US 20180067857A1

(19) United States

(12) Patent Application Publication

Wang et al.

(10) Pub. No.: US 2018/0067857 A1

(43) Pub. Date: Mar. 8, 2018

(54) **EFFICIENT DATA CACHING MANAGEMENT IN SCALABLE MULTI-STAGE DATA PROCESSING SYSTEMS**

(71) Applicant: **Samsung Electronics Co., Ltd.**, Suwon-si (KR)

(72) Inventors: **Jiayin Wang**, Dorchester, MA (US); **Zhengyu Yang**, Boston, MA (US); **Thomas David Evans**, San Marcos, CA (US)

(21) Appl. No.: **15/423,384**

(22) Filed: **Feb. 2, 2017**

Related U.S. Application Data

(60) Provisional application No. 62/384,078, filed on Sep. 6, 2016.

Publication Classification

(51) **Int. Cl.**

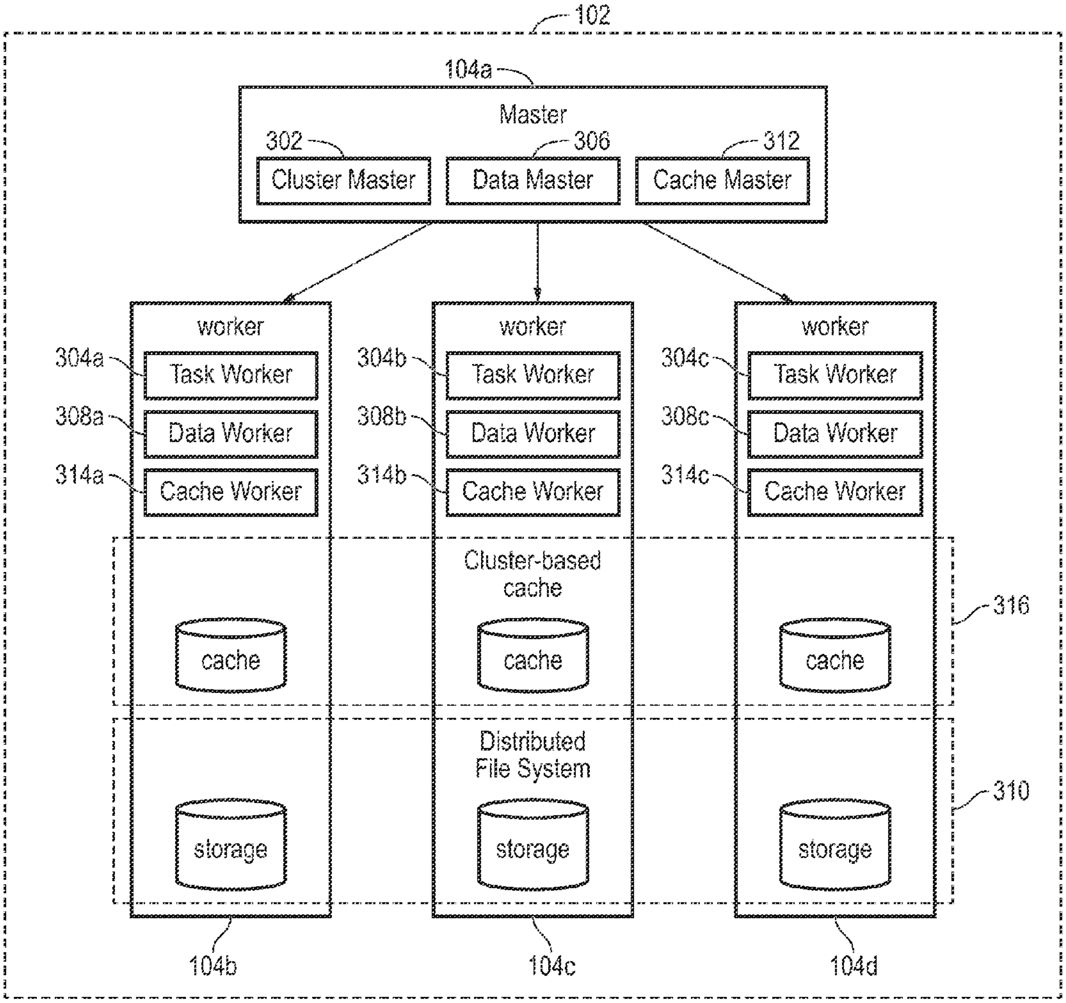
G06F 12/0811	(2006.01)
G06F 12/0862	(2006.01)

(52) **U.S. Cl.**

CPC ***G06F 12/0811*** (2013.01); *G06F 2212/62* (2013.01); *G06F 2212/602* (2013.01); ***G06F 12/0862*** (2013.01)

(57) **ABSTRACT**

According to some example embodiments, a method includes: receiving, by a processor, from a data source, a processing profile comprising input data blocks and a plurality of operations for executing using the input data blocks; executing, by the processor, one or more of the operations of the processing profile to generate a new output data after each of the executed one or more operations; storing, by the processor, the new output data from at least one of the one or more operations as intermediate cache data; and transmitting, by the processor, the new output data from a final operation from among the one or more operations to the data source for display thereby.



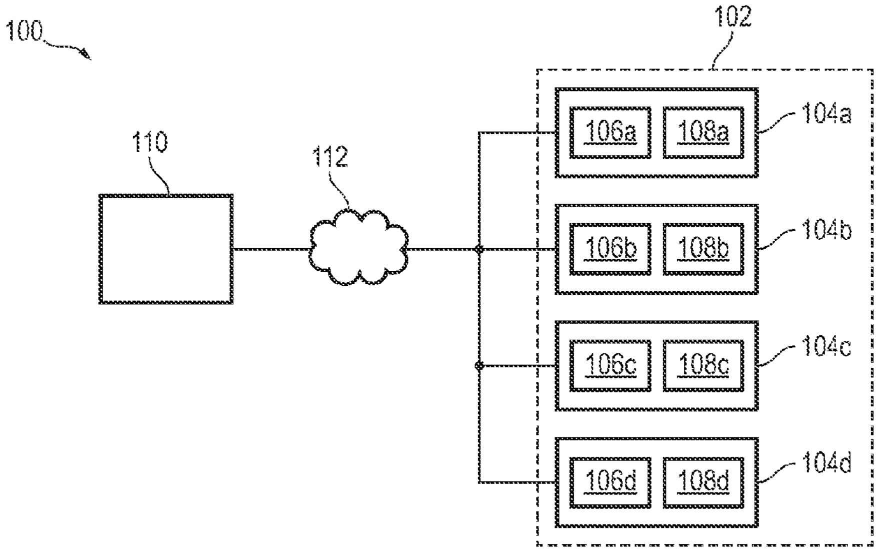

100
102
110
112
106a
108a
104a
106b
108b
104b
106c
108c
104c
106d
108d
104d

FIG. 1

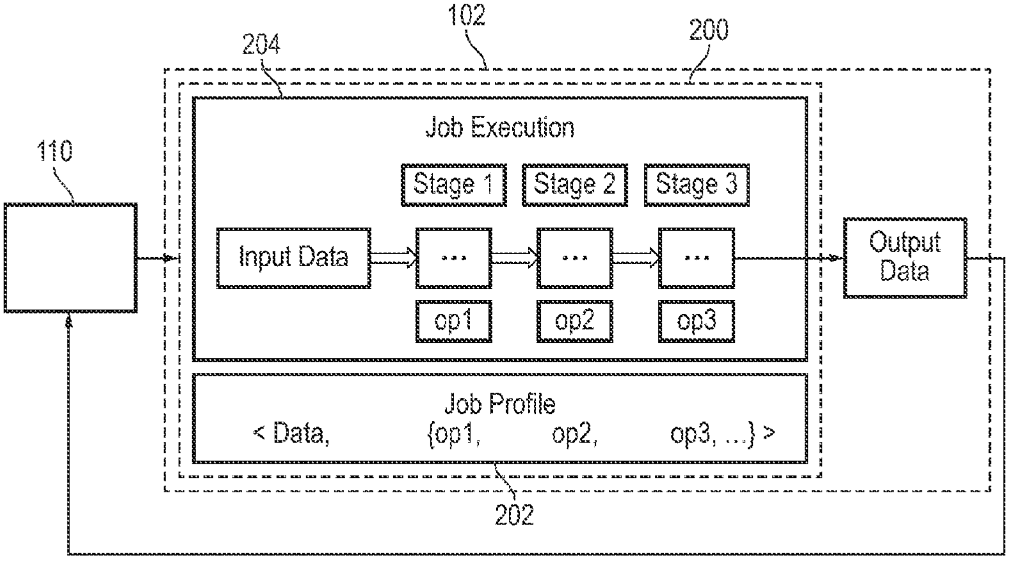

102
204
200
110
Job Execution
Stage 1
Stage 2
Stage 3
Input Data
...
...
...
op1
op2
op3
Output Data
Job Profile
< Data, {op1, op2, op3, ...} >
202

FIG. 2

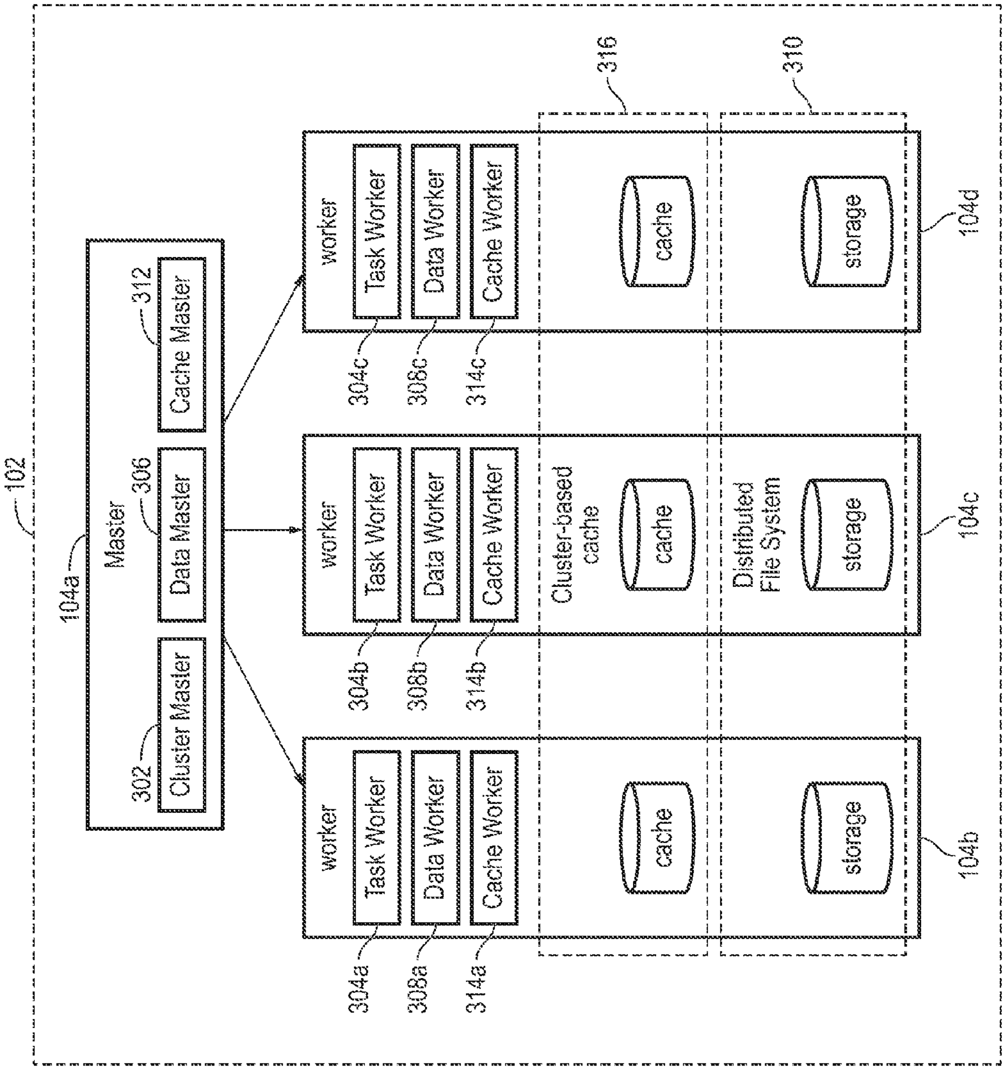

102
104a
Master
302
Cluster Master
306
Data Master
312
Cache Master
worker
Task Worker
Data Worker
Cache Worker
cache
storage
304a
308a
314a
104b
worker
Task Worker
Data Worker
Cache Worker
Cluster-based cache
cache
Distributed File System
storage
304b
308b
314b
104c
worker
Task Worker
Data Worker
Cache Worker
cache
storage
304c
308c
314c
104d
316
310

FIG. 3

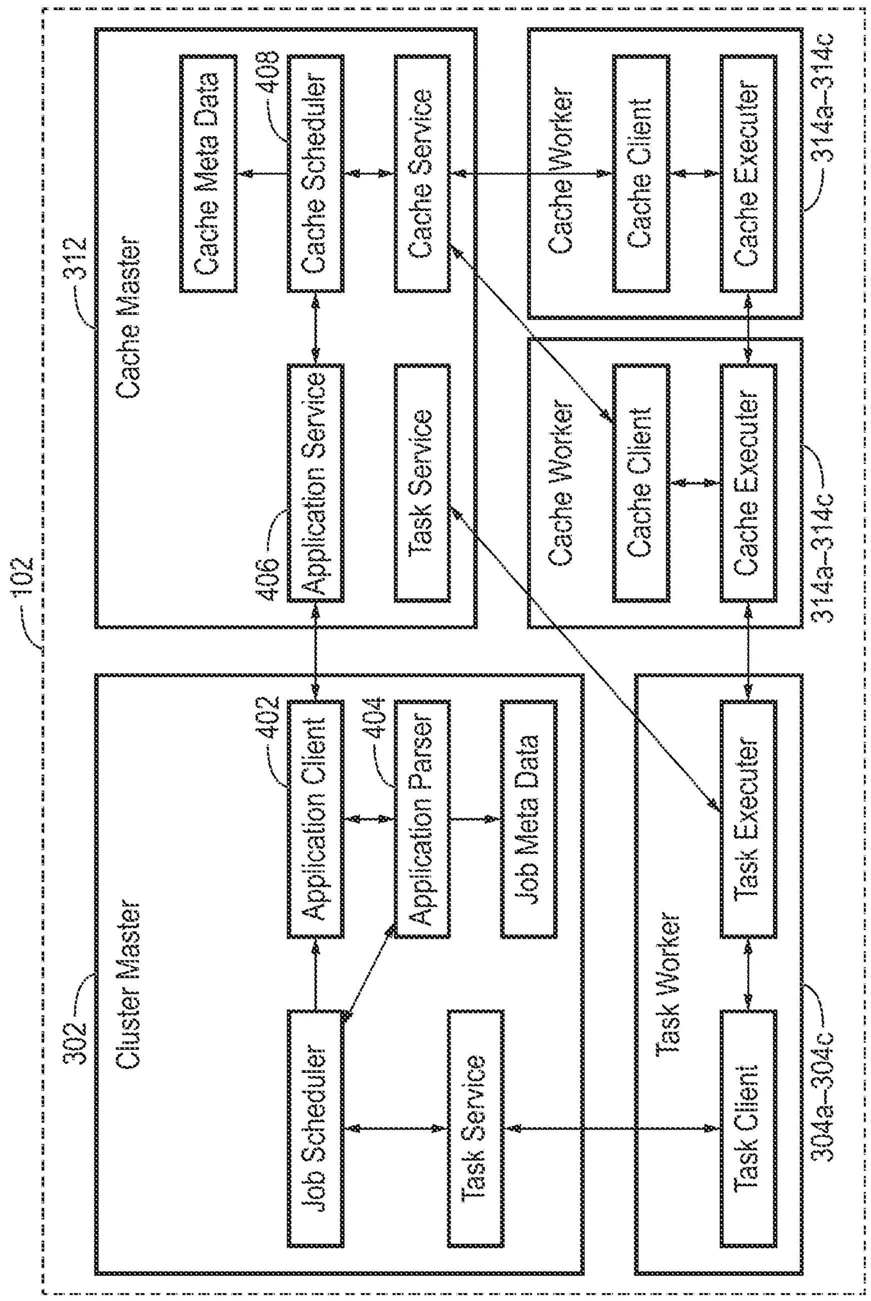

102
302
Cluster Master
Job Scheduler
Task Service
402
Application Client
404
Application Parser
Job Meta Data
Task Worker
Task Client
Task Executer
304a–304c
312
Cache Master
406
Application Service
Task Service
Cache Meta Data
408
Cache Scheduler
Cache Service
Cache Worker
Cache Client
Cache Executer
314a–314c
Cache Worker
Cache Client
Cache Executer
314a–314c

FIG. 4

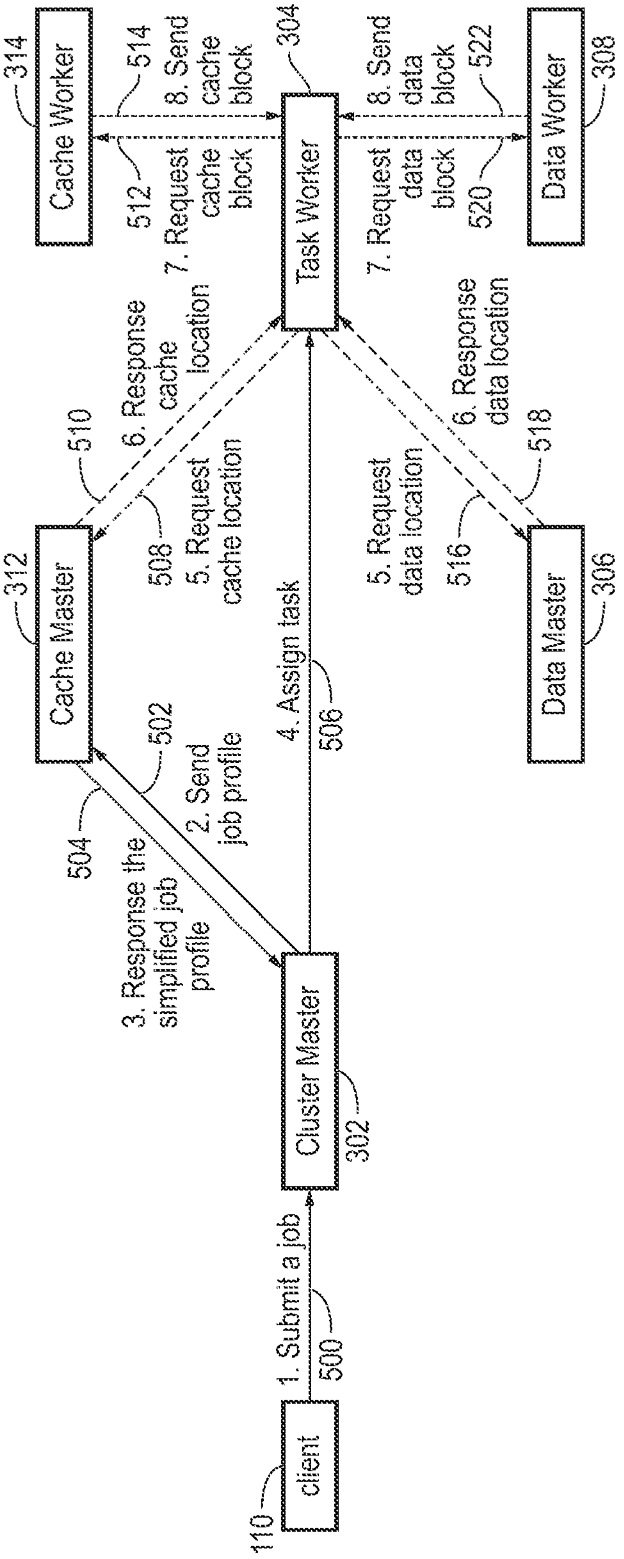

client
110
1. Submit a job
500
Cluster Master
302
2. Send job profile
502
3. Response the simplified job profile
504
Cache Master
312
4. Assign task
506
5. Request cache location
508
6. Response cache location
510
Task Worker
304
7. Request cache block
512
8. Send cache block
514
Cache Worker
314
5. Request data location
516
6. Response data location
518
Data Master
306
7. Request data block
520
8. Send data block
522
Data Worker
308

FIG. 5

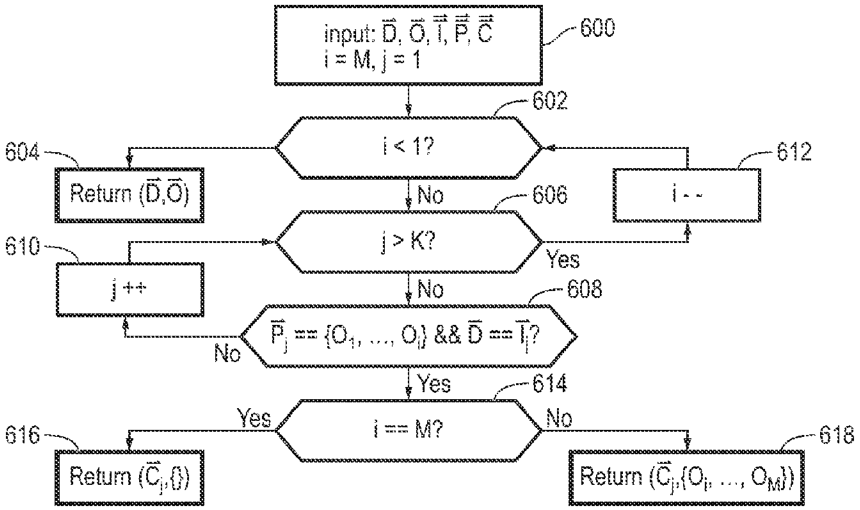

input: $\vec{D}$, $\vec{O}$, $\vec{I}$, $\vec{P}$, $\vec{C}$
i = M, j = 1
600
i < 1?
602
604
Return ($\vec{D}$,$\vec{O}$)
No
606
i - -
612
j > K?
Yes
610
j ++
No
608
$\vec{P}_j$ == {O_1, ..., O_i} && $\vec{D}$ == $\vec{I}_i$?
No
Yes
614
i == M?
Yes
No
616
Return ($\vec{C}_j$,{})
618
Return ($\vec{C}_j$,{O_i, ..., O_M})

FIG. 6

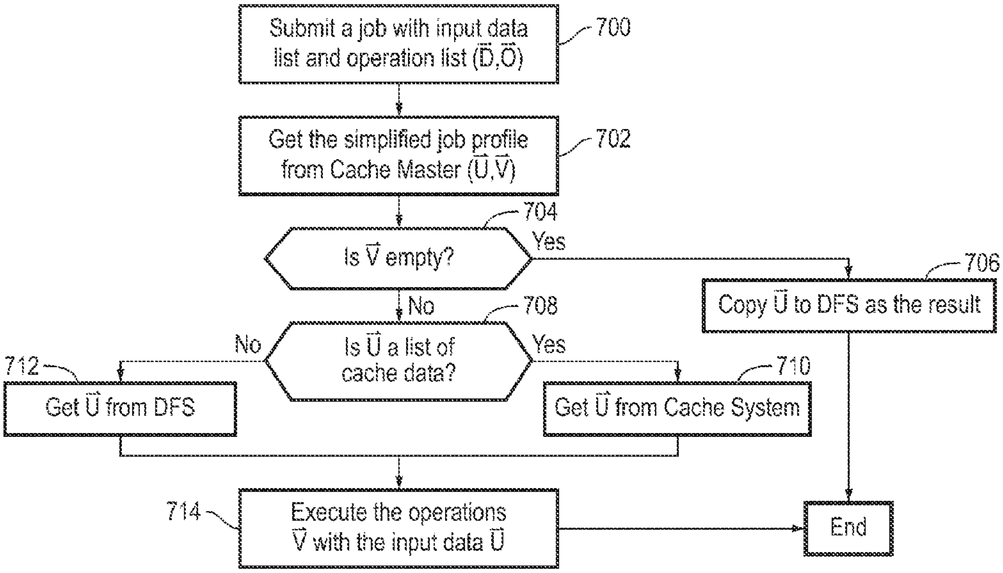

Submit a job with input data list and operation list ($\vec{D}$,$\vec{O}$)
700
Get the simplified job profile from Cache Master ($\vec{U}$,$\vec{V}$)
702
704
Is $\vec{V}$ empty?
Yes
706
Copy $\vec{U}$ to DFS as the result
No
708
Is $\vec{U}$ a list of cache data?
No
Yes
712
Get $\vec{U}$ from DFS
710
Get $\vec{U}$ from Cache System
714
Execute the operations $\vec{V}$ with the input data $\vec{U}$
End

FIG. 7

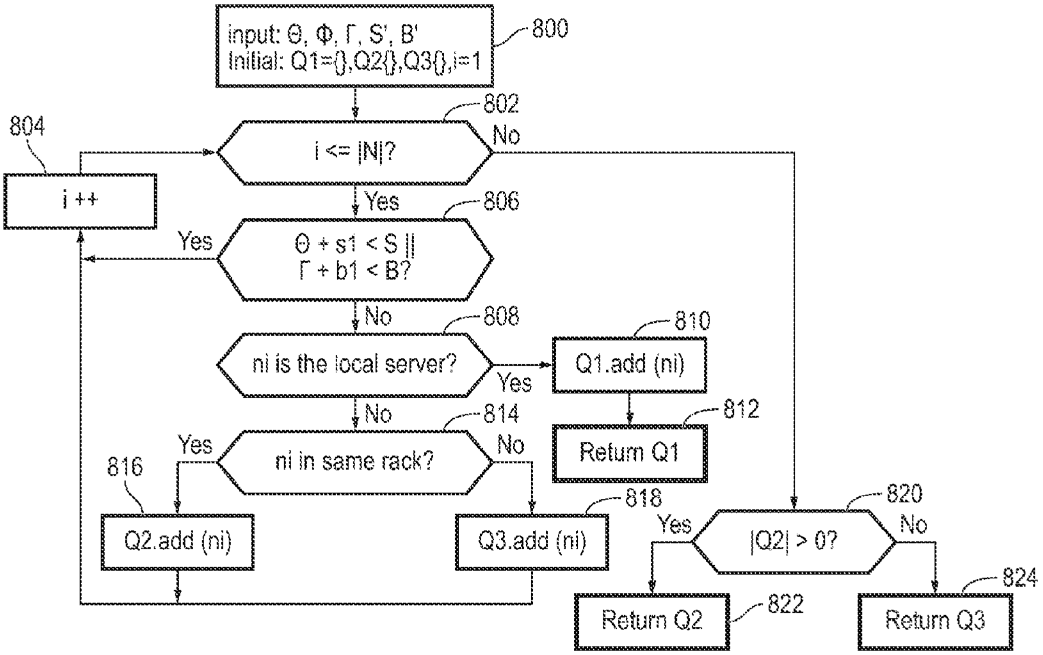

input: Θ, Φ, Γ, S', B'
Initial: Q1={},Q2{},Q3{},i=1
800
802
i <= |N|?
No
804
i ++
Yes
806
Yes
$\Theta + s1 < S \,||$
$\Gamma + b1 < B$?
No
808
ni is the local server?
Yes
810
Q1.add (ni)
812
Return Q1
No
814
Yes
ni in same rack?
No
816
Q2.add (ni)
818
Q3.add (ni)
820
Yes
|Q2| > 0?
No
824
Return Q2
822
Return Q3

FIG. 8

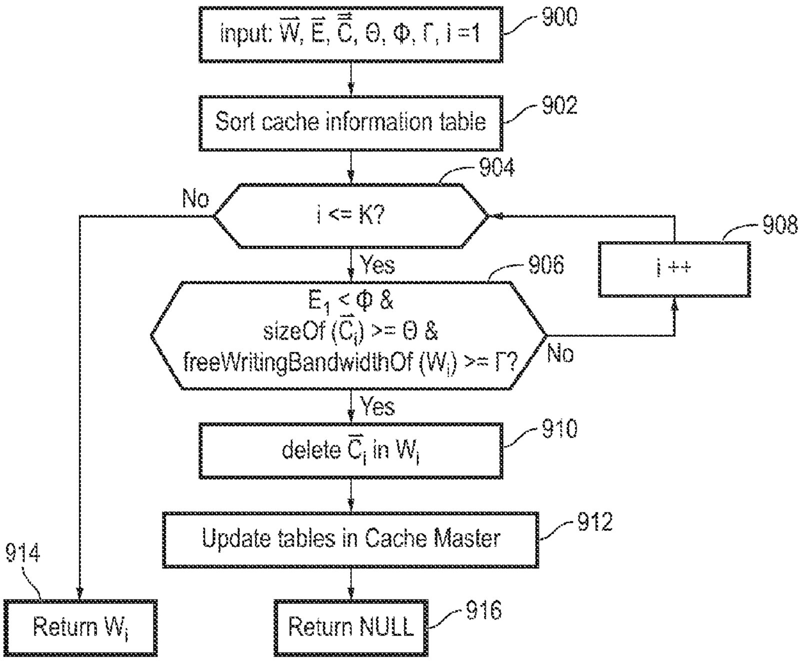

input: $\vec{W}$, $\vec{E}$, $\vec{\vec{C}}$, Θ, Φ, Γ, i =1
900
Sort cache information table
902
904
No
i <= K?
908
Yes
906
i ++
$E_1 < \Phi$ &
sizeOf ($\vec{C}_i$) >= Θ &
freeWritingBandwidthOf (W_i) >= Γ?
No
Yes
delete $\vec{C}_i$ in W_i
910
Update tables in Cache Master
912
914
Return W_i
Return NULL
916

FIG. 9

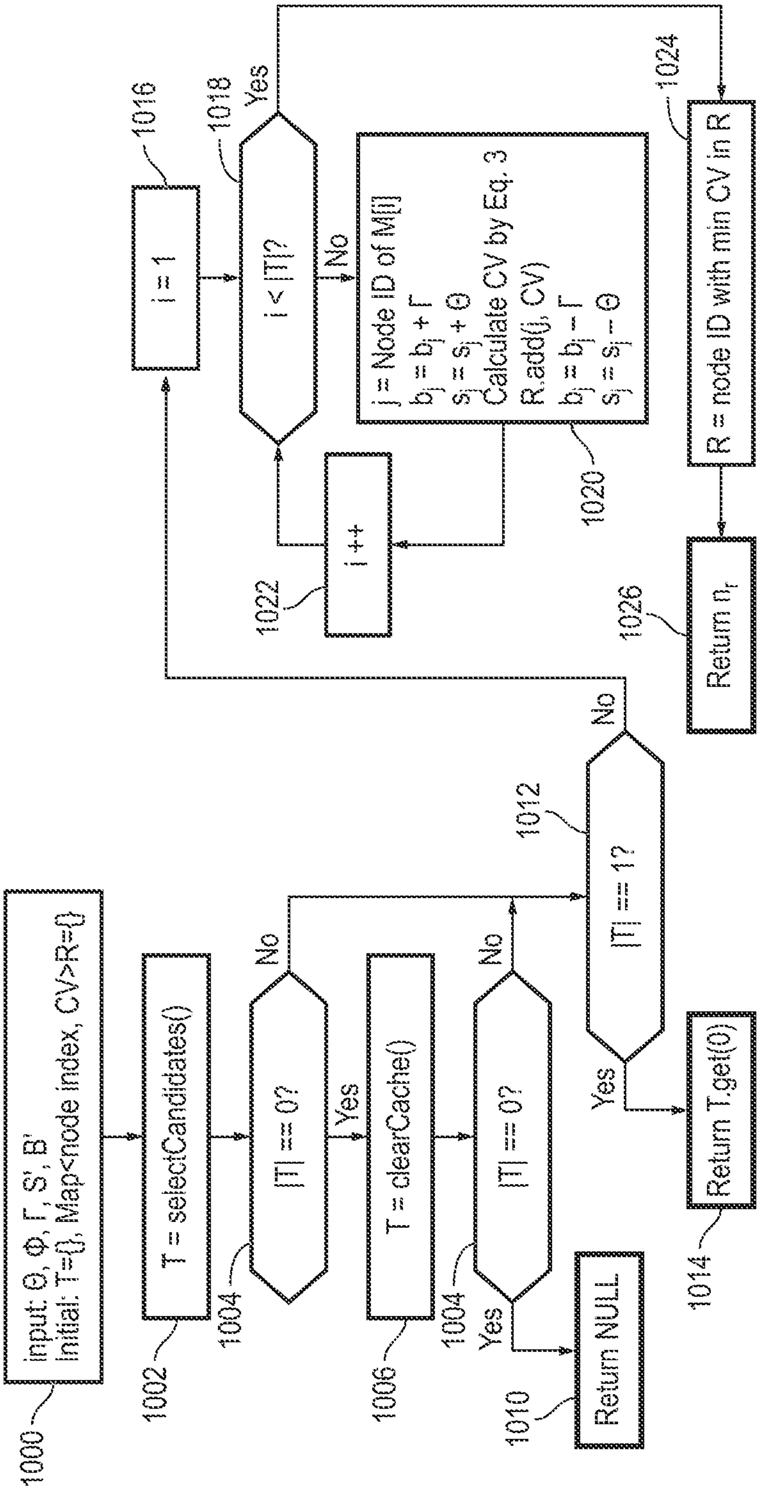

1000
input: Θ, Φ, Γ, S', B'
Initial: T={}, Map<node index, CV>R={}
1002
T = selectCandidates()
1004
|T| == 0?
Yes
No
1006
T = clearCache()
1004
|T| == 0?
Yes
No
1010
Return NULL
1012
|T| == 1?
Yes
No
1014
Return T.get(0)
1016
i = 1
1018
i < |T|?
Yes
No
1020
j = Node ID of M[i]
$b_j = b_j + Γ$
$s_j = s_j + Θ$
Calculate CV by Eq. 3
R.add(j, CV)
$b_j = b_j - Γ$
$s_j = s_j - Θ$
1022
i ++
1024
R = node ID with min CV in R
1026
Return n_r

FIG. 10

EFFICIENT DATA CACHING MANAGEMENT IN SCALABLE MULTI-STAGE DATA PROCESSING SYSTEMS

CROSS-REFERENCE TO RELATED APPLICATION

[0001] The present application claims priority to and the benefit of U.S. Provisional Patent Application No. 62/384,078, entitled "MEMORY SYSTEM AND METHOD OF USING THE SAME", filed in the United States Patent and Trademark Office on Sep. 6, 2016, the entire content of which is incorporated herein by reference.

FIELD

[0002] Aspects of one or more example embodiments of the present invention relate to a system and method for efficient data caching management in scalable multi-stage data processing systems.

BACKGROUND

[0003] Improvements in big data analytics and cloud computing in recent years have given rise to systems and infrastructures that enable various entities to host or rent processing clusters to analyze and process data. In a large-scale computing cluster, various big data processing jobs may include a sequence of multiple stages or operations, in which intermediate data is generated after each stage, but the intermediate data may not be saved or stored for subsequent use during subsequent processing jobs.

[0004] The above information discussed in this Background section is only for enhancement of understanding of the background of the described technology and therefore it may contain information that does not constitute prior art that is already known to a person having ordinary skill in the art.

SUMMARY

[0005] Aspects of one or more example embodiments of the present invention relate to a system and method for efficient data caching management in scalable multi-stage data processing systems.

[0006] According to some example embodiments, a system includes: a processor; and a memory coupled to the processor, wherein the memory stores instructions that, when executed by the processor, cause the processor to: receive, from a data source, a processing profile comprising input data blocks and a plurality of operations for executing using the input data blocks; determine whether or not a block of stored intermediate cache data corresponds to an operation from among of the plurality of operations; in response to determining the block of stored intermediate cache data corresponds to the operation from among the plurality of operations, generate a simplified processing profile based on the block of stored intermediate cache data; execute the simplified processing profile by generating a new output data after each operation of the simplified processing profile; store the new output data from at least one operation as intermediate cache data; and transmit the output data from a final operation in the simplified processing profile to the data source for display thereby.

[0007] According to some example embodiments, the instructions further cause the processor to, in response to determining the block of stored intermediate cache data corresponds to the operation from among the plurality of operations, identify a location of the stored intermediate cache data among a plurality of worker nodes.

[0008] According to some example embodiments, generating the simplified processing profile comprises removing the operation corresponding to the block of stored intermediate cache data.

[0009] According to some example embodiments, the simplified processing profile comprises a subset of the plurality of operations of the processing profile without the operation corresponding to the block of stored intermediate cache data.

[0010] According to some example embodiments, the instructions further cause the processor to: identify a candidate worker node from among a plurality of worker nodes for storing the new output data according to a load balance calculation of at least one of storage space of each of the worker nodes and input/output bandwidth of each of the worker nodes; and store the new output data at the identified candidate worker node.

[0011] According to some example embodiments, the instructions further cause the processor to: identify whether or not there is sufficient space among a plurality of worker nodes to store the new output data; and in response to determining there is not sufficient space among the plurality of worker nodes, clear a block of pre-stored intermediate cache data having a lower priority level than the new output data.

[0012] According to some example embodiments, a method includes: receiving, by a processor, from a data source, a processing profile comprising input data blocks and a plurality of operations for executing using the input data blocks; determining, by the processor, whether or not a block of stored intermediate cache data corresponds to an operation from among of the plurality of operations; in response to determining the block of stored intermediate cache data corresponds to the operation from among the plurality of operations, removing, by the processor, the operation from the processing profile to generate a simplified processing profile; executing, by the processor, the simplified processing profile by generating a new output data after each operation of the simplified processing profile; storing, by the processor, the new output data from at least one operation as intermediate cache data; and transmitting, by the processor, the output data from a final operation in the simplified processing profile to the data source for display thereby.

[0013] According to some example embodiments, the method further includes, in response to determining the block of stored intermediate cache data corresponds to the operation from among the plurality of operations, identifying, by the processor, a location of the stored intermediate cache data among a plurality of worker nodes.

[0014] According to some example embodiments, generating the simplified processing profile includes removing, by the processor, the operation corresponding to the block of stored intermediate cache data.

[0015] According to some example embodiments, the simplified processing profile includes a subset of the plurality of operations of the processing profile without the operation corresponding to the block of stored intermediate cache data.

[0016] According to some example embodiments, the method further includes identifying, by the processor, a candidate worker node from among a plurality of worker nodes for storing the new output data according to a load balance calculation of at least one of storage space of each of the worker nodes and input/output bandwidth of each of the worker nodes.

[0017] According to some example embodiments, the method further includes storing, by the processor, the new output data at the identified candidate worker node.

[0018] According to some example embodiments, the method further includes: identifying, by the processor, whether or not there is sufficient space among a plurality of worker nodes to store the new output data; and in response to determining there is not sufficient space among the plurality of worker nodes, clearing, by the processor, a block of pre-stored intermediate cache data having a lower priority level than the new output data.

[0019] According to some example embodiments, a method includes: receiving, by a processor, from a data source, a processing profile comprising input data blocks and a plurality of operations for executing using the input data blocks; executing, by the processor, one or more of the operations of the processing profile to generate a new output data after each of the executed one or more operations; storing, by the processor, the new output data from at least one of the one or more operations as intermediate cache data; and transmitting, by the processor, the new output data from a final operation from among the one or more operations to the data source for display thereby.

[0020] According to some example embodiments, the method further includes: determining, by the processor, whether or not a block of stored intermediate cache data corresponds to an operation from among of the plurality of operations; and in response to determining the block of stored intermediate cache data corresponds to the operation from among the plurality of operations, removing, by the processor, the operation from the processing profile to generate a simplified processing profile.

[0021] According to some example embodiments, the method further includes in response to determining the block of stored intermediate cache data corresponds to the operation from among the plurality of operations, identifying, by the processor, a location of the stored intermediate cache data among a plurality of worker nodes.

[0022] According to some example embodiments, the simplified processing profile includes a subset of the plurality of operations of the processing profile without the operation corresponding to the block of stored intermediate cache data, and the method further includes executing, by the processor, each of the plurality of operations among the subset.

[0023] According to some example embodiments, the method further includes identifying, by the processor, a candidate worker node from among a plurality of worker nodes for storing the new output data according to a load balance calculation of at least one of storage space of each of the worker nodes and input/output bandwidth of each of the worker nodes.

[0024] According to some example embodiments, the method further includes storing, by the processor, the new output data at the identified candidate worker node.

[0025] According to some example embodiments, the method further includes: identifying, by the processor, whether or not there is sufficient space among a plurality of worker nodes to store the new output data; and in response to determining there is not sufficient space among the plurality of worker nodes, clearing, by the processor, a block of pre-stored intermediate cache data having a lower priority level than the new output data.

BRIEF DESCRIPTION OF THE DRAWINGS

[0026] A more complete appreciation of the present invention, and many of the attendant features and aspects thereof, will become more readily apparent as the invention becomes better understood by reference to the following detailed description when considered in conjunction with the accompanying drawings in which like reference symbols indicate like components, wherein:

[0027] FIG. **1** illustrates a data processing system, according to some example embodiments of the present invention;

[0028] FIG. **2** depicts an example data processing job transmitted from a data source to a data processing cluster, according to some example embodiments of the present invention;

[0029] FIG. **3** illustrates a block diagram showing aspects of an example architecture of a data processing cluster, according to some example embodiments of the present invention;

[0030] FIG. **4** illustrates further details of a data processing cluster, according to some example embodiments of the present invention;

[0031] FIG. **5** illustrates further details of the workflow described above for a data processing job, according to example embodiments of the present invention;

[0032] FIG. **6** illustrates aspects of a process or function for simplifying a data processing job, according to some example embodiments of the present invention;

[0033] FIG. **7** illustrates aspects of a process or function for executing a data processing job according to some example embodiments of the present invention;

[0034] FIG. **8** illustrates aspects of a process for selecting candidates for storing intermediate cache data, according to some example embodiments of the present invention;

[0035] FIG. **9** illustrates aspects of a process for clearing stored cache data, according to some example embodiments of the present invention; and

[0036] FIG. **10** illustrates aspects of a process for selecting a cache worker, according to some example embodiments of the present invention.

DETAILED DESCRIPTION

[0037] Aspects of example embodiments of the present invention relate to a system and method for efficient data caching management in scalable multi-stage data processing systems.

[0038] Features of the inventive concept and methods of accomplishing the same may be understood more readily by reference to the following detailed description of embodiments and the accompanying drawings. Hereinafter, example embodiments will be described in more detail with reference to the accompanying drawings, in which like reference numbers refer to like elements throughout. The present invention, however, may be embodied in various different forms, and should not be construed as being limited to only the illustrated embodiments herein. Rather, these embodiments are provided as examples so that this disclosure will be thorough and complete, and will fully convey the aspects and features of the present invention to those skilled in the art. Accordingly, processes, elements, and techniques that are not necessary to those having ordinary skill in the art for a complete understanding of the aspects and features of the present invention may not be described. Unless otherwise noted, like reference numerals denote like elements throughout the attached drawings and the written description, and thus, descriptions thereof will not be repeated. In the drawings, the relative sizes of elements, layers, and regions may be exaggerated for clarity.

[0039] It will be understood that, although the terms “first,” “second,” “third,” etc., may be used herein to describe various elements, components, regions, layers and/or sections, these elements, components, regions, layers and/or sections should not be limited by these terms. These terms are used to distinguish one element, component, region, layer or section from another element, component, region, layer or section. Thus, a first element, component, region, layer or section described below could be termed a second element, component, region, layer or section, without departing from the spirit and scope of the present invention.

[0040] Spatially relative terms, such as “beneath,” “below,” “lower,” “under,” “above,” “upper,” and the like, may be used herein for ease of explanation to describe one element or feature’s relationship to another element(s) or feature(s) as illustrated in the figures. It will be understood that the spatially relative terms are intended to encompass different orientations of the device in use or in operation, in addition to the orientation depicted in the figures. For example, if the device in the figures is turned over, elements described as “below” or “beneath” or “under” other elements or features would then be oriented “above” the other elements or features. Thus, the example terms “below” and “under” can encompass both an orientation of above and below. The device may be otherwise oriented (e.g., rotated 90 degrees or at other orientations) and the spatially relative descriptors used herein should be interpreted accordingly.

[0041] It will be understood that when an element, layer, region, or component is referred to as being “on,” “connected to,” or “coupled to” another element, layer, region, or component, it can be directly on, connected to, or coupled to the other element, layer, region, or component, or one or more intervening elements, layers, regions, or components may be present. In addition, it will also be understood that when an element or layer is referred to as being “between” two elements or layers, it can be the only element or layer between the two elements or layers, or one or more intervening elements or layers may also be present.

[0042] The terminology used herein is for the purpose of describing particular embodiments only and is not intended to be limiting of the present invention. As used herein, the singular forms “a” and “an” are intended to include the plural forms as well, unless the context clearly indicates otherwise. It will be further understood that the terms “comprises,” “comprising,” “includes,” and “including,” when used in this specification, specify the presence of the stated features, integers, steps, operations, elements, and/or components, but do not preclude the presence or addition of one or more other features, integers, steps, operations, elements, components, and/or groups thereof. As used herein, the term “and/or” includes any and all combinations of one or more of the associated listed items. Expressions such as “at least one of,” when preceding a list of elements, modify the entire list of elements and do not modify the individual elements of the list.

[0043] As used herein, the term “substantially,” “about,” and similar terms are used as terms of approximation and not as terms of degree, and are intended to account for the inherent deviations in measured or calculated values that would be recognized by those of ordinary skill in the art. Further, the use of “may” when describing embodiments of the present invention refers to “one or more embodiments of the present invention.” As used herein, the terms “use,” “using,” and “used” may be considered synonymous with the terms “utilize,” “utilizing,” and “utilized,” respectively. Also, the term “exemplary” is intended to refer to an example or illustration.

[0044] When a certain embodiment may be implemented differently, a specific process order may be performed differently from the described order. For example, two consecutively described processes may be performed substantially at the same time or performed in an order opposite to the described order.

[0045] The electronic or electric devices and/or any other relevant devices or components according to embodiments of the present invention described herein may be implemented utilizing any suitable hardware, firmware (e.g. an application-specific integrated circuit), software, or a combination of software, firmware, and hardware. For example, the various components of these devices may be formed on one integrated circuit (IC) chip or on separate IC chips. Further, the various components of these devices may be implemented on a flexible printed circuit film, a tape carrier package (TCP), a printed circuit board (PCB), or formed on one substrate. Further, the various components of these devices may be a process or thread, running on one or more processors, in one or more computing devices, executing computer program instructions and interacting with other system components for performing the various functionalities described herein. The computer program instructions are stored in a memory which may be implemented in a computing device using a standard memory device, such as, for example, a random access memory (RAM). The computer program instructions may also be stored in other non-transitory computer readable media such as, for example, a CD-ROM, flash drive, or the like. Also, a person of skill in the art should recognize that the functionality of various computing devices may be combined or integrated into a single computing device, or the functionality of a particular computing device may be distributed across one or more other computing devices without departing from the spirit and scope of the exemplary embodiments of the present invention.

[0046] Unless otherwise defined, all terms (including technical and scientific terms) used herein have the same meaning as commonly understood by one of ordinary skill in the art to which the present invention belongs. It will be further understood that terms, such as those defined in commonly used dictionaries, should be interpreted as having a meaning that is consistent with their meaning in the context of the relevant art and/or the present specification, and should not be interpreted in an idealized or overly formal sense, unless expressly so defined herein.

[0047] With the rise of big data analytics and cloud computing, more and more companies hos or rent processing clusters to analyze various types of data to improve their business. In a large-scale computing cluster, large data processing jobs include a sequence of processing stages, where each stage represents or corresponds to a generally defined data operation or transformation. For example, each stage may execute operations such as filtering, merging, sorting, and mapping of data. For a sequence of multiple stages corresponding to an overall processing job, intermediate data are created or generated from each stage except for the final stage. When such intermediate data is generated and then deleted, it may not be reused or shared by other jobs. In many circumstances, however, the intermediate data, if not deleted, would be usable by other processing jobs, for example, when subsequent jobs include stages that have the same input data and operations. Therefore, deletion of the intermediate data may lead to inefficiencies in terms of the use of resources for the processing cluster.

[0048] As will be described in more detail below, embodiments of the present invention may be configured to harness the data sharing among data processing stages across different jobs or operations to avoid or reduce instances of redundant data processing. For example, embodiments of the present invention may operate to identify different stages that utilize the same input data and perform the same processing operation, but serve different jobs. By sharing data between different stages that are executing the same or similar function on the same or similar input, the system may reduce the overall amount of system resources by executing only one of these stages, storing the output data as intermediate cache data, then enabling the subsequent stages to utilize the output data and proceed to the next stage for their corresponding job.

[0049] Embodiments of the present invention may further utilize an efficient cluster-wide caching scheme for hosting the intermediate data to support consecutive data sharing. This scheme may operate, for example, by utilizing aspects of a cluster-based caching system, described in more detail below.

[0050] Some embodiments of the present invention include a mechanism for describing data dependency for each stage in a job's profile. When a job is submitted to the processing cluster, according to some embodiments of the present invention, the profile or metadata of the job includes additional information describing the data dependency of each stage thereby enabling sharing of intermediate cache data.

[0051] Some embodiments of the present invention include an architecture of a cluster-based caching system based on a paralleling big-data processing system.

[0052] Some embodiments of the present invention include an efficient cluster-wide caching scheme for hosting the intermediate data to support consecutive data sharing. This scheme may be operated by utilizing the architecture of a cluster-based caching system. Embodiments of the present invention may first examine and analyze the executions of representative big data processing jobs, understand and model the characteristics of data access (especially the intermediate data access), and then simplify the stages of jobs by reusing the intermediate data in the cache system.

[0053] Some embodiments of the present invention include cluster-based caching system on a big data processing platform for intermediate data sharing and reuse by different applications, which may significantly improve the overall performance of the platform.

[0054] FIG. **1** illustrates a data processing system **100** according to some example embodiments of the present invention. As shown in FIG. **1**, the data processing system **100** includes a data processing cluster **102**, which includes a plurality of nodes **104***a*-**104***d* (the number of nodes **104***a*-**104***d* is not limited to the number illustrated in FIG. **1**, and may include fewer or additional nodes according to the design of the data processing system **100**). Each of the nodes **104***a*-**104***d* may be in electronic communication with one another, for example, through a local area network (LAN), private wide area network (WAN), and/or a public wide area network such as, for example, the Internet, in a wired or wireless configuration.

[0055] Each of the nodes **104***a*-**104***d* includes a processor **106***a*-**106***d*, respectively. Additionally, each of the nodes includes a memory **108***a*-**108***d*, respectively. The memories **108***a*-**108***d* may include volatile memory (e.g., random access memory (RAM), such as dynamic RAM) and non-volatile memory (e.g., a hard disk drive (HDD)), which may serve as long-term storage. Different nodes **104***a*-**104***d* may be distributed across various geographic or system locations. For example, according to some embodiments, one or more nodes **104***a*-**104***d* may be located at a same general geographic location (for example, in the same server rack), while one or more nodes may be located at a different geographic location.

[0056] Collectively, according to embodiments of the present invention, the data processing cluster **102** operates as a distributed data processing system using any suitable cluster computing framework known in the art. For example, according to some embodiments of the present invention, the distributed data processing cluster or system **102** may operate within the Apache Spark framework, and/or may provide a functional application programming interface (API) for manipulating data at scale, in-memory data caching and reuse across computations.

[0057] The data processing cluster **102** may further be in electronic communication with one or more data sources or users **110** over a communication network **112**. That is the data sources or users **110** may be located remotely or externally with respect to the data processing cluster **102**. The communication network **112** may be for example, a LAN, private WAN, and/or a public WAN such as, for example, the Internet. According to some embodiments, the data source **110** may also be local with respect to one or more of the nodes **104***a*-**104***d*. According to embodiments of the present invention, the data processing cluster **102** is configured to receive a data processing job profile and input data from the data source **110**, execute the data processing job on behalf of the data source **110**, and return a result of the data processing job to the data source **110**, as will be described in more detail herein. According to some embodiments, the data source or user **110** may include, for example, a computer system including a processor and a memory (similar to **106** and **108**), one or more user input devices, and a display panel for displaying information (e.g., output data received from the data processing cluster **102**).

[0058] According to some embodiments of the present invention, a node in the data processing cluster **102** (e.g., the node **104***a*) is designated or operates as a master node, and other nodes (e.g., nodes **104***b*-**104***c*) are designated or operate as worker nodes. The master node is responsible for managing worker nodes and scheduling the jobs. Worker nodes are responsible for data storing and task execution.

[0059] According to some embodiments of the present invention, a data source or user **100** submits a job to the data processing cluster **102** through the master node. One job is then split into multiple tasks or operations and one task is assigned one block of input data. During execution of a job, one piece of intermediate data is created after each stage or operation until the final output data is generated following the final stage or operation of the job. The intermediate data is the output data from each intermediate stage or operation of the data processing job, and also serves as the input data of the next stage. After a data processing job is completed, according to embodiments of the present invention, intermediate data may be stored as intermediate cache data for reuse and/or sharing for other jobs.

[0060] Thus, as will be described in more detail below, embodiments of the present invention operate to harness data sharing among data processing stages across different jobs to avoid redundant data processing. In particular, embodiments of the present invention operate to identify stages of different data processing jobs that receive the same input data and perform the same processing operation, and then re-use or share the intermediate data among the different jobs.

[0061] By enabling sharing of the intermediate data, the data processing cluster **102** may reduce the overall system data processing, because repetitive instances of performing a particular operation on particular input data may be reduced or eliminated altogether. For example, after a particular data processing job executes a stage or operation having a particular input data, the output of the stage may be stored as intermediate cache data, and subsequent data processing jobs that have the same stage or operation with the same input data can simply utilize the intermediate cache data as the input for the next stage in their respective sequence of stages, rather than re-executing the operation that was already executed for the previous data processing job.

[0062] As part of the data processing system **100**, embodiments of the present invention may include an enhanced job profile to describe the input data and the operations of all the stages so that the data processing cluster (e.g., the data processing cluster **102**) can identify data processing jobs with the same input data and the same or similar operations. Additionally, as will be described in more detail below, embodiments of the present invention may include a cluster-based caching system or mechanism operating as part of the data processing cluster to manage the cached intermediate data, work with other components in the cluster to simplify the operations of new jobs, and provide and/or retrieve the intermediate cache data for reuse by subsequent data processing jobs. Further, embodiments of the present invention further include schemes for supporting sharing of intermediate cache data for subsequent jobs and for allocating or distributing the intermediate cache data in memory of the various nodes in the data processing cluster.

[0063] According to embodiments of the present invention, a data processing cluster (e.g., the data processing cluster **102**) may have H worker nodes (including, e.g., nodes **104***b*-**104***d*). Thus, the set N of worker nodes may be represented as N: N={n_1, n_2, . . . , n_H}. If S is the storage capacity and B is the maximum IO bandwidth for intermediate data caching in each worker node, then the set S': {s_1, s_2, . . . , s_H} and the set B': {b_1, b_2, . . . , b_H} represent the collective remaining capacities and IO bandwidth of each worker node for caching, respectively.

[0064] According to embodiments of the present invention, a data processing job submitted to the data processing cluster, includes two elements: the input data and a list of operations to be executed upon the input data to generate the final output data. The input data contains a list of data blocks $\vec{D}$; {D_i|i ∈ [1, N]}, where D_i represents the data block ID and N is the number of data blocks. Thus, the input data includes a set or list of data blocks, each having a corresponding data block ID (e.g., ranging from 1 to N, where N is the total number of data blocks). Additionally, the operation list $\vec{O}$; {O_i|i ∈ [1,M]}, where O, is the operation ID in stage i, and M is the number of stages or operations for the data processing job. According to embodiments of the present invention, both the data block ID and the operation ID are unique in the data processing cluster. For example, {{block1}, {op1, op2}} shows a data processing job with one data block block1 and two operations op1 and op2.

[0065] According to embodiments of the present invention, the data processing cluster **102** stores two tables of data, an operation table and a cache information table.

[0066] The operation table stores the input data for different stages, the operations executed for each stage, and the intermediate cache data for each stage. For example, according to some example embodiments of the present invention, an example operations table includes three columns, as illustrated in Table 1, below. A first column of the operation table may include a block list of input data, may include an operation list for each stage, and a third column may include a list of cached intermediate data blocks. Thus, for a table with K rows, the elements in the operations table may be a collection of three 2-D vectors which can be represented as: ($\vec{I}$, $\vec{P}$, $\vec{C}$), where $\vec{I}$; {$\vec{I}_i$|i ∈ [1, K]}, $\vec{P}$: {$\vec{P}_i$|i ∈ [1, K]}, and $\vec{C}$: {$\vec{C}_i$|i ∈ [1, K]}. For a list of cached data blocks $\vec{C}_i$ row i of the table, $\vec{I}_i$ is the list of original input data blocks and $\vec{P}_i$ is related the operation list. For instance, Table 1 shows the record in the operation table of a completed data processing job j1 whose intermediate data have been saved in the cache system. The input data of j1 contains two blocks: block1 and block2. There are three stages in j1 with operation IDs op1, op2, and op3. cache1 and cache2 are the cache IDs of the intermediate data after stage 1 and stage 2. There are two data blocks created after the third stage, cache3 and cache4. Additionally, the operation ID is unique for each operation and the cache ID is unique for each cached data block in the cache cluster. The original input data may be either the data saved in a distributed file system (DFS) of the data processing cluster or intermediate cache data stored in a cluster-based cache system of the data processing cluster.

TABLE 1

Original Input Data	Operation List	Cached Intermediate Data List
{block1, block2}	{op1}	{cache1}
{block1, block2}	{op1, op2}	{cache2}
{block1, block2}	{op1, op2, op3}	{cache3, cache4}

[0067] A second table stored as part of the data processing cluster **102** is the cache information table. The cache information table may be stored, for example, in a Cache Master memory of a master node, and includes a record for the location of each list of cache blocks and the priority of the cache list. According to some embodiments, a score may be utilized to represent a priority of a cached data list where, for example, the higher the score, the higher the priority is. Certain intermediate cache data may have a higher priority than other intermediate cache data, for example, if the intermediate cache data is utilized more frequently, utilizes more system resources to generate, and the like.

[0068] The cache information table may be formatted, for example, to include three columns as: $(\vec{C}, \vec{W}, \vec{E})$, where $\vec{C}$: $\{\vec{C}_i | i \in [1, K]\}$, $\vec{W}$: $\{W_i | i \in [1, K]\}$ and $\vec{E}$: $\{E_i | i \in [1, K]\}$. The cache information table may have the same number of rows as the operation table, represented as K. The collection of cache data lists $\vec{C}$ is the same as the one in the operation table. For the list of cache data IDs $\vec{C}_i$ in row i, $\vec{W}$ shows the node IDs of the data worker which contain this cache list and the score of the cache list is E_i. Table 2, below, shows an example of a cache information table.

TABLE 2

Cached Intermediate Data List	Data Worker List	Priority
{cache1}	worker1	score1
{cache2}	worker2	score2
{cache3, cache4}	worker3	score3

[0069] In addition to the cache information table and operation table, according to some embodiments of the present invention, a data processing job request from a user may include a data processing job profile, which includes information data dependency. Each stage in a data processing job may, for example, only have one input data. For example, the input data for an individual stage in a data processing job may be the original input data provided with the data processing job request from the data source or user, or may be intermediate data generated from the immediately preceding stage. Additionally, each stage may, for example, only have one output data which is provided as the only input to the next stage or constitutes the final output data for the data processing job. Thus, according to embodiments of the present invention, data processing jobs received from different data sources or users may share the same input data (or intermediate data) and one or more common processing stages. Embodiments of the present invention enable common stages, with common input, from different data processing jobs to be executed fewer times (e.g., only one time) and the output data and/or intermediate data can be shared among the different data processing jobs.

[0070] According to some embodiments of the present invention, a job profile for a data processing job is transmitted to the data processing cluster **102** from the data source or user **110**, which specifies the data input for each stage, thereby enabling data sharing for multiple data processing jobs. When submitting a data processing job, the data source or user **110** creates and transmits the job profile to the cluster master (e.g., the node **104***a*) of the data processing cluster **102** with the original input data files and a list of operations of all of the stages in the data processing job. According to some embodiments of the present invention, the job profile includes unique data file names and each operation has a unique operation ID. Such job profiles can be managed by the cluster master (e.g., of the master node) for the further scheduling.

[0071] FIG. **2** depicts an example data processing job **200** transmitted from a data source (e.g., the data source **110**) to a data processing cluster (e.g., the data processing cluster **102**) according to some example embodiments of the present invention. The data processing job **200** includes a sequence of processing stages (e.g., stages 1-3), and each stage represents a generally defined data operation or transformation (e.g., op1, op2, op3, etc.) such as filtering, merging, sorting, and mapping. The data processing job **200** includes a job profile **202**, for example, where op1, op2, and op3 represent the operation IDs of the data operations in stage 1 to stage 3.

[0072] As is described in more detail below, when a new data processing job is submitted to or received by the data processing cluster **102**, the data processing cluster **102** may analyze the job profile to determine whether or not the data processing cluster **102** already has intermediate cache data that can be reused for the present data processing job. Additionally, the cluster master (e.g., of the master node) maintains a table to track the locations of all the cached data. If the data processing cluster **102** identifies intermediate cache data that is already stored in the data processing cluster **102**, the data processing cluster **102** may simplify the data processing job, for example, by removing stages or operations from the job execution path **204** where the output data for the stages is already stored as intermediate cache data. The stage(s) following any removed stages then are provided with the stored intermediate cache data as input data, thereby reducing the overall resources utilized for the data processing job.

[0073] FIG. **3** illustrates a block diagram showing aspects of an example architecture of a data processing cluster according to some example embodiments of the present invention. Embodiments of the present invention utilize a cluster-based cache system with one cache master operating as part of a master node (e.g., the node **104***a*) and a plurality of cache workers operating as part of a plurality worker nodes (e.g., the nodes **104***b*-**104***d*) in the data processing cluster **102**.

[0074] The Cluster Master **302** and Task Workers **304***a*-**304***c*, of the master node (e.g., the node **104***a*) and the worker nodes (e.g., the nodes **104***b*-**104***d*), respectively, are responsible for executing the stages of a data processing job. For example, the Cluster Master **302** may schedule the data processing jobs and assign resources for different tasks or operations executed as part of a data processing job. The Task Workers **304***a*-**304***c* may execute the assigned tasks and update the resource availability of their corresponding worker node to Cluster Master **302**.

[0075] Additionally, the Data Master **306**, of the master node, and the Data Workers **308***a*-**308***c*, of the worker nodes, are responsible for storing data. The data processing cluster **102** may further include a distributed file system (DFS) **310** for distributing data storage across the memory of each of the worker nodes. The metadata of the file system may be managed in the Data Master **306** and the data itself may be stored by the Data Workers **308***a*-**308***c* (e.g., in the distributed file system **310**). When clients or a Task Worker requires data, they will request the Data Master to get the location of the data.

[0076] According to embodiments of the present invention, the master node (e.g., the node **104***a*) may include a cache master **312**, which communicates with other components in the master node and send writing/reading cache requests to the cache workers **314***a*-**314***c* of the worker nodes. The cache workers **314***a*-**314***c* update the resource capacities of their corresponding worker node to the Cache Master **312** and execute the writing/reading requests from the Cache Master **312**, for example, by storing intermediate cache data in a cluster-based cache **316** and reporting the location and nature of the stored intermediate cache data to the cache master **312**.

[0077] FIG. **4** illustrates further details of a data processing cluster according to some example embodiments of the present invention.

[0078] According to some embodiments, the cluster master **302** in the master node is responsible for scheduling applications and managing task workers. A data processing job is divided into multiple tasks or operations by the cluster master **302**, and these tasks or operations will be assigned to the task workers by the cluster master **302**, according to any suitable task assignment and distribution scheme.

[0079] According to some embodiments of the present invention, the cluster master **302** may further include an Application Client **402** and an Application Parser **404**.

[0080] According to some embodiments of the present invention, the Application Client **402** operates as a communication mechanism between the cache master **312** and the cluster master **302**. When a data processing job request is submitted to and/or received by the cluster master **302**, a job profile may be recorded in a job metadata table. The job metadata table may, for example, include three columns: a Job ID column, an Input Data column, and an Operation List column, as illustrated in Table 3 below.

[0081] The Job ID column stores unique IDs for each data processing job submitted to the data processing cluster. The Input Data column stores a list of input data blocks for each data processing job. Each input data block may be stored, for example, in the DFS of the data processing cluster **102** with a unique block ID. The operation list column includes a the list of operation IDs for a data processing job in each stage. Table 3 illustrates an example format of a data processing job profile, where j0, j1 and j2 are the job IDs of three different data processing jobs, block1 and block2 are the block IDs of the input data, and op1, op2, and op3 are the operation IDs.

TABLE 3

Job ID	Input Data	Operation List
j0	{block1}	{op1}
j1	{block1}	{op1, op2}
j2	{block1, block2}	{op1, op2, op3}

[0082] For a given data processing job, the Application Client **402** transmits the data processing job profile to the Cache Master **312** to determine whether or not there are intermediate cache data is already stored by the data processing cluster **102** that can be shared. If such intermediate cache data exists, the Cache Master **312** transmits the cache ID and its location (e.g., the data location among the nodes) to the application client (e.g., by way of an application service block).

[0083] In the case where intermediate cache data is available for sharing, the Application Parser **404** operates to simplify the operations of a job depending on the response from the cluster master of the cached intermediate data, by removing stages or operations from the job at hand. For example, if j0 is completed and its output data is cached with a unique cache ID cache0, the metadata of j1 can be simplified as illustrated in Table 4, below.

TABLE 4

Job ID	Input Data	Operation List
j1	{cache0}	{op2}

[0084] According to embodiments of the present invention, the Task Worker in each worker node executes the tasks assigned by the cluster master **302**. A Task Client block of the task worker may operate to maintain a heartbeat with Cluster Master **302**, to report the resource utility of the worker node, and to receive the commands from the Master node. Each task worker **304***a*-**304***c* may further include one or more Task, where each Task Executer executes one task at a time. The Task Executers request the data location from the data master and read/write data to the Data Worker.

[0085] Additionally, according to embodiments of the present invention, when a task is assigned to a task worker, the task worker identifies the type or nature of the input data. For example, if the input data is a block in the DFS, the task worker will request the data from Data Master and Data Executer. If the input data is a block of intermediate cache data, the task worker will send the cache ID to the Cache Master and request the location of the cache data. After the task worker receives the input data location, the task worker will read the data from the Cache Executer. The output data of a Task Executer will be written into cache or DFS according to the priority of the data and the capacity of the cache.

[0086] According to some embodiments of the present invention, the cache master may include four blocks, components or modules: Application Service, Cache Service, Task Service and Cache Scheduler. Application Service **406** communicates with the Cluster Master **302**. Such communication may be based on any suitable communication protocol, such as RPC protocol (Remote Procedure Call Protocol). When a data processing job submitted to or received by the application master, the data processing job profile may be reported to the cache master **312** by way of the application client **402** and the application service **406**.

[0087] A cache service block communicates with the Cache Worker in each worker node. Such communication may also be based on any suitable communication protocol, such as an RPC protocol. A heartbeat mechanism is maintained between the Cache Service and the Cache Worker periodically. Via a heartbeat, the Cache Worker may report to the Cache Master the resource capacities of the worker node including CPU, Memory, Cache and Storage utilities and responses whether a cache writing/deleting command is successfully executed. Additionally, the Cache Master may transmit writing/deleting cache messages to the cache worker by way of the heartbeat communication. If the Cache Master cannot receive the heartbeat from a Cache Worker for a period of time (e.g., a predetermined period of time, e.g., 10 minutes), the Cache Worker will be removed from the cache cluster.

[0088] A Cache Scheduler block **408** may execute cache management operations such as writing, reading, and deleting. For example, while caching a piece of data, the cache scheduler **408** may calculate the priority for the data (represented as a score) in the entire cache cluster and then select one or more appropriate Cache Workers to execute the writing commands to store the cache data. Additionally, when cached data is read from the data processing cluster **102** as part of a data processing job, the Cache Scheduler block **408** may operate to retrieve the cached data from the corresponding Cache Worker node.

[0089] Further, according to some embodiments, in order to improve the hit ratio of prestored or cached data, thereby improving the utility of the system, if the cache capacity of the entire data processing cluster is full, the Cache Scheduler **408** may cache data having a low priority (e.g., the lowest priority) in favor of storing or caching data having a higher priority.

[0090] The relative priority value (represented as a score) of cache data or potential cache data may be determined based on a probability of future accesses of the data. For example, more frequently accessed data may be assigned a higher priority value. Additionally, the relative priority value of cache data or potential cache data may be determined based on reconstruction costs for generating or obtaining the data in the future. For cached intermediate data, the reconstruction cost indicates the penalty if the data is completely evicted out of the cluster. Therefore, data with a higher reconstruction cost may be assigned a higher priority value. Further aspects of calculating and assigning the priority score for cached data or potential cache data is described in further detail in U.S. patent application Ser. No. 15/404,121, entitled "In-memory Shared-Intermediate Data Reuse Replacement and Caching," filed on Jan. 11, 2017, and U.S. patent application Ser. No. 15/404,100, entitled "A Duplicate In-Memory Shared-Intermediate Data Detection and Reuse Module in Spark Framework," filed on Jan. 11, 2017 the entirety of both of which is incorporated by reference herein.

[0091] In addition, the Cache Master **312** maintains two metadata tables: the operation table and the cache information table, described above. Once the intermediate data of a completed job is saved in the cache system successfully or any cached data is deleted, both tables are updated by the cache master **312**.

[0092] FIG. **5** illustrates further details of the workflow described above for a data processing job, according to example embodiments of the present invention. At **500**, a data source **110** transmits, and the cluster master **302** receives, at operation **500**, a data processing job, including input data, and the data processing job profile and one or more operations to be performed as part of the data processing job, as described above. When the cluster master **302** receives a submitted data processing job, the cluster master **302** may transmit, at operation **502**, the job profile as originally received (e.g., the enhanced job profile) to the cache master **312**, for determining whether or not the data processing job can be simplified. The cache master **312** may then initiate a search from among the stored intermediate cache data to determine whether or not any of the stored intermediate cache data corresponds to any operations of the data processing job (e.g., such that the stored intermediate cache data corresponds to a same input data and a same operation). If there is stored intermediate cache data in the cache system which can be reused by the job, at operation **504**, the cache master **312** may transmit the locations (e.g., within the cluster-based cache memory) of the stored intermediate cache data, the stored intermediate cache data itself, and/or a simplified job profile for which the redundant operations are substituted with the stored intermediate cache data. At operation **506**, the cluster master **302** may generate and/or assign tasks for the job profiled (e.g., the updated job profile) and assigned to task workers **304** to execute the operations of the job. If any aspect, operation, or intermediate stage of the data processing job relies on retrieving stored intermediate cache data from the cache memory, the individual task workers may, at operation **508**, transmit a request to the cache master **312** the cache location of the stored intermediate cache data. The task workers **304** may then, at operation **510**, receive from the cache master **312** the cache location within the cluster-based cache **316**. The task workers **304** may then request to, and receive from, respectively at operations **512** and **514**, the previously stored intermediate cache data from the corresponding cache worker **314**. Similarly, if any aspect, operation, or intermediate stage of the data processing job relies on retrieving data from a data worker **308**, the individual task workers may, at operation **516**, transmit a request to the data master **306** the data location of the data. The task workers **304** may then, at operation **518**, receive from the data master **306** the data location within the corresponding data worker **308**. The task workers **304** may then request to, and receive from, respectively at operations **520** and **522**, the data from the corresponding data worker **308**.

[0093] FIG. **6** illustrates aspects of a process or function for simplifying a data processing job in Cache Master, according to some example embodiments of the present invention. Referring to FIG. **6**, at operation **600**, input data and data processing job profile data is received (e.g., received by the cluster master) from an input source (e.g., data source **110**). Referring to the input data that may be received at operation **600**, $\overrightarrow{\boldsymbol{D}}$ is the list of input data blocks and $\vec{o}$ is the list of operations of a job. The number of operations is M, and the number of records (e.g., rows) in the operation table is $\overline{\overrightarrow{\boldsymbol{I}}}$, $\overline{\overrightarrow{\boldsymbol{P}}}$, $\overline{\overrightarrow{\boldsymbol{C}}}$ represent the original input data, the operation list, and the cached intermediate data of the operation table in Cache Master, respectively.

[0094] The system may then determine, at operation **602**, whether or not a counter i, initially set to be equal to the number of operations, is less than 1. If the counter i is less than 1, the system may return the list of input data blocks and the operations list, at operation **604**, after which the system will proceed with processing the returned list of input data blocks and the operations list as the simplified data processing job. Otherwise, at operation **606**, the system may determine whether or not a counter j, initialized at 1, is greater than the number of entries in the operation table of the operations list.

[0095] At operation **608**, the system may iterate through the table of cached intermediate data to determine whether or not any of the operations, and corresponding input data, from the input list of operations matches the cached intermediate data. In the process, the system may proceed at operations **610** and **612**, to increment the counter j and decrement the counter i, respectively, after evaluating each operation and the corresponding input data block to determine whether or not there is a corresponding stored intermediate cache data entry. If there is a corresponding stored intermediate cache data entry, the operation is removed from the job profile, and the cached intermediate data that would have been generated by the removed operation is retrieved in its place. Once each operation and corresponding input data has been evaluated to determine whether pre-stored intermediate cached data already exists, the system may return, at operations **616** or **618**, the simplified data processing job. At operation **616**, the simplified data processing job may include the set of intermediate cache data resulting from each of the operations being removed. At operation, **618**, the simplified processing job may include the set of intermediate cache data obtained during the simplification process, plus the remaining operations for which the data was not already stored as pre-stored intermediate cache data. Process 1, below, also illustrates the algorithm of FIG. **6** in pseudo code.

[0096] FIG. **7** illustrates aspects of a process or function for executing a data processing job according to embodiments of the present invention. At operation **700**, a data processing job, including input data D and an operations list O to be executed as part of the data processing job, is submitted to the system (e.g., the cluster master) from a data source. After a job profile is submitted to or received by the data processing cluster, the data processing cluster may generate a simplified job profile, at operation **702**, in which certain operations or stages may be removed as described above, for example, with respect to FIG. **6**. As illustrated at operation **702**, U is the input data of the simplified data processing job profile, and V is the simplified operations list. At operation **704**, the system determines whether or not the set or list of simplified operations is empty or zero, and if so, at operation **706**, the system may simply utilize or copy the corresponding cached intermediate data as the final output and the data processing job is completed. If, at operation **704**, the system determines the simplified operations list is not empty, system may determine, at operation **708**, to determine whether or not the input data U of the simplified job profile includes intermediate cache data. If so, at operation **710**, the system may proceed to retrieve the input data U from the cluster-based cache memory. Alternatively, if the input data U includes data stored in the distributed file system, the system may, at operation **712**, retrieve the input data U from the distributed file system. Then, at operation **714**, the system may proceed with executing the simplified operations V by utilizing the input data U. Process **2**, below, also illustrates the algorithm of FIG. **7** in pseudo code.

Process 1: simplifyJob

```
Procedure simplifyJob( )
Input Data: $\vec{D}$, $\vec{O}$ , $\vec{I}$ , $\vec{P}$ ,  $\vec{C}$
for i = M to 1 do
    for j = 1 to K do
        if $\vec{P}_j$ == $\{O_1,\ldots, O_i\}$ then
            if $\overline{D}$ == $\vec{I}_j$ then
                if i == M then
                    return ($\vec{C}_j$, { })
                else
                    return ($\vec{C}_j$, $\{O_i, \ldots, O_M\}$)
    return ($\vec{D}$, $\vec{O}$ )
```

Process 2: jobExecution( )

```
Procedure jobExecution( )
($\vec{U}$, $\vec{V}$ ) = simplifyJob( )
if | $\vec{V}$ | == 0 then
    copy $\vec{U}$ to DFS as the result
else
    if $\vec{U}$ is cache data then
        get $\vec{U}$ from cache system
    else
        get $\vec{U}$ from DFS
    Execute the operations $\vec{V}$ with input data $\vec{U}$
```

[0097] As illustrated in FIG. **8**, after a data processing job is completed (or during execution of a data processing job), before the data processing cluster deletes all intermediate data, the cache master may check the available cache capacity of the cache cluster, identify a priority of intermediate data generated as part of the data processing job, and store some or all of intermediate data in the cache memory of the worker nodes. All newly created intermediate data from all stages may be sorted by their priority score in descending order and the system may request to store or determine whether or not to store the intermediate data in the cache system one by one. For the intermediate data from one stage, if there are more than one worker nodes with enough space, the cache master may select a node to complete storage of the intermediate cache data, for example, based on input/output bandwidth or demand of each node, and/or available cache memory space in each node. For example, the data processing cluster may select a node based on a determination of which node will provide the most even balance of I/O bandwidth or demand and/or available cache memory space from among all of the nodes.

[0098] As illustrated in FIG. **9**, when there is not enough space to save the new intermediate data, the cache master may check the priority value (score) of each cached data and delete data with a lower priority than the new data, as illustrated in FIG. **8**. Referring to FIGS. **8** and **9**, $\vec{C}$: $\{\vec{C}_i | i \in [1, K]\}$, $\vec{W}$: $\{W_i | i \in [1, K]\}$ and $\vec{E}$: $\{E_i | i \in [1, K]\}$ represent the cached data list, the worker node ID of the data and their scores recorded in the cache information table of Cache Master. There are K rows in the table. $\vec{C}_i$ is the cache list in row i, w_i is the worker node ID the cache list is in, E_i is the score of $\vec{C}_i$, Θ be the size of the new intermediate data, ϕ is the score of the new intermediate data, and Γ is the I/O bandwidth for one writing operation.

[0099] When there is no space to save the new intermediate data, Cache Master will execute the algorithm illustrated in FIG. **9** to clear the cache system. The cache information table may be sorted by the score of the cached data, and the data processing cluster searches the cache list to identify cached data that has a larger size and a lower priority score than the new data, and for which the corresponding Cache Worker has enough writing bandwidth to write new data. If the data processing cluster identifies such previously stored intermediate cache data, it will be deleted from the corresponding worker node and the new intermediate cached data may be stored in the corresponding worker node. In addition, both the cache information table and the operation table in Cache Master will be updated. Process 3, below, also illustrates the algorithm of FIG. **9** in pseudo code.

Process 3: clearCache ( )

```
Procedure clearCache ( )
Input Data: $\Theta$, $\phi$, $\Gamma$, $\overline{\vec{C}}$, $\vec{W}$, $\vec{E}$
sort cache information table by score in ascending order
for i = 1 to K do
    if ( $E_i < \phi$) and ( sizeOf ($\overline{\vec{C}}_i$) >= $\Theta$ ) and
    (freeWritingBandwidthOf ($W_i$) > $\Gamma$ )
then
        delete $\overline{\vec{C}}_i$ in $W_i$
        update the cache information table and the operation table
        return {$W_i$}
return NULL
```

[0100] When there is sufficient space in more than one cache worker in the system, cache master may identify a cache worker to finish the writing. The load balance of both the available cache memory space and the I/O bandwidth may be considered. According to some embodiments of the present invention, the coefficient of variation (CV) may be utilized to express the deviation of storage and I/O bandwidth in each worker node. As CV is expressed as a percentage of the overall deviation, the deviation of storage and I/O bandwidth may be compared based on their CV values. The lower the value is, the better the load balance is presented in the cluster. As mentioned in the problem formulation, in the cluster with H worker nodes, S is the storage capacity and B is the maximum IO bandwidth for caching intermediate data in each worker node. And the set S': $\{s_1, s_2, \ldots, s_H\}$ B': $\{b_1, b_2, \ldots, b_H\}$ represent the remaining storage capacities and IO bandwidth of each worker node for caching. The coefficient of variations of the storage space and I/O bandwidth of the cache cluster can be expressed according to the following equations (1) and (2), below

$$CV(\text{space}) = \frac{\sqrt{\frac{1}{H}\sum_{i=1}^{H}(s_i - \bar{s})^2}}{\bar{s}} \tag{1}$$

$$CV(io) = \frac{\sqrt{\frac{1}{H}\sum_{i=1}^{H}(b_i - \overline{b})^2}}{\overline{b}} \tag{2}$$

[0101] The overall load balance of the cluster considers the load balance of both storage spaces and the disk I/O. An overall coefficient of variation for each node can be expressed according to equation 3, below. Some embodiments of the present invention may set a predetermined value t to adjust the weight of storage balance and disk IO balance, where ϵ is a number ranging from 0 and 1. According to some embodiments, the space balance may be assigned a greater weight, because the balance of storage space can improve the balance of disk I/O.

$$CV = \epsilon CV(\text{space}) + (1-\epsilon) CV(lo),\ \epsilon \in (0,1). \tag{3}$$

[0102] In addition, embodiments of the present invention may further consider the locality of the intermediate data. As the intermediate data are saved in the data workers, it may be more efficient to select a cache worker which is close to (e.g., geographically close to) the data worker with the intermediate data (e.g., in a same server rack). Embodiments of the present invention may categorize or group cache workers into three different groups: the local server (the one with the intermediate data), the servers in the same rack, and the servers in different racks. When selecting a node for storing intermediate cache data, embodiments of the present invention may first consider the local server, followed by servers in the same rack, and then servers in different racks.

[0103] Referring to FIG. **8**, Θ is the size of the intermediate data and Γ is the I/O bandwidth for one writing operation. Cache Master will first choose the candidate servers according to the process illustrated in FIG. **8**, and then select a cache server to save the intermediate data, according to the algorithm illustrated in FIG. **10**. If there is no candidate server, Cache Master will invoke the process illustrated in FIG. **9** to clear the cache system. Once the intermediate data is successfully stored in the cache system, the tables of cache information and operation will be updated in the Cache Master.

[0104] Referring to FIG. **8**, at operation **800**, in selecting a candidate server, the system may receive input data Θ, Γ, N, S', B', where Θ is the size of the intermediate data, Γ is the IO bandwidth for one writing operation, and N is the list of worker nodes. Q**1** is the set of the candidate server which is also the local server (e.g., the server with the intermediate data), Q**2** is the set of candidate servers in the same rack, and Q**3** is the set of candidate servers in a different rack. Beginning at operation **802**, for i=1 to |N| (the length of N, which is the number of worker servers or nodes in the cluster), the system may determine, at operation **806**, whether or not $\Theta + S_1$ is less than S or whether $\Gamma + b_1 < B$. If so, the system may increment the counter i at operation **804**. If not, the system may determine, at operation **808**, whether or not n_i is the local server. If so, the system may add n_i to Q1 at operation **810**, and return Q1 at **812**. Alternatively, if n_i is not in the local server, the system may determine, at operation **814**, whether or not n_i is in the same rack. If so, the system may add n_i to Q**2**, at operation **816**. If n_i is not in the same rack, the system may add n_i to Q**3**, at operation **818**. Returning to operation **802**, if i is greater than N, the system may determine whether or not |Q**2**| is greater than 0. If so, the system may return Q**2**, at operation **822**. If not, the system may return Q**3**, at operation **824**.

[0105] Referring to FIG. **9**, at operation **900**, in clearing cache from the cluster-based cache memory, the system may receive, as input, Θ, ϕ, Γ, $\overline{\vec{C}}$, $\vec{W}$, $\vec{E}$ where Θ is the size of the new intermediate data to be stored in the cache system, ϕ is the score or priority of the new intermediate data, Γ is the bandwidth that would be utilized to write the new intermediate data, and $\overline{\vec{C}}$, $\vec{W}$, $\vec{E}$ are the values from the cache information table. At operation **902**, the system may sort the cache information table according to the priority or score (e.g, in ascending order). Then, at operation **904**, starting at i=1, the system may determine whether or not i is less than or equal to K. If not, the system may proceed at operation **908**, to increment i. If so, the system may determine, at operation **906**, whether or not $E_i<\phi$ and the size of $\vec{C}_i$ is greater than or equal to zero and the free writing badwidth of W_i is greater than Γ. If so, the system may delete C_i in W_i, at operation **910**. Then, the system may proceed, at operation **912**, to update the tables in the cache master according to the deleted intermediate cache data. Continuing from operation **904**, once i is greater than K, the system may return W_i.

[0106] Referring to FIG. **10**, at operation **1000**, in selecting a cache worker, the system may receive, as input, Θ, Γ, N, S', B'. Additionally, T be initialized as an empty set, where T is a set used to save the list of candidate worker nodes returned from the algorithm for selecting candidates. Additionally, for each worker node, there is a tuple to save the node index and it is matched (e.g., Map <node index, CV>R={}). At operation **1002**, the system may proceed to select a candidate, for example, as discussed above with respect to FIG. **8**. Then, at operation, **1004**, the system may determine whether or not |T|==0. If |T|==0, the system may proceed at operation **1006** to clear the cache, as discussed above, for example, with respect to FIG. **9**. Next, at operation **1008**, the system may again determine whether or not |T|==0. If, at operation **1008**, |T|==0, the system may return NULL at operation **1010**. Alternatively, if, at operations **1004** or **1008**, |T|!==0, the system may proceed, at operation **1012**, to determine whether or not the length of T is 1 (e.g., |T|==1). If, at operation **1012**, T==1, the system may return the first worker node installed in the list T (e.g., T.get(0)). If, however, at operation **1012**, |T|!==1, the system may set a counter i to a value of 1, at operation **1016**. Then, at operation **1018**, the system may determine whether or not i<|T|. If so, the system may proceed to operation **1024** to set R to the node ID with the minimum CV in R. Alternative, if, at operation **1018**, i is not <|T|, they system may proceed, at operation **1020**, to set j=node ID of M[i], $b_j=b_j+\Gamma$, $s_j=s_j+\Theta$,

$$CV=\varepsilon\frac{\sqrt{\frac{1}{H}\sum_{i=1}^{H}(s_i-\bar{s})^2}}{\bar{s}}+(1-\varepsilon)\frac{\sqrt{\frac{1}{H}\sum_{i=1}^{H}(b_i-\bar{b})^2}}{\bar{b}},$$

R.add(j, CV), $b_j=b_j-\Gamma$, and $s_j=s_j-\Theta$. Then, at operation **1022**, the system may increment i by 1 and then continue looping through operations **1018**, **1020**, and **1022**, until i is not less than |T|.

[0107] The processes of FIGS. **8** and **10** are also illustrated in pseudo code in processes 4 and 5 below, respectively.

Process 4: selectCandidates ( )

-continued

```
Procedure selectCandidates ( )
Input Data: $\Theta$, $\Gamma$ , N, S', B'
Initial: $Q_1$ = { }, $Q_2$ = { }, $Q_3$ = { }
for i = 1 to |N| do
    if {$\Theta + s_1 < S$) or ($\Gamma + b_1 < B$) then
        continue
    else
        if $n_i$ is the local server then
            $Q_1$.add ( $n_i$ )
            return $Q_1$
        else if $n_i$ is in the same rack then
            $Q_2$.add ( $n_i$ )
        else
            $Q_3$.add ( $n_i$ )
for if | $Q_2$| > 0 then
    return $Q_2$
return $Q_3$
```

Process 5: selectCacheWorker( )

```
Procedure selectCacheWorker( )
Input Data: $\Theta$, $\Gamma$ , N, S', B'
Initial: T = { }, Map<node index, CV> R = { }
T = selectCandidates ( )
if |T| == 0 then
    T = clearCache( )
if |T| == 0 then
    return NULL
else if |T| == 1 then
    return T.get(0)
else
    for i = 1 to |T| do
        j = node ID of M[i]
        $b_j = b_j + \Gamma$
        $s_j = s_j + \Theta$
        $CV = \varepsilon\frac{\sqrt{\frac{1}{H}\sum_{i=1}^{H}(s_i-\bar{s})^2}}{\bar{s}}+(1-\varepsilon)\frac{\sqrt{\frac{1}{H}\sum_{i=1}^{H}(b_i-\bar{b})^2}}{\bar{b}}$
        R.add(j, CV)
        $b_j = b_j - \Gamma$
        $s_j = s_j - \Theta$
    r = node ID with min(CV) in R
    return $n_r$.
```

[0108] Thus, according to one or more example embodiments of the present invention, a data processing cluster may facilitate improved use of resources in the data processing cluster.

[0109] Although this invention has been described in certain specific embodiments, those skilled in the art will have no difficulty devising variations to the described embodiment, which in no way depart from the scope and spirit of the present invention. Furthermore, to those skilled in the various arts, the invention itself herein will suggest solutions to other tasks and adaptations for other applications. It is the Applicant's intention to cover by claims all such uses of the invention and those changes and modifications which could be made to the embodiments of the invention herein chosen for the purpose of disclosure without departing from the spirit and scope of the invention. Thus, the present embodiments of the invention should be considered in all respects as illustrative and not restrictive, the scope of the invention to be indicated by the appended claims and their equivalents rather than the foregoing description.

What is claimed is:

1. A system comprising:

a processor; and

a memory coupled to the processor, wherein the memory stores instructions that, when executed by the processor, cause the processor to:

receive, from a data source, a processing profile comprising input data blocks and a plurality of operations for executing using the input data blocks;

determine whether or not a block of stored intermediate cache data corresponds to an operation from among of the plurality of operations;

in response to determining the block of stored intermediate cache data corresponds to the operation from among the plurality of operations, generate a simplified processing profile based on the block of stored intermediate cache data;

execute the simplified processing profile by generating a new output data after each operation of the simplified processing profile;

store the new output data from at least one operation as intermediate cache data; and

transmit the output data from a final operation in the simplified processing profile to the data source for display thereby.

2. The system of claim **1**, wherein the instructions further cause the processor to, in response to determining the block of stored intermediate cache data corresponds to the operation from among the plurality of operations, identify a location of the stored intermediate cache data among a plurality of worker nodes.

3. The system of claim **1**, wherein generating the simplified processing profile comprises removing the operation corresponding to the block of stored intermediate cache data.

4. The system of claim **3**, wherein the simplified processing profile comprises a subset of the plurality of operations of the processing profile without the operation corresponding to the block of stored intermediate cache data.

5. The system of claim **1**, wherein the instructions further cause the processor to:

identify a candidate worker node from among a plurality of worker nodes for storing the new output data according to a load balance calculation of at least one of storage space of each of the worker nodes and input/output bandwidth of each of the worker nodes; and

store the new output data at the identified candidate worker node.

6. The system of claim **1**, wherein the instructions further cause the processor to:

identify whether or not there is sufficient space among a plurality of worker nodes to store the new output data; and

in response to determining there is not sufficient space among the plurality of worker nodes, clear a block of pre-stored intermediate cache data having a lower priority level than the new output data.

7. A method comprising:

receiving, by a processor, from a data source, a processing profile comprising input data blocks and a plurality of operations for executing using the input data blocks;

determining, by the processor, whether or not a block of stored intermediate cache data corresponds to an operation from among of the plurality of operations;

in response to determining the block of stored intermediate cache data corresponds to the operation from among the plurality of operations, removing, by the processor, the operation from the processing profile to generate a simplified processing profile;

executing, by the processor, the simplified processing profile by generating a new output data after each operation of the simplified processing profile;

storing, by the processor, the new output data from at least one operation as intermediate cache data; and

transmitting, by the processor, the output data from a final operation in the simplified processing profile to the data source for display thereby.

8. The method of claim **7**, further comprising, in response to determining the block of stored intermediate cache data corresponds to the operation from among the plurality of operations, identifying, by the processor, a location of the stored intermediate cache data among a plurality of worker nodes.

9. The method of claim **7**, wherein generating the simplified processing profile comprises removing, by the processor, the operation corresponding to the block of stored intermediate cache data.

10. The method of claim **9**, wherein the simplified processing profile comprises a subset of the plurality of operations of the processing profile without the operation corresponding to the block of stored intermediate cache data.

11. The method of claim **7**, further comprising identifying, by the processor, a candidate worker node from among a plurality of worker nodes for storing the new output data according to a load balance calculation of at least one of storage space of each of the worker nodes and input/output bandwidth of each of the worker nodes.

12. The method of claim **11**, further comprising storing, by the processor, the new output data at the identified candidate worker node.

13. The method of claim **7**, further comprising:

identifying, by the processor, whether or not there is sufficient space among a plurality of worker nodes to store the new output data; and

in response to determining there is not sufficient space among the plurality of worker nodes, clearing, by the processor, a block of pre-stored intermediate cache data having a lower priority level than the new output data.

14. A method comprising:

receiving, by a processor, from a data source, a processing profile comprising input data blocks and a plurality of operations for executing using the input data blocks;

executing, by the processor, one or more of the operations of the processing profile to generate a new output data after each of the executed one or more operations;

storing, by the processor, the new output data from at least one of the one or more operations as intermediate cache data; and

transmitting, by the processor, the new output data from a final operation from among the one or more operations to the data source for display thereby.

15. The method of claim **14**, further comprising:

determining, by the processor, whether or not a block of stored intermediate cache data corresponds to an operation from among of the plurality of operations; and

in response to determining the block of stored intermediate cache data corresponds to the operation from among the plurality of operations, removing, by the processor, the operation from the processing profile to generate a simplified processing profile.

16. The method of claim **15**, further comprising, in response to determining the block of stored intermediate cache data corresponds to the operation from among the plurality of operations, identifying, by the processor, a location of the stored intermediate cache data among a plurality of worker nodes.

17. The method of claim **15**, wherein the simplified processing profile comprises a subset of the plurality of operations of the processing profile without the operation corresponding to the block of stored intermediate cache data, and the method further comprises executing, by the processor, each of the plurality of operations among the subset.

18. The method of claim **14**, further comprising identifying, by the processor, a candidate worker node from among a plurality of worker nodes for storing the new output data according to a load balance calculation of at least one of storage space of each of the worker nodes and input/output bandwidth of each of the worker nodes.

19. The method of claim **18**, further comprising storing, by the processor, the new output data at the identified candidate worker node.

20. The method of claim **14**, further comprising:

identifying, by the processor, whether or not there is sufficient space among a plurality of worker nodes to store the new output data; and

in response to determining there is not sufficient space among the plurality of worker nodes, clearing, by the processor, a block of pre-stored intermediate cache data having a lower priority level than the new output data.

* * * * *