United States Patent Application Publication 20250265246
Kind Code A1
Publication Date August 21, 2025
Inventor(s) Falcao; Eric Manuel et al.

TEMPORAL OPTIMIZATION OF DATA OPERATIONS USING DISTRIBUTED SEARCH AND SERVER MANAGEMENT

Abstract

The disclosure describes temporal optimization of data operations using distributed search and server management, including configuring one or more host groups, determining one or more stripes associated with one or more shards distributed among the one or more host groups, receiving a query to retrieve data, evaluating the query to identify a time characteristic associated with the data, identifying a location from which to retrieve the data, and rewriting the query to run on at least one of the one or more host groups at the location using a distributed search platform, the another query being targeted at a host group associated with the class.

Inventors: **Falcao; Eric Manuel (Austin, TX), Hoerner; Brett (Austin, TX), Swain; Matthew (Austin, TX), McElwee; Adam (Austin, TX)**

Applicant: **Spredfast, Inc.** (Austin, TX)

Family ID: **1000008578456**

Assignee: **Spredfast, Inc. (Austin, TX)**

Appl. No.: **19/063152**

Filed: **February 25, 2025**

Related U.S. Application Data

parent US continuation 18200488 20230522 parent-grant-document US 12235842 child US 19063152
parent US continuation 17332391 20210527 parent-grant-document US 11657053 child US 18200488

Publication Classification

Int. Cl.: **G06F16/2453** (20190101); **G06F16/2458** (20190101); **G06F16/27** (20190101); **H04L67/10** (20220101)

U.S. Cl.:

CPC **G06F16/24542** (20190101); **G06F16/2471** (20190101); **G06F16/27** (20190101); H04L67/10 (20130101)

Background/Summary

CROSS-REFERENCE TO RELATED APPLICATIONS [0001] This nonprovisional patent application is a continuation application of copending U.S. patent application Ser. No. 18/200,488, filed May 22, 2023 and entitled, "TEMPORAL OPTIMIZATION OF DATA OPERATIONS USING DISTRIBUTED SEARCH AND SERVER MANAGEMENT," U.S. patent application Ser. No. 18/200,488 is a continuation application of U.S. patent application Ser. No. 17/332,391, filed May 27, 2021, now U.S. Pat. No. 11,657,053 and entitled, "TEMPORAL OPTIMIZATION OF DATA OPERATIONS USING DISTRIBUTED SEARCH AND SERVER MANAGEMENT," U.S. patent application Ser. No. 17/332,391 is a continuation application of U.S. patent application Ser. No. 15/877,381, filed Jan. 22, 2018, now U.S. Pat. No. 11,061,900 and entitled, "TEMPORAL OPTIMIZATION OF DATA OPERATIONS USING DISTRIBUTED SEARCH AND SERVER MANAGEMENT," all of which are herein incorporated by reference in their entirety for all purposes.

FIELD
[0002] The present invention relates generally to computer and data science, computer software and hardware, computer and server memory, distributed server configuration, and data storage and retrieval management. More specifically, techniques for temporal optimization of data operations using distributed search and server management are described.
BACKGROUND
[0003] As various computer programs, applications, platforms, and other users and consumers of data increase, there are also a rising number of problems associated with managing large amounts of data. Processing, storage, and retrieval of large quantities of data, including that generated from social media and social networks, are areas of innovation that are struggling to keep pace with the rising demand for increasingly complex and sophisticated data storage. Searching and retrieving data stored in large amounts across distributed data networks that use extensive physical, virtual, and logical resources is becoming increasingly difficult to deploy and manage and incurs significant expense to users, consumers, and customers of data. In other words, conventional techniques for managing large amounts of data address the inherent issue of scalability by providing expensive solutions that typically involve adding more resources instead of managing existing resources for greater efficiency, lower latency, and higher reliability; these techniques are technologically limited and expensive in terms of time, labor, and financial cost. With data sources such as online commerce, social media, social networks, enterprises (i.e., large corporate, governmental, academic, institutional, military, financial, medical/healthcare, or other types of private data networks) generating increasingly large quantities of data, conventional techniques for processing, storing, and managing are failing to provide solutions that are able to support these data needs. Further, finding specific items within these large quantities of data is also increasingly difficult. Still further, there are individuals, entities, and organizations that wish to commercialize data, but due to the large quantities, are finding it increasingly difficult to communicate, market, sell, promote, or otherwise generate targeted messages to intended users. Conventional data

management techniques store large amounts of data in a manner that do not facilitate rapid and accurate searching and retrieval. Conventional techniques typically rely upon increasing the amount and types of data storage servers (i.e., adding physical, virtual, or logical processing or storage resources) and, when combined with conventional partitioning techniques such as striping, are problematic because these techniques do not scale. Often conventional techniques are not only slow and inefficient when searching and retrieving data from databases, but these also typically result in generating server indices that are also massive in scale and difficult to search for specific data. More importantly, these conventional techniques are prohibitively expensive as data storage servers tend to be expensive, complex, and difficult to deploy, particularly for smaller enterprises and businesses with substantially lesser technology budgets and financing options. For hosted services such as computing cloud-based storage services, some of the complexities of deployment and management are lessened, but the expense of using these services continues to remain high as different classes of servers with different levels of performance also carry different prices. Faster performance typically requires higher cost, which is problematic as computer and data science continues to improve.
[0004] Thus, what is needed is a solution for managing, storing, and retrieving data without the limitations of conventional techniques.

Description

BRIEF DESCRIPTION OF THE DRAWINGS
[0005] Various embodiments or examples ("examples") of the invention are disclosed in the following detailed description and the accompanying drawings:
[0006] FIG. **1** illustrates an exemplary computing environment for temporal optimization of data operations using distributed server and shard management;
[0007] FIG. **2** illustrates an exemplary application architecture for temporal optimization of data operations using distributed server and shard management;
[0008] FIG. **3** illustrates an exemplary host group and server cluster configuration for temporal optimization of data operations using distributed server and shard management;
[0009] FIG. **4** illustrates an exemplary manifest file structure for temporal optimization of data operations using distributed server and shard management;
[0010] FIG. **5**A illustrates an exemplary data model and flow diagram for temporal optimization of data operations using distributed server and shard management;
[0011] FIG. **5**B illustrates an alternative exemplary data model and flow diagram for temporal optimization of data operations using distributed server and shard management;
[0012] FIG. **5**C illustrates a further exemplary data model and flow diagram for temporal optimization of data operations using distributed server and shard management;
[0013] FIG. **6** illustrates an exemplary process for temporal optimization of data operations using distributed server and shard management;
[0014] FIG. **7**A illustrates another exemplary process for temporal optimization of data operations using distributed server and shard management;
[0015] FIG. **7**B illustrates an exemplary process for host group and server configuration for temporal optimization of data operations using distributed server and shard management;
[0016] FIG. **7**C illustrates an exemplary process for reconciliation operations for temporal optimization of data operations using distributed server and shard management;
[0017] FIG. **8**A illustrates an alternative exemplary process for temporal optimization of data operations using distributed server and shard management;
[0018] FIG. **8**B illustrates an exemplary process for temporally optimized data retrieval using distributed server and shard management;

[0019] FIG. **8**C illustrates an exemplary process for temporally optimized data queries using distributed server and shard management;
[0020] FIG. **8**D illustrates an exemplary process for retrieving temporally optimized data using distributed server and shard management;
[0021] FIG. **8**E illustrates an alternative exemplary process for retrieving temporally optimized data using distributed server and shard management; and
[0022] FIG. **9** illustrates an exemplary computing system suitable for temporal optimization of data operations using distributed server and shard management.

DETAILED DESCRIPTION

[0023] Various embodiments or examples may be implemented in numerous ways, including as a system, a process, an apparatus, a user interface, or a series of program instructions on a computer readable medium such as a computer readable storage medium or a computer network where the program instructions are sent over optical, electronic, or wireless communication links. In general, operations of disclosed processes may be performed in an arbitrary order, unless otherwise provided in the claims.
[0024] A detailed description of one or more examples is provided below along with accompanying figures. The detailed description is provided in connection with such examples, but is not limited to any particular example. The scope is limited only by the claims and numerous alternatives, modifications, and equivalents are encompassed. Numerous specific details are set forth in the following description in order to provide a thorough understanding. These details are provided for the purpose of example and the described techniques may be practiced according to the claims without some or all of these specific details. For clarity, technical material that is known in the technical fields related to the examples has not been described in detail to avoid unnecessarily obscuring the description.
[0025] FIG. **1** illustrates an exemplary computing system for temporal optimization of data operations using distributed server and shard management. Here, system **100** includes application **102**, logic module **104**, cluster manifest file manager **106**, resource manager **108**, indexer **110**, search platform **112**, query manager **114**, application programming interface (hereafter "API") **115**, data networks **116-118**, index database **120**, database **122**, social database **124**, manifest files database **126**, metadata database **128**, client **130**, mobile clients **132-134**, and computer **136**. In some examples, "system" may refer to or include the description of a computer network system or topology associated with a map, network, layout, environment, or the like of computing resources that are used to implement a feature, function, process, element, component, part, or multiple instances thereof, without any particular limitation as to the type, configuration, programming or formatting language, service, class, resource, specification, protocol, or other attributes thereof. As used herein, "application" may also be used interchangeably or synonymously with, or refer to a computer program, software, program, firmware, or any other term that may be used to describe, reference, or refer to a logical set of instructions that, when executed, performs a function or set of functions within a computing system or machine, regardless of whether physical, logical, or virtual and without restriction or limitation to any particular implementation, design, configuration, instance, or state. Also as used below, "platform" may refer to any type of computer hardware (hereafter "hardware") and/or software environment using one or more local, remote, distributed, networked, or computing cloud (hereafter "cloud")-based computing resources (e.g., computers, clients, servers, tablets, notebooks, smart phones, cell phones, mobile computing platforms or tablets, and the like) to execute an application, such as those described above, without restriction or limitation to any particular implementation, design, configuration, instance, or state. Distributed resources such as cloud networks (also referred to interchangeably as "computing clouds," "storage clouds," "cloud networks," or, simply, "clouds," without restriction or limitation to any particular implementation, design, configuration, instance, or state) may be used for processing and/or storage computing resources, without restriction or limitation to any particular implementation, design,

configuration, instance, or state of a device or machine, physical, virtual, logical, or otherwise.
[0026] Referring back to FIG. **1**, in some examples, data may be processed from various sources, including, but not limited to index database **120**, database **122**, social database **124**, manifest files database **126**, and metadata database **128**. Data may be retrieved from local, remote, distributed, networked, or cloud-based databases or data sources such as data streams or sources of data that are generated from various types of applications or clients. As used herein, data may be stored in various types of data structures including, but not limited to databases, data repositories, data warehouses, data stores, or other data structures configured to store data in various computer programming languages and formats in accordance with various types of data schemas such as SQL, MySQL, NoSQL, DynamoDB™ from Amazon® Web Services, Inc. of Seattle, Washington, FMP, Oracle®, relational or unstructured, or others, without limitation or restriction to any particular instance or implementation. Further, references to databases, data structures, or any type of data storage facility may include any embodiment as a local, remote, distributed, networked, cloud-based, or combined implementation thereof. For example, social networking applications being used on different types of devices (e.g., client **130**, mobile clients **132-134**, and computer **136**) may be generating data in different forms, formats, and data storage schema on different types of data storage devices and databases that are configured to store data generated for text messaging, image, audio or video streams, blog (i.e., World Wide Web-posted log) or vlog (i.e., video log) entries, instant messages, disappearing messages (e.g. snaps using Snapchat™, simple message system (i.e., SMS, IRC, and the like) messages, and the like, without limitation or restriction. Data may be generated from “tweets” or messages from applications such as Twitter® of San Francisco, California, “snaps” using the Snapchat® application such as those developed by Snap® of Venice, California, or “messenger” posts using applications such as Facebook® of Menlo Park, California. In some examples, data may be formatted using simple messaging data communication and transmission protocols such as Internet Relay Chat (IRC), SMS, instant messaging (IM), or others, without limitation. In other examples, client **130**, mobile clients **132-134**, and computer **136** may generate data in other forms and formats using programming and formatting languages intended to transmit, transfer, or transport data at different levels of an application architecture (as described in greater detail below). For example, Java®, JavaScript®, Python™, XML, and other data formats and programs may be used in conjunction with the techniques described herein by application **102**, which may be implemented to provide messaging functionality at the application layer of a seven-layer “stack” application architecture following a standard such as the Open Systems Interconnect (OSI) model.
[0027] As shown in FIG. **1**, in some examples, application **102** may be an application configured to receive data from various sources (e.g., index database **120**, database **122**, social database **124**, manifest files database **126**, metadata database **128**, client **130**, mobile clients **132-134**, computer **136**, or others), which may include stored data (e.g., index database **120**, database **122**, social database **124**, manifest files database **126**, metadata database **128**, or others) or data generated in real-time or substantially real-time from other applications, clients, or systems. In some examples, data may be sent or received over data networks **116-118**, which may be local area networks (LAN), wide area networks (WAN), municipal area networks (MAN), wireless local area networks (WLAN), cloud computing or cloud data networks that use disparate computing and storage resources managed by other applications or resources to provide storage and processing capabilities. As described herein, data may be sent using any type of data network that is configured to send and receive data in any type of digital format, regardless of program or formatting language, without limitation.
[0028] As shown here, data may be queried from, retrieved, or stored to one or more of index database **120**, database **122**, social database **124**, manifest files database **126**, metadata database **128**, or other types, schema, or structures, all, some, or none of which may be configured to store data in data structures such as a database, data store, data repository, data facility, data warehouse,

or the like. Data may also be generated and received from or sent to one or more of client **130**, mobile clients **132-134**, computer **136**, or other clients (not shown) that consume, use, or generate data.

[0029] As shown, client **130**, mobile clients **132-134**, and computer **136** are representative of types of clients and devices that may send data to or receive data from application **102** over data network **118**. Data may be in any type of output format generated by an application (e.g., Twitter®, Snapchat®, Messenger®, instant messaging, email, text messaging, IRC, HTML, XML, Java®, or others, without limitation), transmitted over data networks **118** to application **102**. Likewise, data may be stored in any type of format and, in some examples, classified as various types of data formats in one or more of index database **120**, database **122**, social database **124**, manifest files database **126**, metadata database **128**, or other databases (not shown). For example, messaging data may be transmitted from a mobile tablet (e.g., mobile clients **132-134**) over data network **118** to application **102**. Application **102** may be configured to receive or send data directly or indirectly to/from any of modules **104-115** over data networks **116-118**. In some examples, data received by or sent from application **102** may be transmitted over application programming interface **115**, which may be configured to receive, convert, and transfer (i.e., send) data in any type of data format from clients **130-136** for processing by application **102** (or any of modules **104-114**) before being stored to one or more of index database **120**, database **122**, social database **124**, manifest files database **126**, metadata database **128**, or other databases (not shown). Similarly, data may be queried and retrieved from one or more of index database **120**, database **122**, social database **124**, manifest files database **126**, metadata database **128**, or other databases (not shown) using, for example, API **115**. Further, data may also be processed by one or more of modules **104-114** and transferred using API **115** over data network **116** (which may also be implemented using a single or multiple data networks) to one or more of index database **120**, database **122**, social database **124**, manifest files database **126**, metadata database **128**, or other databases (not shown) to be stored at a location that may be indicated in a server or database index (hereafter "index"). Application **102** may also be configured to transfer data to/from one or more of clients **130-136** over API **115** to one or more of logic module **104**, cluster manifest file manager **106**, resource manager **108**, indexer **110**, search platform **112**, or query manager **114** to perform processing or other functions on the data before transmitting data back to clients **130-136** or storing processed data (i.e., "processed data" refers to resultant data generated by one or more of logic module **104**, cluster manifest file manager **106**, resource manager **108**, indexer **110**, search platform **112**, query manager **114**, or application **102**) on one or more of index database **120**, database **122**, social database **124**, manifest files database **126**, metadata database **128**. For example, a stream of "social data" (e.g., data generated by a social media application or network such as those described above or others, without limitation) may be received as input to application **102** over data network **118**. Social data, in some examples, may refer to data generated from any type of application, system, device, platform, program, or other set of computer instructions that are intended for use in enabling data transfer and communication between endpoints used by and for social purposes such as the creation, management, or participation in a computing or online community. As used herein, "online" may refer to any type of computing environment in which endpoints (e.g., clients **130-136**) are configured to transfer data between each other, either directly or indirectly over one or more data networks (e.g., data networks **116-118**). As an example, social data may be generated from social media (e.g., content that is transmitted digitally to/from various types of devices such as posting digital still images, video and audio files, text-based content such as editorials, news feeds, entertainment-related information, and others, without limitation or restriction), social networks (e.g., data networks that are purposes-configured and deployed to transfer social data), or applications that can be used for purposes such as social media, among others. Further, social data is an example of data that application **102** may be executed upon, but is neither a limitation nor a restriction on the types of data that may be processed by application **102**.

[0030] Referring back to FIG. **1**, social data may be stored in social database **124** and transmitted from or sent to one or more of clients **130-136** using application **102**. In other examples, data (including social data and other types of data) may be transferred between clients **130-136** and data storage facilities (e.g., index database **120**, database **122**, social database **124**, manifest files database **126**, metadata database **128**, or others as described herein) directly (not shown) or indirectly through, for example, application **102** using data networks **116-118** or other data networks (not shown). Regardless, data received by application **102** may be processed by one or more of logic module **104**, cluster manifest file manager **106**, resource manager **108**, indexer **110**, search platform **112**, and/or query manager **114** in order to temporally optimize the storage and retrieval of data generated from, for example, clients **130-136**. "Temporal optimization" techniques are described in greater detail below.

[0031] Here, application **102** is configured with logic module **104**, cluster manifest file manager **106**, resource manager **108**, indexer **110**, search platform **112**, and query manager **114**. In some examples, application **102** may be configured differently than is shown with more, fewer, or different modules apart from those shown and described (i.e., logic module **104**, cluster manifest file manager **106**, resource manager **108**, indexer **110**, search platform **112**, and query manager **114**). As shown, logic module **104**, cluster manifest file manager **106**, resource manager **108**, indexer **110**, search platform **112**, and query manager **114** may be implemented as software, hardware, firmware, or a combination thereof, without limitation or restriction to any computer programming or formatting language, device, type, or configuration. However, here, one or more of logic module **104**, cluster manifest file manager **106**, resource manager **108**, indexer **110**, search platform **112**, and query manager **114** may be implemented as software modules that are configured to communicate with each other module over a local or distributed data network. In some examples, application **102** and logic module **104**, cluster manifest file manager **106**, resource manager **108**, indexer **110**, search platform **112**, and query manager **114** may be implemented as part of a software platform where each module may be encoded using, for example, Java®, Python™, JavaScript (also referred to as JS), binary encoding, machine assembly, COBOL, HTML, xHTML, XML, or any other type of computer programming and formatting languages.

[0032] Here, logic module **104** is configured to provide processes and algorithms for various features and functionality, including providing control functionality for cluster manifest file manager **106**, resource manager **108**, indexer **110**, search platform **112**, and query manager **114**. In some examples, logic module **104** may be a set of rules that are manually input to provide parameters, thresholds, and other quantitative restrictions or comparative statements to help determine actions to be taken, which of cluster manifest file manager **106**, resource manager **108**, indexer **110**, search platform **112**, or query manager **114** to invoke, service calls to make (e.g., over API **115**), or any other functions that require decisions or a determinant action to be taken. In other examples, logic module **104** may be implemented not as a rules-based engine or module, but instead as a heuristic, semantic, or context evaluation set of algorithms that evaluate incoming data from, for example, clients **130-136** or elsewhere (e.g., index database **120**, database **122**, social database **124**, manifest files database **126**, metadata database **128**, or other data storage or processing resources over data networks such as data networks **116-118**) in order to parse, evaluate, analyze, or perform other functions provided by application **102**. As shown, application **102** is configured with several modules that provide different platform functions for temporally managing data from various sources in order to optimize or improve the efficiency of searching and retrieval in response to queries as well as reducing latency in returning data in response to a query, accurately and efficiently. For example, if social data (not shown) is generated from an input or "stream" of social data coming from one or more of clients **130-136**, it may be received at API **115** and further processed by resource manager **108**, which may be configured to manage various types of storage resources (e.g., local, distributed, networked, cloud-based, and the like) to determine the technical configuration and location of the social data to be stored (e.g., in social database **124**).

Resource manager **108**, in some examples, may also exchange data with one or more other modules (e.g., cluster manifest file manager **106**, indexer **110**, search platform **112**, query manager **114**, or API **115**) to perform other functions. As shown, resource manager **108**, in some examples, can be used cooperatively with cluster manifest file manager **106** and indexer **110** in order to determine a location that has been temporally-optimized (as described in further detail below) for subsequent search and retrieval in response to query requests (hereafter "query" or "request," which may be used substantially synonymously or interchangeably with each other to indicate a request for data in response to a request formatted in accordance with a query language (without limitation or restriction to any particular language) from a given location and resource (e.g., server or group of logically or physically coupled (directly, indirectly, or remotely) servers).

[0033] As shown, cluster manifest file manager **106** may be configured to receive instructions from logic module **104** and/or resource manager **108** to configure one or more databases intended to store processed data generated from data received by application **102** from, for example, clients **130-136**. Databases may include index database **120**, database **122**, social database **124**, manifest files database **126**, metadata database **128**, and others, without limitation or restriction to those shown and described. In some examples, storage and/or processing resources such as index database **120**, database **122**, social database **124**, manifest files database **126**, metadata database **128**, and others may be configured by cluster manifest file manager **106** to create a managed system of resources that can store data processed by application **102** by optimizing the storage and retrieval of data (i.e., increasing the speed of execution of program instructions to execute a query, delete, add, or write operation in one or more of index database **120**, database **122**, social database **124**, manifest files database **126**, metadata database **128**, or other database(s), without limitation). As described in further detail below, cluster manifest file manager **106** is configured to provide program instructions from application **102** to each of index database **120**, database **122**, social database **124**, manifest files database **126**, metadata database **128**, or other databases (not shown) in order to configure and partition by establishing shards and slicing each storage server in order to ensure that queries, when received by application **102**, are executed and sent responses by minimalizing search time. Distributed search platforms or applications such as Solr™ the Apache Lucene™ Foundation, or others may be used, in some examples, to implement search platform **112** to provide data searching functionality. In other examples, different search platforms or applications may be used and are not limited to the examples shown and described. Here, search platform **112** may be configured to receive a query from API **115** and, using the techniques described herein, "rewrite" (i.e., structure and configure a query request received by application **102** for a given data schema, without limitation or restriction to any particular data storage facility or schema shown or described) a query and execute it against a given data structure (i.e., database and data schema such as those implemented by one or more of index database **120**, database **122**, social database **124**, manifest files database **126**, metadata database **128**, and others).

[0034] Referring back to cluster manifest file manager **106**, in some examples, one or more of index database **120**, database **122**, social database **124**, manifest files database **126**, and metadata database **128** may be configured in one or more host groups. In some examples, host group configuration may be performed using a manifest file (not shown), which includes various server parameters, ports, services, rules, thresholds, limitations, restrictions, data transmission protocols, policies, addresses, partitioning (i.e., identifying, allocating, and/or configuring, among other functions shards, stripes, and partitions of servers), or any other setting that may be used to determine how to allocate storage resources for storing data received from and retrieved by application **102**. Further, as described herein, a manifest file may be used to configure one or more host groups in order to partition storage resources to temporally optimize data processing and storage functions.

[0035] In some examples, a manifest file may be used to configure a given host group to store data that is aged within a give time range. A time characteristic such as a time range may be measured in seconds, minutes, hours, days, weeks, months, or any other measurement of time that may be useful for server configuration. As an example, a host group may be configured using a manifest file to identify a given resource to store data and/or processed data generated between t=0 (i.e., time of initial generation (i.e., the time of posting, transmission, sent, or the like)) and t=86400 seconds, which may be a time range for saving data occurring in the first twenty-four (24) hours following data generation. As used herein, a “time characteristic” may refer to a time range, a time bucket, a bucket of time, a time bracket, or any expression that is used to encompass a discrete period of time, regardless of the actual unit of time measurement. Continuing the example described above, for data that “ages” or is “aged” past the first twenty-hour (24) hours, a different host group may be allocated for movement and storage of the data. In other words, as data to be stored in one or more of index database **120**, database **122**, social database **124**, manifest files database **126**, and metadata database **128** ages, the data may be moved from one host group to another; from a device or resource of a given server class to another device or resource of a different server class. This “bucketing” or sharding of data permits faster query responses for rewritten queries generated by query manager **114** because indexer **110** can return a location to be searched by search platform **112** for the requested (i.e., target) data to be returned. Further detail regarding temporal optimization of server resources for processing, storing, and retrieving data (e.g., social data, among others, without limitation or restriction) is described in greater detail below.

[0036] As described herein, any of index database **120**, database **122**, social database **124**, manifest files database **126**, and metadata database **128** may be configured using one or more manifest files to configure for temporal optimization of data processing, storage, and retrieval. In other words, database **122** may include multiple storage servers configured to store data for different time ranges and, as stored data (e.g., data and processed data that has been directed for storage by application **102**) ages, it may be moved between different partitions (e.g., shards, slices, stripes, and the like), host groups, and server classes in order to provide storage and retrieval functionality that is optimized to retrieve recently-generated data at higher data retrieval speeds using servers with data retrieval speeds and technologies that are faster and more expensive. However, by storing data based on aging, data storage can be optimized by targeting data within given time ranges onto different host groups and different server classes, each of which may have different costs associated with the use of servers assigned to these classes. In so doing, data storage is optimized to not only optimize storage and processor resource costs, but also minimize latency and increase accuracy of data retrieval.

[0037] In some examples, server configuration using a manifest file may be further performed by determining, for a given host group, a server type and class to be used to implement the host group. In the example above, data generated more recently (i.e., not aging significantly since initial generation from clients **130-136**) may be of greater interest to a user or community of users. As social media applications and networks can generate large quantities of data and numerous discrete data files (e.g., posts, “tweets,” messages, and the like) resulting in millions of users generating query requests for a given text, still image, audio, or video file nearly simultaneously, the techniques described herein for host group configuration (i.e., using a manifest file) can partition processing and storage resources based on time characteristics and assign different server classes having different access speeds that are allocated amongst data that is more likely to be requested in large numbers of query requests as opposed to aged data that may be stored on other servers or storage resources that have slower (i.e., and less expensive) server classes assigned. Using the described techniques, system **100** is capable of responding to large numbers of queries due to increased read-write-access capabilities (e.g., solid state memory technologies providing faster addressing speeds than mechanically-addressed memory technologies, quantum computing processors relative to conventional semiconductor, silicon-based memory technologies, and others) and provide increased performance by assigned to faster processing, storage, and retrieval resources more recent data (e.g., data from clients **130-136** within the first twenty-four (24) hours since initial generation)). Further, by periodically reconciling processing and storage resources, data may be moved between different server classes as aging increases, or other parameters or characteristics, in addition to time characteristics, change. In some examples, a periodicity for performing reconciliation may also be indicated in a manifest file. Thus, continuing the above example, when a query is received for recent data, query manager **114** may be configured to parse, evaluate, and re-write a given request into a rewritten query that is run against those servers identified by indexer **110** as being assigned to data stored in the first twenty-four (24) hours. As data is stored, in one or more of index database **120**, database **122**, social database **124**, manifest files database **126**, and metadata database **128**, indexer **110** may be configured to modify a server index (hereafter "index") to specify a location, including host group, server, shard, slice, partition, server class, or other parameter to be used by search platform **112** in executing a query. As shown, indices for given servers may be stored in index database **120**, among others. Data (e.g., actual data or copies thereof as generated by clients **130-136**) may be stored in one or more databases such as database **120** or others. Different types of data may also be segmented, isolated, or otherwise separated and stored in different host groups using the techniques described herein. For example, data generated from social media applications and networks may be stored as "social data" in social database **124**. Further examples of using different data storage resources may also include storing manifest files in manifest files database **126** and metadata in metadata database **128**, among others. As described herein, metadata may refer to descriptive data that is generated during the parsing and evaluation of data from clients **130-136** by application **102** or any modules or elements (e.g., logic module **104**, cluster manifest file manager **106**, resource manager **108**, indexer **110**, search platform **112**, and query manager **114**, among others that may not be shown or described in FIG. **1**). Metadata may be generated when application **102** parses data generated by clients **130-136** such as HTML or XML tags that may be used in a database to associate a given data file with another data file. For example, data received from clients **130-136** by application **102** may be parsed and evaluated prior to be stored on database **122**. A copy of the received data file(s) may be generated and the original data may be stored as a "master" version of the data file(s) while a copy is stored in another database (not shown) apart from database **122**. When storage resources (e.g., index database **120**, database **122**, social database **124**, manifest files database **126**, and metadata database **128**, among others) are updated due to changes in data file(s) indicated by more recent incoming data received by application **102**, metadata stored in metadata database **128** may be used to identify any copies of data file(s) intended for updating (e.g., due to data operations such as deleting, adding, or writing data to a given data file previously stored in a different state). In some examples, metadata may also be used to identify aging associated with a given data file in order to determine a host group and server class to which the data is moved in order to maintain temporal optimization.

[0038] In some examples, a process (e.g., an algorithm or set of algorithms that may or may not be logically related to update a given state of stored data and storage resources) known as reconciliation (as described in greater detail below) may be performed by system **100** and application **102**, index(s) are updated by indexer **110** to reflect modified locations and any parameters such as those specified above. Various types of data operations may be performed in order to "reconcile" a previously stored version of a data file with a more recently received data file by application **102**. In some examples, a master version of a data file (not shown) and a copy of a data file stored (not shown), for example, in database **122**, may be stored on a server shard located on a device of a server class that has the highest data retrieval rate in system **100**. A newly received data file may be evaluated by application **102** to determine that one or more data operations (e.g., add, delete, read, write, transfer, or others) is to be performed based on, for example, time characteristics that indicate the data file is to be transferred to another host group in which a different server class is used for data storage and retrieval. As described in greater detail below, reconciliation may be used to manage how data files are stored on one or more of index database **120**, database **122**, social database **124**, manifest files database **126**, and metadata database **128**, among others. In other examples, system **100** and the above-described elements may be implemented differently, with variations in function, order, procedure, and process, without limitation to any of the examples or accompanying descriptions.

[0039] FIG. **2** illustrates an exemplary application architecture for temporal optimization of data operations using distributed server and shard management. Here, an alternative view of application **102** is shown with application **202**, including logic module **104** (FIG. **1**), cluster manifest file manager **106** (FIG. **1**), resource manager **108** (FIG. **1**), indexer **110** (FIG. **1**), search platform **112** (FIG. **1**), query manager **114** (FIG. **1**), API **115** (FIG. **1**), shard manager **214**, class manager **216**, cluster/host group module **220**, and reconciliation module **222**. As shown, logic module **104**, cluster manifest file manager **106**, resource manager **108**, indexer **110**, search platform **112**, query manager **114**, and API **115** may be implemented substantially similarly to the like-numbered elements shown above and described in connection with FIG. **1**. Referring back to FIG. **2**, application **202** may be implemented as an alternative configuration of application **102** (FIG. **1**) with additional elements (i.e., shard manager **214**, class manager **216**, cluster/host group module **220**, and reconciliation module **222**) that may be implemented and configured to provide additional functionality. Structurally, in some examples, application **202**, like application **102**, and the elements shown and described may be implemented as hardware, software, firmware, logic-specific circuitry, or as a combination thereof, without restriction or limitation to any particular implementation environment, state, or configuration.

[0040] In some examples, application **202** using API **115** may be in data communication with one or more data processing and/or storage resources (not shown) over one or more data networks (not shown). Using a manifest file retrieved from manifest files database **126** (not shown), one or more host groups can be configured by application **202** using the techniques described herein. For example, shard manager **214** can be configured to determine how, across a number of storage resources (as identified by resource manager **108** (FIG. **1**)), shards can be partitioned and identified to store various segments of data and data files (as used herein, “data” and “data files” may be used interchangeably without limitation and are not subject to any particular implementation or configuration) based on a time associated with the generation, transmission, receipt, or other time characteristic associated with the data. When received and parsed, data files can be determined by application **202** to identify metadata (e.g., metadata stored in metadata database **128**) to indicate time characteristics such as those described herein. Time characteristics can, in some examples, be used to determine shards in which to store data and processed data, including determining individual partitions (e.g., shards, slices) and partitioning techniques such as striping. Here, shard manager **214** may be configured to determine how shards are partitioned across host groups managed by application **202** as well as identifying to cluster/host group module **220**, which shards are assigned to a given host group or, in some examples, whether given shards span multiple servers that are being used in different host groups, as described in greater detail below.

[0041] As shown, class manager **216** may be configured as a software, hardware, firmware, circuitry, or combination thereof and used as an individual component or element of application **202** to assign or determine the assignment of particular classes of servers to host groups being managed by application **202**. For example, if two host groups are each assigned a server class, server class 1 may be assigned to a host group as solid state memory for fast data access, retrieval, and data operations for data to be stored that was originally generated in the last 12 hours. Continuing the above example, data that has aged past 12 hours (e.g., 12 hours and 1 second or longer) may be moved to a different host group using older, mechanically-addressed memory technology (e.g., disk drives, and the like). The location of data stored on either of the above-referenced host groups may be identified to query manager **114** (FIG. **1**), which can track and associate queries intended for data within the first 12 hour time range or the “12 hour and older” time range and, in so doing, target more precisely and quickly, rewritten queries so as to not consume storage and/or processing resources searching host group(s) outside of the time range of requested data (i.e., as indicated in a query).

[0042] As described herein, class manager **216** may be configured to associate, assign, track, or identify which classes of servers are assigned to various host groups. Data transferred between class manager **216** and cluster/host group module **220** may be used by application **202** to implement a management system of various host groups and clusters of servers in order to deploy the techniques described herein. As used herein, clusters of servers may be used interchangeably with “server cluster” or “cluster” and is intended to refer to a computing cluster or node of servers that are logically assigned together as a group, which may include multiple instances of physical or virtual machines or servers, partitions, shards, slices, or the like. Further, cluster/host group module **220** may also exchange data with reconciliation module **222** in order to reconcile (i.e., perform reconciliation) data stored throughout various host groups and/or server clusters if, in some examples, new incoming data (e.g., input data streams for various social media, social networks, or social media-oriented applications, and the like) indicate that data previously stored has changed. As an example, when a change is detected (e.g., an older posted video or image is replaced with a newer version, an news article or feed is replaced with a recent news update, an audio file previously stored (e.g., database **122**) has been edited to be replaced by a newer version of the audio file, and the like), cluster/host group module **220** may send data as a command in order to trigger or initiate reconciliation module **222** to perform one or more data operations in order to effect the change. As used herein, “data operation” may refer to any type of computer programming or formatting operation that results in data being added, deleted, modified, replaced, changed, reordered, joined, shifted, or the like, without limitation or restriction to any particular type of database or data schema operation. As described in greater detail below, reconciliation may be performed on a schedule, intermittently, infrequently, frequently, periodically, rapidly, or on any type of schedule or random occurrence, without limitation or restriction. Reconciliation, in some examples, is used to ensure that host groups and clusters of servers managed by application **202** store updated data and data files, or copies thereof, in order to provide accurate and updated information in response to queries. Various types of programming and formatting languages may be used to implement reconciliation module **222** (or any of the elements shown in FIGS. **1** and **2**), including Scala, or others, without limitation or restriction. In other examples, application **202** and the above-described elements may be implemented differently, with variations in function, order, procedure, and process, without limitation to any of the examples or accompanying descriptions.

[0043] FIG. **3** illustrates an exemplary host group and server cluster configuration for temporal optimization of data operations using distributed server and shard management. Here, computing environment **300** includes application **202** (FIG. **2**), cluster/host group module **220** (FIG. **2**), reconciliation module **222** (FIG. **2**), databases **306-308**, data network **310**, clusters **312-316**, and servers **318-332**. In some examples, application **202** may be implemented similarly to the examples shown and described above. Similarly cluster/host group module **220** and reconciliation module **222** may also be implemented functionally and structurally in a substantially similar manner to that described above in connection with FIG. **2**. In other examples, application **202**, cluster/host group module **220**, and/or reconciliation module **222** may be implemented functionally and/or structurally differently than the examples shown and described above in connection with FIG. **2**.

[0044] Referring back to FIG. **3**, databases **306** and **308** may be placed in data communication with application **202**, directly or indirectly. For example, application **202** may access, store, or retrieve data (among other performing other data operations) in database **306** by transferring data over data network **310**. In some examples, data network **310** may be implemented as a computing cloud or distributed set of data networks that are configured to route and transfer data between application **202**, databases **306-308**, or servers **318-332**, the latter of which may be implemented using various types and quantities of servers, without limitation or restriction. Further, application **202** may also be in data communication with database **308** in order to access, store, or retrieve data, but using a direct data communication link. Various types of data communication protocols such as transport control protocol/internet protocol (TCP/IP), hypertext transfer protocol (HTTP or HTTPS), ICMP, or others may be used and are not intended to be a specific limitation or restriction to the techniques described herein.

[0045] In some examples, data or data files accessed, stored, or retrieved by application **202** from various sources (e.g., clients **130-136** (FIG. **1**)) may be stored in any of databases **306-308** or on any of servers **318-332** on clusters **312-316**, which may be configured into one or more host groups by application **202** using a manifest file (not shown). Here, application **202** may invoke cluster/host group module **220** to use configuration settings and parameters in a manifest file to configure clusters **312-316** into one or more host groups. In some examples, clusters **312-316** may represent individual host groups, wherein varying types and quantities of servers are implemented. For example, cluster **312** may be implemented using servers **318-320** of a given server class (e.g., class 1). Continuing the above example, cluster **314** may be implemented using servers **322-328**, which may be of a different server class than those deployed for cluster **312** (e.g., class 2). Further, cluster **316** may be implemented as another host group having servers **330-332**, which may of a similar or different class than those implemented for clusters **312-314**. Still further, clusters **312-316** could be grouped logically to create one or more addressable host groups. In other words, multiple clusters of servers can be grouped to create a host group and the examples shown and described are not intended to be limiting. In yet another example, more or fewer clusters and host groups using servers of similar or different classes can be implemented (i.e., deployed) and the examples shown and described are not intended to be limiting.

[0046] As shown in FIG. **3**, application **202** can use cluster/host group module **220** to configure and manage one or more host groups, as described above. Further, application **202** can generate control data and signals that are intended to control, manage, and logically group, address, or otherwise organize various servers into host groups and/or clusters by partitioning (e.g., sharding, slicing, striping, and the like) server resources to store data in a temporally-optimized manner. For example, if cluster **312** includes servers **318-320**, which are “class 1” servers that utilize fast, but expensive, solid state memory technologies that are capable of performing data operations at a rate faster than that of other classes (e.g., “class 2,” “class n,” or the like), application **202** (FIG. **2**) can designate cluster **312** to store and/or process data (i.e., data files) that has “aged” beyond an initial time range. In some examples, “aging” may refer to a time characteristic such as the length of time that has transpired since a subject data file was generated, transmitted, or otherwise initially received from a source (e.g., clients **130-136**). As data ages, system **300** may be configured by application **202** to move data as it ages from an initial period, for example, from server **318** to server **326** and, when further aging has occurred, to server **332**. The configuration of host groups and clusters (e.g., clusters **312-316**) may be performed by application **202** to optimize the use of different classes of servers (e.g., for storage, processing, or other operations) for different types of data, which may be segregated based on time characteristics such as aging. As described herein, temporal optimization of data processing and storage resources may be achieved by directing different classes of servers, which may be distinguished based on read/write/access speeds, performance, and cost, to store and/or process different types of data based on characteristics such as time characteristics (e.g., aging, time ranges, time buckets, or others). Temporal optimization storage and processing techniques described herein also enable data and data files (regardless of whether data (i.e., master versions of data and data files) or copies of data files) to be accessed and retrieved in response to queries rapidly and accurately by discretely identifying clusters and/or host groups on which requested data is stored. Further, temporal optimization may also be increased by using reconciliation, as described herein, to ensure that requested data and data files returned in response to queries are accurate and updated.

[0047] In some examples, reconciliation module **222** may be used (for example, in a Scala-

implemented architecture) to perform various data operations on data stored on one or more of clusters **312-316** and servers **318-332**. Data operations such as those described above may be performed by reconciliation module **222** when logic module **204** (FIG. **2**) detects, in newly received data, that a change has occurred or is indicated to data previously stored. In some examples, logic module **204** may generate control data and signals to servers **318-332** to identify a location, access the location, and perform a data operation on data or data files stored at the location. In other examples, logic module **204** may generate control data and signals to create a copy of the data targeted for data operations to be performed by reconciliation module **222** in order to keep an earlier version of the data, in addition to a master version of the data as it was originally received, for various purposes such as disaster recovery, catastrophic recovery, rollbacks (i.e., rolling back stored data to an earlier state in order to pre-data a catastrophic, destructive, or corrupting event). In other examples, system **300**, application **202**, and the above-described elements may be implemented differently, with variations in function, order, procedure, and process, without limitation to any of the examples or accompanying descriptions.
[0048] FIG. **4** illustrates an exemplary manifest file structure for temporal optimization of data operations using distributed server and shard management. As shown, data structure **400** may be an example of an implementation of a manifest file that may be used by application **102** (FIG. **1**) or **202** (FIGS. **2-3**) to configure one or more servers into one or more host groups and/or clusters to temporally optimize data storage, processing, and retrieval. As shown, data structure **400** includes substructure **402-408**, which may be used as described below. For example, substructure **402** may be used to present parameters and settings that, when received by a server (e.g., any of those described above) are used to designate a cluster name, shard size, slice size, and one or more host group names. In some examples, substructure **404** may be used to provide parameters and settings to configure a first set of hosts within the host group based on an assigned class of servers. Likewise, substructure **406** may be used to provide parameters and settings for configuring a second group of hosts assigned, within the host group, to one or more servers that may be implemented using devices of a different class than those used for the set of hosts above (e.g., Hosts A1 through A (n) comparative, based on server class, to Hosts B1 through B (n)). Finally, substructure **408** includes a file system prefix that may be used to identify manifest file represented by data structure **400**, without limitation or restriction to any particular parameters, settings, or examples shown and described. Further, the above-described exemplary data structure is intended to provide an illustrative example of how a manifest file may be implemented in various types of computer programming or formatting languages such as Java®, JavaScript, JSON, or others, without limitation or restriction. In other examples, data structure **400** and the above-described substructures **402-408** may be implemented differently, with variations in function, order, procedure, and process, without limitation to any of the examples or accompanying descriptions.
[0049] FIG. **5**A illustrates an exemplary data model and flow diagram for temporal optimization of data operations using distributed server and shard management. Here, data model **500** includes platform **508** (i.e., which may be implemented as a process, apparatus, system, device, machine (physical or virtual; singular or multiple instances), algorithm, or as a logical set of processes or algorithms (e.g., such as a platform application or system, which may be a collection of functions implemented using, for example, various types of software, firmware, hardware, circuitry, or a combination thereof)) and data types **504-516**, one or more of which may be used by platform **508** for various data operations. In some examples, data types **504-516** may include data **504**, query data **506**, cluster configuration data **510**, host group data **512-514**, and reconciliation data **516**. As shown in data model **500**, data **504** may be any type of data configured to be received by platform **508**, which may include, but is not limited to, digital and/or analog (e.g., binary) data in any type of format or schema, including, but not limited to files, packets, segments, frames, partitions, containers, .jar files, characters, strings, text, integers, real, Boolean, alphanumeric, ASCII, RSS, or any other type beyond those listed here), or others, without limitation or restriction. Data **504** may

include data from one or more other applications or clients (e.g., clients **130-136** (FIG. **1**); not shown) such as social media applications, social networks, applications (e.g., other computer programs, applications, operating systems, services, or the like), content feeds, computers, servers, mobile devices such as smart phones, tablet computers, or others without limitation. As an example, data **504** may be an input stream of data to platform **508** with social data (i.e., data from social media applications, social networks, or the like) including a single or multiple data files, each of which has been assigned a time and date stamp that might be included in, for example, a packet header, footer or payload. When parsed by platform **508**, a date and time may be determined and converted into a time period based on resolving the assigned time into a number of seconds, minutes, hours, or the like, as measured against a clock or clock signal (not shown) referenced by platform **508**. In some examples, a "stream" (e.g., continuous output of data) of data may be continuous, contiguous, regular, irregular, infrequent, periodic, occasional, on-demand, random, or the like. Once received by platform **508**, data **504** may be parsed, evaluated, and processed by platform **508**, application **102** (FIG. **1**), application **202** (FIG. **2**), or other elements shown and described above. In some examples, when data **504** is parsed and evaluated, other types of data (e.g., query data **506**, cluster configuration data **510**, host group data **512-514**, among others) may be generated.

[0050] In some examples, query data **506** may be received by platform **508** and, when received, may represent a query request initiated by a client (e.g., clients **130-136** (FIG. **1**); not shown), application, platform, system, or other computing device, machine, or process. A query, in some examples, may be a machine or algorithmically generated request for previously stored data, irrespective of data type, content or format. When platform **508** receives query data **506**, for example, parsing and evaluation may generated other data such as control signals or instructions to one or more storage locations implemented using virtual, physical, logical, cloud, or other types of storage or computing resources (e.g., storage servers). As described herein, temporal optimization of stored data may have stored the requested data at a location that is assigned to a given time range, which has been assigned to a given host group implemented using one or more server classes.

[0051] In some examples, platform **508** may rewrite or write a new query (not shown) using parsed data from query data **506** and send a rewritten request that has been targeted to a given host group based on determining a time associated with the requested data (i.e., stored data to be returned in response to the initial query). As an example, host group data **512-514** may include rewritten queries, queries generated by platform **508**, or other data or signals that, when sent by application **508**, or any of the elements shown and described herein, is used to executed one or more data operations on servers assigned to host groups, but which are targeted for execution based on temporal optimization and identification of time ranges in which queried data is assigned. In other words, host group data **512-514** may be issued to retrieve stored data from a given host group if, based on parsing query data **506**, a given host group or servers assigned to a host group have been partitioned to encompass the time range within which the requested stored data (i.e., data may be stored on a server, shard, slice, stripe, or other type of partition based on a time of when the data was generated, transmitted, received, or otherwise assigned a given time).

[0052] Referring back to FIG. 5A, host groups and servers assigned to given time ranges for purposes of storing data in a temporally optimized manner may be configured using data, signals, or instructions included within cluster configuration data **510**. In some examples, configuration of clusters, host groups, and servers may be performed, instructed, or otherwise managed by platform **508** using cluster manifest file **502** and cluster configuration data **510**. As described above, cluster manifest file **502** may be illustrative of data that includes manifest files (such as those described above) that may be used to configure one or more storage and/or processing resources in order to temporally optimize the storage and retrieval of data. As shown, cluster manifest file **502** and cluster configuration data **510** may include information on cluster and host group configuration, server configuration, shard determination and configuration, server striping (hereafter "striping"), server slice determination, or other partitioning and configuration data or information that may be used to determine how to store and retrieve (or otherwise process) data. In some examples, data is stored based on a time assigned to the stored data. Resources (e.g., clusters, host groups, servers, shards, slices, stripes, partitions, or the like) may be configured to store data according to time characteristics such as time ranges. As stored data ages (as described above), platform **508** may instruct moving data from a given resource (e.g., a location, server, host group, cluster, or the like) to another resource in response to reconciliation data **516**. In other words, platform **508** may provide data and instructions in reconciliation data **516** to determine whether any stored data, based on aging, should be deleted from a given resource and stored on another resource (e.g., moving data that has aged from a faster, more expensive class of server hardware to a slower, less expensive class of server hardware). In other examples, reconciliation data **516** may also include data or signals that may be used to modify, by executing various data operations (e.g., delete, copy, add, join, or others, without limitation or restriction), stored data (not shown). In other examples, data model **500** and the above-described elements may be implemented differently, with variations in function, order, procedure, or process, without limitation or restriction to any of the examples or accompanying descriptions.

[0053] FIG. **5**B illustrates an alternative exemplary data model and flow diagram for temporal optimization of data operations using distributed server and shard management. Here, data model **520** includes platform **508** (FIG. **5**A), which may be implemented in function and structure substantially similar to the like-numbered elements described above in connection with FIG. **5**A (i.e., manifest file **502**, query data **506**, cluster configuration data **510**, host group data **512-514**, and reconciliation data **516**. Here, social data **522** is input to platform **508** and, once parsed and evaluated, is stored in a host group. In some examples, social data **522** may be stored in a server at a location that is stored in a server index (hereafter "index"). A location (i.e., on a storage server) may be selected within a host group on a cluster of servers based on parsing and evaluating the social data to determine a time of transmission, generation, creation, receipt, or other event that, when detected by platform **508**, can be used to initially store social data **522** on a server. Further, when social data **522** ages past a given time range, platform **508** can copy social data **522**, delete it from the first location, and save it to another location on a different server associated with a slower class of server. In some examples, aging may refer to the amount of time that has accrued since the time of initial transmission, generation, posting, creation, placement, or other event that is considered to represent an initiating or originating time, t=0. As time passes, the time assessed to social data **522** increments in the unit of measurement (e.g., seconds, minutes, hours, days, weeks, months, years, decades, centuries, millennia, or a fraction thereof) assigned by platform **508** as a "time characteristic." When a time characteristic indicates that a given data file, packet, frame, segment, or the like has incremented so as to traverse a given time range to a different time range, platform **508** may generate data and signals to direct moving social data **522** (similarly to data **504** (FIG. **5**A)) from one location to another, which may occur by performing a series of data operations, for example, to copy stored data from a given location, deleting the stored data from the location, and writing the copy to another location (e.g., "moving" stored data from a host group to another host group, from a server class to another server class, from a cluster to another cluster, from a server to another server, from a shard to another shard, from a slice to another slice, from a partition to another partition, and the like). Consequently, as data ages and is moved to temporally optimize data storage and retrieval operations, instructions to store data and queries requesting particular data (e.g., a query is received for a given content item, which may be a blog post, a news article, a news item from a feed, a video or audio file, or others, without limitation or restriction to any particular type or format of data that can be queried from a storage server or database) may be performed at decreased speeds, reducing processor demands and latency times for generating responses or returning data in response to queries. In other examples, data model **520** and the above-described elements may be implemented differently, with variations in function, order, procedure, or process, without limitation or restriction to any of the examples or accompanying descriptions.

[0054] FIG. **5**C illustrates a further exemplary data model and flow diagram for temporal optimization of data operations using distributed server and shard management. Here, data model **530** illustrates exemplary flow of data **532** across servers **534-542**, which may be implemented using different classes of servers. As shown, data **532** may be directed, instructed, or otherwise signaled (e.g., by application **102** (FIG. **1**), application **202** (FIG. **2**), platform **508** (FIGS. **5**A-**5**B), or other elements such as those shown and described herein) to be stored in one or more storage servers (not shown). Here, a time scale (designed by variable "t") is shown across which storage and/or retrieval activity of data **532** occurs over servers A-n (servers **534-542**), which may be implemented using different classes of servers (e.g., i2, r3, r(n), and others, without limitation or restriction as to server type or schema, including, but not limited to, physical, virtual, logical, or others).

[0055] As shown, for example, at time t=0 data **532** is stored on server **534**, which may be implemented as a class "i2" server. As used herein, arbitrary designation of servers are used for purposes of illustration and the exemplary designations of server classes (e.g., i2, r3, r(n)) are not intended to be specific, limiting, or restrictive, and other server classes apart from those shown and described may be used. As used herein, "server class" or "class" may refer to a type or categorization of processing or storage server based on various attributes and characteristics such as memory technology type (e.g., solid state, random access memory (RAM), read only memory (ROM), disk drive, quantum computing, semiconductor, or others), data transfer rates, storage capacity or volume, compatibility (e.g., for use in arrays such as a redundant array of independent disks ("RAID")), or others, without limitation or restriction to any particular or specific type of class.

[0056] Here, as data **532** ages towards time t=a, data **532** may be retrieved from storage server **534** as a copy, deleted from server **534**, and stored on server **536** until time t=b. At t=b, data **532** may be moved server **536** to server **538**, which may be implemented as a different class of server (e.g., r3) until t=c, at which time data **532** may be moved in further data operations, as illustrated by server **542**, which is representative of any number of servers and classes of servers using the techniques described herein. As used herein movement of data may refer to data retrieval operations to remove data from a location on a server, performing another data operation to delete the data from the location at the server, and performing a further data operation to write (i.e., save) the data to a new location on a different server; although the order of deleting and writing may be performed synchronously, simultaneously, in reverse order, or in no particular order. In other examples, the above-described data model **530** and corresponding flow description may be varied in process, order, or other aspects, without limitation or restriction to the examples shown and described.

[0057] FIG. **6** illustrates an exemplary process for temporal optimization of data operations using distributed server and shard management. Here, process **600** may be implemented as an algorithm, program, application, process, method, procedure, operation, or a group thereof configured to be performed by application **102** (FIG. **1**), application **202** (FIG. **2**), platform **508** (FIGS. **5**A-**5**B), or another instance of the described techniques. In some examples, process **600** starts by configuring a data network using a manifest file, as described above (**602**). Using a manifest file, a cluster or set of clusters having one or more host groups that, in turn, have one or more servers configured for storage or processing, may be used to provide parameter and settings to configure servers to store data for temporal optimization of data operations to store and retrieve data (e.g., in response to queries). In some examples, data (which may include any type of data, such as those described herein including, but not limited to, social data) is received at a platform (e.g., platform **508** (FIGS. **5**A-**5**B)) (**604**). Once received at platform **508**, one or more elements such as those described above in connection with FIGS. **1-3** may be configured to parse the received data (**606**) to determine, for example, a time characteristic such as a time of generation, transmission, creation, or other attribute of the received data, which may be a post to a social media application or network, as an example. In other examples, data received by platform **508** (not shown) may be parsed to determine other time characteristics, parameters, or other quantitative or qualitative attributes beyond those described herein and are not limited to any specific example shown or described.

[0058] Referring back to FIG. **6**, once parsed to determine a time characteristic (e.g., a time associated with an event of the data), the data may be stored to a location on one or more servers of a given server class (**608**). As storage resources in data communication with a platform (e.g., platform **508** (FIGS. **5**A-**5**B) have been configured using a manifest file (not shown), the stored data is written to a location (e.g., memory location) that is either annotated or referenced by, for example, a pointer in a server index. By referencing the index, the pointer can direct queries from platform **508** to the location where requested data has been written and can be retrieved (**610**). Once stored, data can be reconciled, as described herein, at various, scheduled, regular, irregular, periodic, occasional, on-demand, or random time intervals (**612**). In other examples, the above-described process **600** may be varied in process, order, steps, sub-processes, or other aspects, without limitation or restriction to the examples shown and described.

[0059] FIG. **7**A illustrates another exemplary process for temporal optimization of data operations using distributed server and shard management. Here, process **700** may be an alternative implementation of a temporal optimization process practicing the techniques described herein which, in some examples, may commence by configuring a host group having one or more servers (e.g., configured to provide processor or storage resources) using a manifest file as described herein (**702**). In some examples, configuration of server resources using a manifest file is described in further detail below in connection with FIG. **7**B. Referring back to FIG. **7**A, data may be received by, for example, application **102** (FIG. **1**), application **202** (FIGS. **2-3**), or platform **508** (FIGS. **5**A-**5**B) over an application programming interface (hereafter referred to as “API”) (**704**). In some examples, API may be used to provide an adapter or data communication structure to facilitate data transfer between, for example, application **102** (FIG. **1**), application **202** (FIGS. **2-3**), or platform **508** (FIGS. **5**A-**5**B) and one or more applications, networks, devices, clients, computers, servers, hosts, or any other type of computing device or machine, physical or virtual, remote or local, single instance or distributed, without limitation or restriction. In other examples, multiple APIs may be implemented and the techniques described herein are not limited to any particular number of instances or specific configurations of said instances of an API.

[0060] In some examples, after receiving data from an API, the data may be parsed by application **102** (FIG. **1**), application **202** (FIGS. **2-3**), platform **508** (FIGS. **5**A-**5**B), or one or more modules thereof such as those described above in connection with FIGS. **1-3**. For example, logic module **104** (FIGS. **1-2**) may parse received data to generate a processed data file, which may include, for example, time characteristics such as a time associated with the initial posting of a “blog” (i.e., a log configured for display on multiple devices over the World Wide Web (“Web”)) entry, or a user's post to a social media site, application, or network (e.g., Instagram™, Facebook®, Snapchat®, or others), or the like (**706**). Once parsed and processed, received data or data files may be stored at a location on a server, host group, cluster, or a set thereof based on the assignment of the data to a given time range and, subsequently, a server class. Once stored, the location of the stored data may be referenced in an index by using various types of reference techniques such as pointers, addresses, and the like (**708**). Once stored, additional data may be received that, when parsed, may indicate one or more data operation that need to be performed in order to reconcile changes (**710**). As described herein, data may be reconciled by performing one or more data operations to delete, modify, move (e.g., as described above and in reaction to aging from an initial event such as generation, creation, initialization, posting, publishing, or others), retrieve, copy, or other operations. In other examples, the above-described process **700** may be varied in process, order, steps, sub-processes, or other aspects, without limitation or restriction to the examples shown and

described.
[0061] FIG. **7**B illustrates an exemplary process for host group and server configuration for temporal optimization of data operations using distributed server and shard management. Here, process **714** is shown and described, illustrating an exemplary process for configuring a cluster, host group, server, or a set thereof using a manifest file. Initially, a determination is made by, for example, application **102** (FIG. **1**), application **202** (FIGS. **2**-**3**), platform **508** (FIGS. **5**A-**5**B) to determine whether multiple or a single server class is available for configuration using a manifest file (**720**). If multiple classes of servers are available, multiple host groups are created and configured in accordance with indicated parameters and settings in a manifest file. In some examples, available server resources may include, but are not limited, to local, distributed, or cloud-based storage and processing resources made available by service networks such as Amazon™ Web Services ("AWS") as provided by Amazon, Inc. of Seattle, Washington, among others, without limitation or restriction.
[0062] Referring back to FIG. **7**B, a time range may be designated to represent a time range of data to be stored on each resource assigned to a given server class, if multiple server classes are available. In other examples, multiple time ranges may be further associated with each resource of a host group of a server class (**724**). In some examples, time ranges may be identified as aging (as described herein) occurs, subsequently generating, from available server resources, additional clusters, host groups, shards, and slices of different servers. Partitioning of available server resources may be performed by identifying and assigning shards and slices of each available server resource or striping server resources in order to determine how and when data is to be stored (i.e., multiple instances or a single instance of a physical or virtual (i.e., virtual machine) storage or processor-configured server) based on control data and signals generated by application **102** (FIG. **1**), application **202** (FIGS. **2**-**3**), and platform **508** (FIGS. **5**A-B) (**726**). Here, multiple server classes may be assigned to different time ranges in order to segregate or separate stored data into different "time buckets" (i.e., ranges of time that indicate locations for data to be stored based on time characteristics associated with or determined or parsed from received data). Available server resources are then assigned to host groups configured for temporal optimization of data storage, processing, and retrieval operations (**728**).
[0063] Alternatively, if process **720** determines there is a single class of servers, data received by application **102** (FIG. **1**), application **202** (FIGS. **2**-**3**), platform **508** (FIGS. **5**A-**5**B may be processed (e.g., performing any type of data operations using available server resources of the single server class, without limitation or restriction) or stored (**730**). Once stored, reconciliation may be performed as described herein and in connection with FIG. **7**C below. In other examples, the above-described process **714** may be varied in process, order, steps, sub-processes, or other aspects, without limitation or restriction to the examples shown and described.
[0064] FIG. **7**C illustrates an exemplary process for reconciliation operations for temporal optimization of data operations using distributed server and shard management. Here, process **740** begins by invoking reconciliation, which may be a single instruction or algorithm or a logical group or algorithms, programs, applications, or processes that are configured to reconcile one or more available server resources configured for temporal optimization to store or retrieve data (**742**). In some examples, a determination is made as to whether data being received by, for example, application **102** (FIG. **1**), application **202** (FIGS. **2**-**3**), platform **508** (FIGS. **5**A-**5**B) indicates changes (i.e., modifications in the current state of stored data) are present (**744**). If received data (e.g., newly received data or data received by application **102** (FIG. **1**), application **202** (FIGS. **2**-**3**), platform **508** (FIGS. **5**A-**5**B)) in an input stream, feed, data received after previously received data is stored, or the like) indicates no changes are to be made to stored data, process **740** ends. However, if changes are indicated by data received after the stored data was received by application **102** (FIG. **1**), application **202** (FIGS. **2**-**3**), platform **508** (FIGS. **5**A-**5**B), or a sub-module thereof (e.g., modules such as those indicated in FIGS. **1**-**3**), then the type of change is

determined by parsing the received data (**746**).
[0065] In some examples, determining the type of change that has occurred to previously stored data may include identifying one or more data operations to be performed on stored data. Once the type(s) of change has been determined and one or more data operations associated with enacting the indicated change(s) has also been determined, a server index may be referenced to determine a location associated with the targeted stored data. After determining the location of the stored data targeted for change (i.e., modification) by referencing a pointer, address, or other locating information included within a data structure configured as a server index ("index"), the associated data operation(s) are performed (**748**). As used herein, data operation may refer to any type of algorithmic, digital, software, method, or process that can modify data, regardless of programming language, format, or application architectural level (e.g., various levels of an application "stack" as identified in various types of models such as the Open Systems Interconnect ("OSI") model).
[0066] In some examples, performance of a data operation may include identifying a location of stored data indicated for change by received data (i.e., data received after the target data was stored), copying the stored data in its unmodified state to a cache, buffer, or temporary memory location, performing the data operation on the copy of the stored data, and storing the modified stored data to the original location from which the data was retrieved or to a different location, if specified by data received by application **102** (FIG. **1**), application **202** (FIGS. **2-3**), platform **508** (FIGS. **5**A-**5**B), or a sub-module thereof (e.g., modules such as those indicated in FIGS. **1-3**) (**750**). In other examples, the above-described process **740** may be varied in process, order, steps, sub-processes, or other aspects, without limitation or restriction to the examples shown and described.
[0067] FIG. **8**A illustrates an alternative exemplary process for temporal optimization of data operations using distributed server and shard management. Here, process **800** may be implemented as an alternative implementation or embodiment of process **700** described above in connection with FIG. **7**A. In some examples, process **800** starts by configuring clusters, host groups, and available server sources (e.g., individual or multiple instances of servers (physical, virtual, or logical), server classes, or the like) by making assignments based on time characteristics. As described above, available server resources and server classes may be assigned based on time characteristics such as time ranges into which stored data can be grouped (**802**). As shown, striping of available server resources and generation and assignment of shards (as described above) may be assigned to available storage resources based on time characteristics (**804**). A query requesting data retrieval may be received by application **102** (FIG. **1**), application **202** (FIGS. **2-3**), platform **508** (FIGS. **5**A-**5**B), or a sub-module thereof (e.g., modules such as those indicated in FIGS. **1-3**) (**806**). As shown, when a query requesting data retrieval of stored data is received, it is further parsed and evaluated to identify a time characteristic (e.g., a time range in which the requested data was stored) associated with the stored data (**808**). For example, a received query may indicate that a news article posted eighteen (18) hours earlier is being requested by another client, application, system, or the like, which, once identified, indicates that a given time range and assigned host group and its corresponding server resources (which may be of one or more server classes) is to be searched. As another example, a query is received requesting a copy of a stored data feed in which various clients are posting different types of content (e.g., content, still images, burst image files, video, audio, or others, without limitation or restriction) to be copied and, further, additional data related to the stored data and which is being stored is to also be copied in real-time or near real-time and returned in response to the query continuously. Other examples may be envisioned by those of ordinary skill in the art, and are not intended to be limited or restricted to the examples shown and described herein.
[0068] After identifying a time characteristic associated with the stored data sought to be retrieved in response to a query, in some examples, a location is determined to identify one or more server resources on which the stored data is housed (**810**). Upon identifying a location(s) of the stored data to be retrieved, application **102** (FIG. **1**), application **202** (FIGS. **2-3**), platform **508** (FIGS.

5A-**5**B), or a sub-module thereof (e.g., modules such as those indicated in FIGS. **1**-**3**) generates (i.e., rewrites the initial query) a request to the location to copy the stored data to a cache, buffer, or other temporary or permanent memory location, which may be collectively referred to as "retrieval" (**812**). As used herein, when a query is rewritten, as an example, a request to a storage server location (i.e., "location") may be formatted based on the class and data storage schema of the location, but also targeted at a given cluster, host group, and/or server based on comparing the determined time characteristic with one of the indexed time ranges associated with available server resources. By comparing the time characteristic of the requested stored data and identifying a time range within which the stored data is likely to be stored, server resources assigned to time ranges inside or outside that of the stored data may be excluded, thus preventing unnecessary server and processor resources being consumed. As another example, a precise location of data being requested by a given query may not be readily identified within a server index. However, by parsing the query to determine a characteristic for the data sought, an enveloping time range can be identified and, once identified, assigned server resources can be searched in a targeted manner avoiding expending processor and computing sources on unrelated server resources, which not only reduces latency in returning accurate results in response to queries, but also improves the allocation of server resources to storage of large amounts of stored. In some examples, when stored data requested for retrieval is copied into a cache, buffer, or other temporary or permanent memory location, a copy may be maintained for disaster or catastrophic recovery or rollback (i.e., resetting a state of a memory device to a previous state in order to restore data from a destructive, catastrophic, or corrupting event) purposes. In other examples, when stored data is requested for retrieval in response to a query, a copy is made to a cache, buffer, or other temporary or permanent memory location and then the stored data is deleted from the location. In still other examples, stored data may be handled differently and is not limited to the specific examples shown and described. In other examples, the above-described process **800** may be varied in process, order, steps, sub-processes, or other aspects, without limitation or restriction to the examples shown and described.
[0069] FIG. **8**B illustrates an exemplary process for temporally optimized data retrieval using distributed server and shard management. Here, process **820** starts by determining clusters and host groups to be configured (**822**). Upon determining the number and configuration of clusters and/or host groups to be configured, application **102** (FIG. **1**), application **202** (FIGS. **2**-**3**), platform **508** (FIGS. **5**A-**5**B), or a sub-module thereof (e.g., logic module **104** (FIGS. **1**-**2**)) determines times ranges to assign to each host group or, in some examples, individual hosts, or sub-host groups (**824**). Next, a determination is made as to which server classes are available (**826**). Once available server classes have been determined, server classes are assigned to host groups, individual hosts, or sub-host groups being temporally optimized for data retrieval (**828**). Once assigned, shards may be determined for assigned to various time ranges assigned to host groups (**830**). In other words, some shards may be implemented using one or multiple server resources (e.g., physical, virtual, or logical) of a single or multiple server classes. Logically, shards may be assigned to a given time range and server class, but actually implemented using a server resource or a fraction thereof, in some examples.
[0070] After configuring cluster(s), host group(s), server resources, and shards, a query requesting stored data may be rewritten in the data schema, format, or programming or formatting language of databases installed the elements set forth above (**832**). In some examples, queries may also be rewritten in order to be executed using a given search facility, application, platform, or system. For example, queries may be rewritten by query manager **114** (FIG. **2**) in order to transform a query from a web-input, text-based submission from one or more of clients **130**-**136** (FIG. **1**) into a query request in Solr, which may be used as an enterprise-level search platform. When rewritten, a Solr query may be run against various, disparate databases and server resources within a given time range in order to locate, retrieve (as discussed above), and return targeted stored data in response to

the original request. By using an enterprise-level search platform (e.g., Solr) that is configured to search various, disparate databases and data structures deployed on server resources that have been configured to store data on host groups supporting different server classes, latency is decreased and accuracy is increased in return stored data in response to queries by avoiding the need to search all server resources deployed for a given cluster or host group. In other examples, the above-described process **820** may be varied in process, order, steps, sub-processes, or other aspects, without limitation or restriction to the examples shown and described.

[0071] FIG. **8**C illustrates an exemplary process for temporally optimized data queries using distributed server and shard management. Here, process **840** starts by logic module **104** (FIGS. **1-2**) initiating evaluation of a query received by, for example, application **102** (FIG. **1**), application **202** (FIGS. **2-3**), platform **508** (FIGS. **5**A-**5**B), or a sub-module thereof (e.g., logic module **104** (FIGS. **1-2**)) (**842**). Once initiated, query manager **114** (FIGS. **1-2**) parses the received query to identify one or more time characteristics (**844**). Once determined, the identified time characteristics may be used by logic module **104**, class manager **216** (FIG. **2**), and cluster/host group module **220** (FIG. **2**) to identify a given host group and one or more server resources and classes assigned to the host group on which the requested data is likely stored (**846**). In some examples, search platform **112** (FIGS. **1-2**) may be configured to rewrite and execute a received query in a data schema and format of a given host group, server, or class as identified in sub-process **846** above (**848**), which is described in greater detail below in connection with FIG. **8**D.

[0072] In some examples, when a rewritten query is executed by search platform **114** (e.g., in Solr or another enterprise-level search language or format), a copy of the data being sought by the query is retrieved (**850**). Once retrieved, the copy of the requested data is returned in response to the request in a format and language (e.g., HTML, XML, Java®, JavaScript, command line interface, binary, or any other type of programming or formatting language that may be used to display data) to the client that initiated the query (**852**). In other examples, the above-described process **840** may be varied in process, order, steps, sub-processes, or other aspects, without limitation or restriction to the examples shown and described.

[0073] FIG. **8**D illustrates an exemplary process for retrieving temporally optimized data using distributed server and shard management. Here, an exemplary process **860** for rewriting queries using a search platform (e.g., search platform **112** (FIGS. **1-2**)) is shown and described. In some examples, a query is evaluated to determine a value associated with the queried or requested data (**862**). For example, a query may request stored data has an assigned value such as a hash value or index value that, when matched with a value in a query, enables requested stored data to be located rapidly without searching all server resources to find a complete and comprehensive match of each element of the stored data. In other words, instead of performing a search of all stored data to located a requested data file or set of data, data, when stored, may also be hashed or transformed into a value that is stored in an index file. When a query is executed a value is submitted to be matched in order to identify a location from which to retrieve stored data to return in response to the query.

[0074] Here, the value determined from a received query is used to reference an index to determine a match to locate stored data to be returned in response to a query (**864**). In some examples, a determination is then made to determine if a value in the index matches the value parsed from the original query (**868**). If not match is found, an error message may be returned and the process ends (**870**). If a match is found, then the matched value is converted into the data schema for the server resource on which the requested data is stored (**872**). The requested data is then retrieved and returned in response to the original query (**874**). In other examples, the above-described process **860** may be varied in process, order, steps, sub-processes, or other aspects, without limitation or restriction to the examples shown and described.

[0075] FIG. **8**E illustrates an alternative exemplary process for retrieving temporally optimized data using distributed server and shard management. Here, process **880** illustrates an alternative process for rewriting and executing a query in a database format or schema for a target server resource. In some examples, several sub-processes of process **880** may be similar to those described above in connection with FIG. **8**D. For example, when a query is received, it is evaluated to determine or identify a value associated with or indicated of the data requested (**862**; FIG. **8**D). Once a value has been determined, a location associated with the value is identified to the extent that a host group and/or a server class are identified for querying (**863**). Once a host group and a server class have been identified, an index associated with the host group can be referenced with the value to find a matching value that provides a location of the requested data (**864**; FIG. **8**D). In some examples, a determination is made as to whether the value is found in the referenced index (**868**; FIG. **8**D). If a match is found, a given location may be found using, for example, an address, pointer, or other indexing information associated with the matched value (**872**; FIG. **8**D). A copy of the data may be retrieved from the location and returned in response to the original request (**874**; FIG. **8**D). However, unlike FIG. **8**D, when a value is not found to match another value in an index, a different process may be invoked.

[0076] In some examples, when a value parsed from a query does not have a corresponding matching value in an index for a server resource(s) assigned to a host group assigned to a time range in which the requested data is stored, a master copy of the requested data may be located using metadata parsed from the original query. For example, a query may be parsed to identify various types and formats of metadata that may be used to identify data requested. Using the metadata, a search may be performed of the host group and associated server resources to locate a master copy of the requested data (**882**). Once located, a copy of the master data (or master data file(s)) may be retrieved (**884**). Once retrieved, in some examples, a data operation to reconcile the stored data may be performed by storing a copy of the master data at the location indicated by the index (**886**). After storing the copy of the master data at the location where the queried data should have been stored (i.e., but was damaged or deleted), another copy of the master data is returned in response to the original query (**888**). In some examples, the above-described process **880** may be used to replace, modify, fix, replace, address, write, or otherwise save data to a location that may have been corrupted for various reasons, including, but not limited, to power outages, disasters, catastrophic loss, server damage, corrupted memory or data structures, or others. In other examples, when data stored on a server resource does not match a state indicated by an index, process **880** may be used to restore stored data to an uncorrupted or undamaged state. In other examples, the above-described process **880** may be varied in process, order, steps, sub-processes, or other aspects, without limitation or restriction to the examples shown and described.

[0077] FIG. **9** illustrates an exemplary computing system suitable for temporal optimization of data operations using distributed server and shard management. In some examples, computer system **900** may be used to implement computer programs, applications, methods, processes, or other software to perform the above-described techniques. Computing system **900** includes a bus **902** or other communication mechanism for communicating information, which interconnects subsystems and devices, such as processor **904**, system memory **906** (e.g., RAM), storage device **908** (e.g., ROM), disk drive **910** (e.g., magnetic or optical), communication interface **912** (e.g., modem or Ethernet card), display **914** (e.g., CRT or LCD), input device **916** (e.g., keyboard), cursor control **918** (e.g., mouse or trackball), communication link **920**, and network **922**.

[0078] According to some examples, computing system **900** performs specific operations by processor **904** executing one or more sequences of one or more instructions stored in system memory **906**. Such instructions may be read into system memory **906** from another computer readable medium, such as static storage device **908** or disk drive **910**. In some examples, hard-wired circuitry may be used in place of or in combination with software instructions for implementation.

[0079] The term “computer readable medium” refers to any tangible medium that participates in providing instructions to processor **904** for execution. Such a medium may take many forms,

including but not limited to, non-volatile media and volatile media. Non-volatile media includes, for example, optical or magnetic disks, such as disk drive **910**. Volatile media includes dynamic memory, such as system memory **906**.
[0080] Common forms of computer readable media includes, for example, floppy disk, flexible disk, hard disk, magnetic tape, any other magnetic medium, CD-ROM, any other optical medium, punch cards, paper tape, any other physical medium with patterns of holes, RAM, PROM, EPROM, FLASH-EPROM, any other memory chip or cartridge, or any other medium from which a computer can read.
[0081] Instructions may further be transmitted or received using a transmission medium. The term "transmission medium" may include any tangible or intangible medium that is capable of storing, encoding or carrying instructions for execution by the machine, and includes digital or analog communications signals or other intangible medium to facilitate communication of such instructions. Transmission media includes coaxial cables, copper wire, and fiber optics, including wires that comprise bus **902** for transmitting a computer data signal.
[0082] In some examples, execution of the sequences of instructions may be performed by a single computer system **900**. According to some examples, two or more computing system **900** coupled by communication link **920** (e.g., LAN, PSTN, or wireless network) may perform the sequence of instructions in coordination with one another. Computing system **900** may transmit and receive messages, data, and instructions, including program, i.e., application code, through communication link **920** and communication interface **912**. Received program code may be executed by processor **904** as it is received, and/or stored in disk drive **910**, or other non-volatile storage for later execution. In other examples, the above-described techniques may be implemented differently in design, function, and/or structure and are not intended to be limited to the examples described and/or shown in the drawings.
[0083] Although the foregoing examples have been described in some detail for purposes of clarity of understanding, the above-described inventive techniques are not limited to the details provided. There are many alternative ways of implementing the above-described invention techniques. The disclosed examples are illustrative and not restrictive.

Claims

1. A method, comprising: receiving data in association with an application implemented with one or more processors and memory, the application being configured to manage disparate computing and storage resources; accessing, by the application, a manifest file including data representing parameters to configure one or more clusters of servers to allocate storage of the data among partitioned portions of the one or more clusters of servers in accordance with the parameters specifying classes of servers and ranges of time characteristics; determining aging of partitioned data associated with the partitioned portions based on the classes of servers and the ranges of time characteristics; moving one or more subsets of the partitioned data associated with one or more of the partitioned portions to different partitioned portions of different server classes based on the aging of the partitioned data, each of the different server classes having an access speed, the different server classes being assigned to each of the one or more partitioned portions based on the access speed of a server class, the access speed and the server class being determined based on an anticipated number of query requests requesting retrieval of the partitioned data associated with each of the one or more partitioned portions; receiving a query at a processor to retrieve the data stored as stored data among the among the partitioned portions; evaluating the query to identify a time characteristic associated with the stored data; identifying a location from which to retrieve the stored data based on the query to determine a partitioned portion of the different server classes in which the stored data resides as a function of the time characteristic; and rewriting the query to run on at least one server of the one or more clusters of servers.