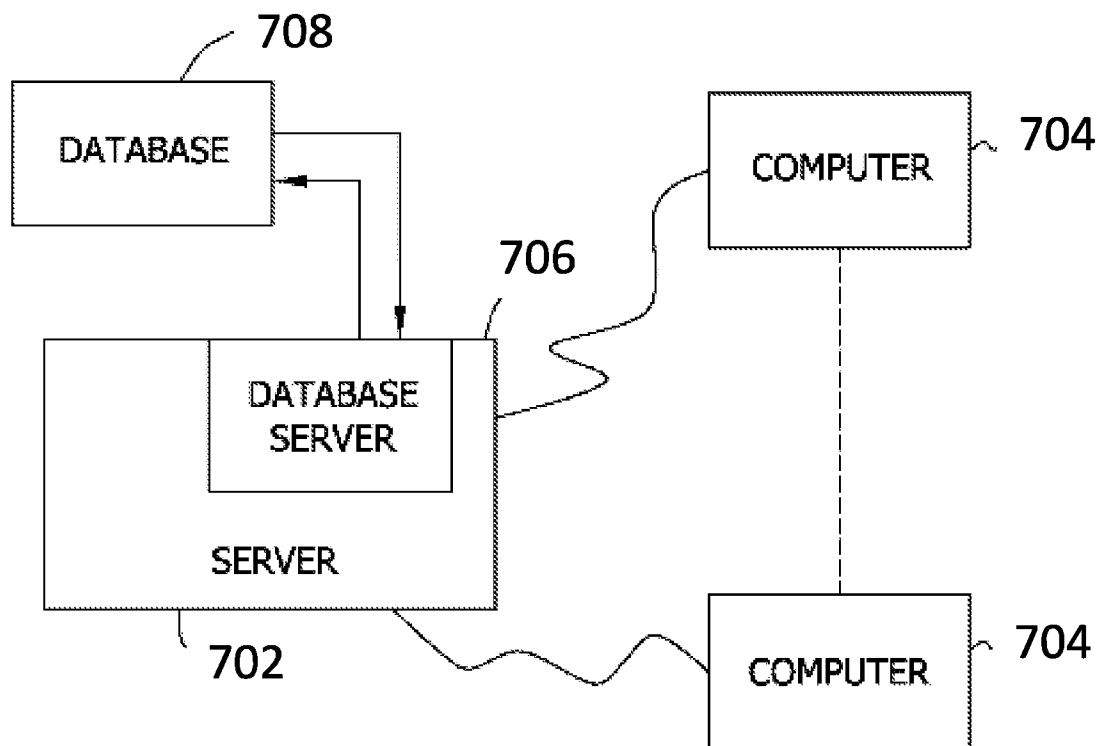
Figure 5

500

Figure 6



6/7



700 ↗

Figure 7

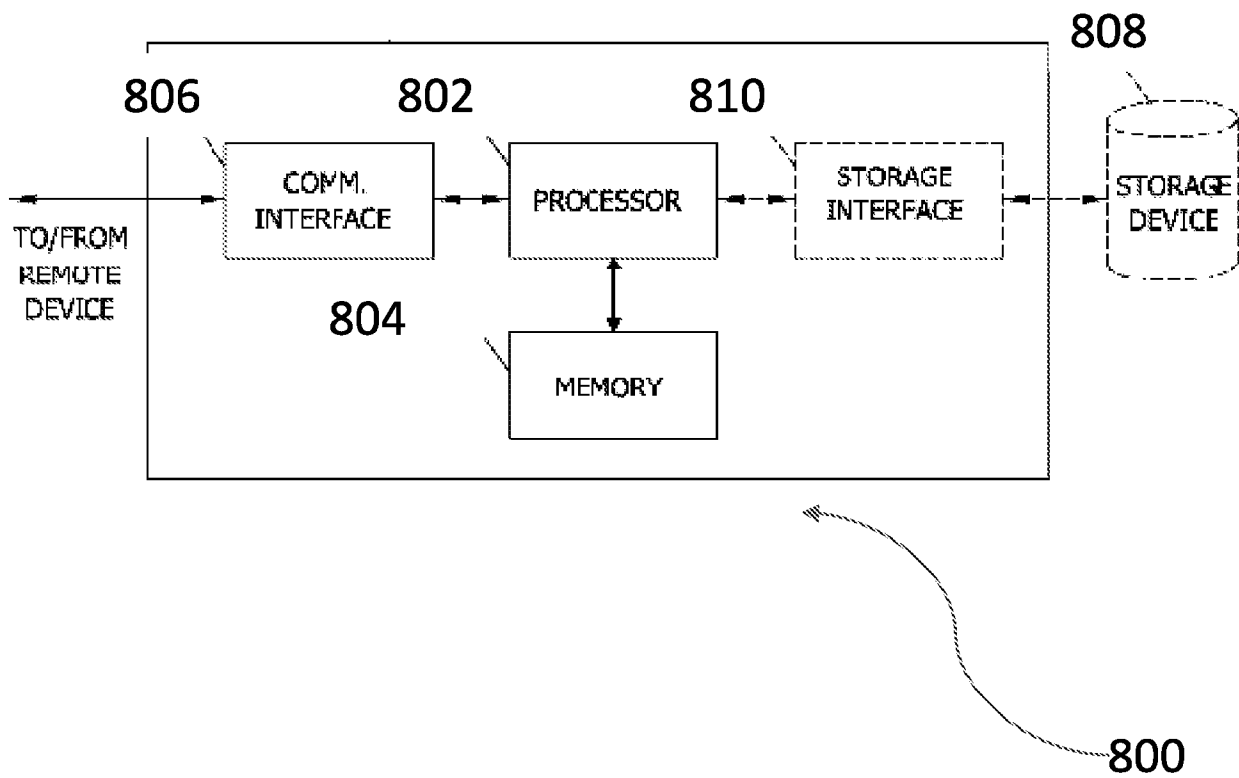


Figure 8

INTERNATIONAL SEARCH REPORT

International application No.

PCT/SG2016/050156

A. CLASSIFICATION OF SUBJECT MATTER

G06F 17/30 (2006.01)

According to International Patent Classification (IPC)

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)

G06F

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

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

EPODOC, WPI, FAMPAT, IEEE: mapreduce, spark, dryad, shuffle, mix, permute, tuple and related terms

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
P,X	DINH T. T. A., ET AL., M ² R: Enabling Strong Privacy in MapReduce Computation. <i>Proceedings of the 24th USENIX Conference on Security Symposium (SEC'15)</i> , 14 August 2015, pages 447-462 [Retrieved on 2016-05-26] <DOI: Not available> whole document	1-24
P,X	Anonymous Routing for Privacy-Preserving Distributed Computing. 31 October 2015 [Retrieved on 2016-05-26 from http://www.scholarbank.nus.edu.sg/handle/10635/121358?show=full] Chapter 3	1-24
A	LI J., ET AL., Improving the Shuffle of Hadoop MapReduce. <i>2013 IEEE 5th International Conference on Cloud Computing Technology and Science (CloudCom)</i> , 5 December 2013, Vol. 1, pages 266 - 273 [Retrieved on 2016-05-26] <DOI: 10.1109/CLOUDCOM.2013.42> whole document	

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

☒ See patent family annex.

*Special categories of cited documents:

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

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

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

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

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

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

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

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

"&" document member of the same patent family

Date of the actual completion of the international search

27/05/2016 (day/month/year)

Date of mailing of the international search report

09/06/2016 (day/month/year)

Name and mailing address of the ISA/SG

Intellectual Property Office of Singapore
 51 Bras Basah Road
 #01-01 Manulife Centre
 Singapore 189554

Email: pct@ipos.gov.sg

Authorized officer

Cai Chengjie (Dr)

IPOS Customer Service Tel. No.: (+65) 6339 8616

INTERNATIONAL SEARCH REPORT

International application No.

PCT/SG2016/050156**C (Continuation). DOCUMENTS CONSIDERED TO BE RELEVANT**

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	WO 2012/135319 A1 (GOOGLE INC.) 4 October 2012 whole document	
A	US 2013/0179466 A1 (MIZOBUCHI Y., ET AL.) 11 July 2013 whole document	

INTERNATIONAL SEARCH REPORT

Information on patent family members

International application No.

PCT/SG2016/050156

Note: This Annex lists known patent family members relating to the patent documents cited in this International Search Report. This Authority is in no way liable for these particulars which are merely given for the purpose of information.

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
WO 2012/135319 A1	04/10/2012	AU 2012236577 A1	24/10/2013
		CN 103748579 A	23/04/2014
		EP 2695087 A1	12/02/2014
		US 2012/254193 A1	04/10/2012
US 2013/0179466 A1	11/07/2013	JP 2013-140510 A	18/07/2013