

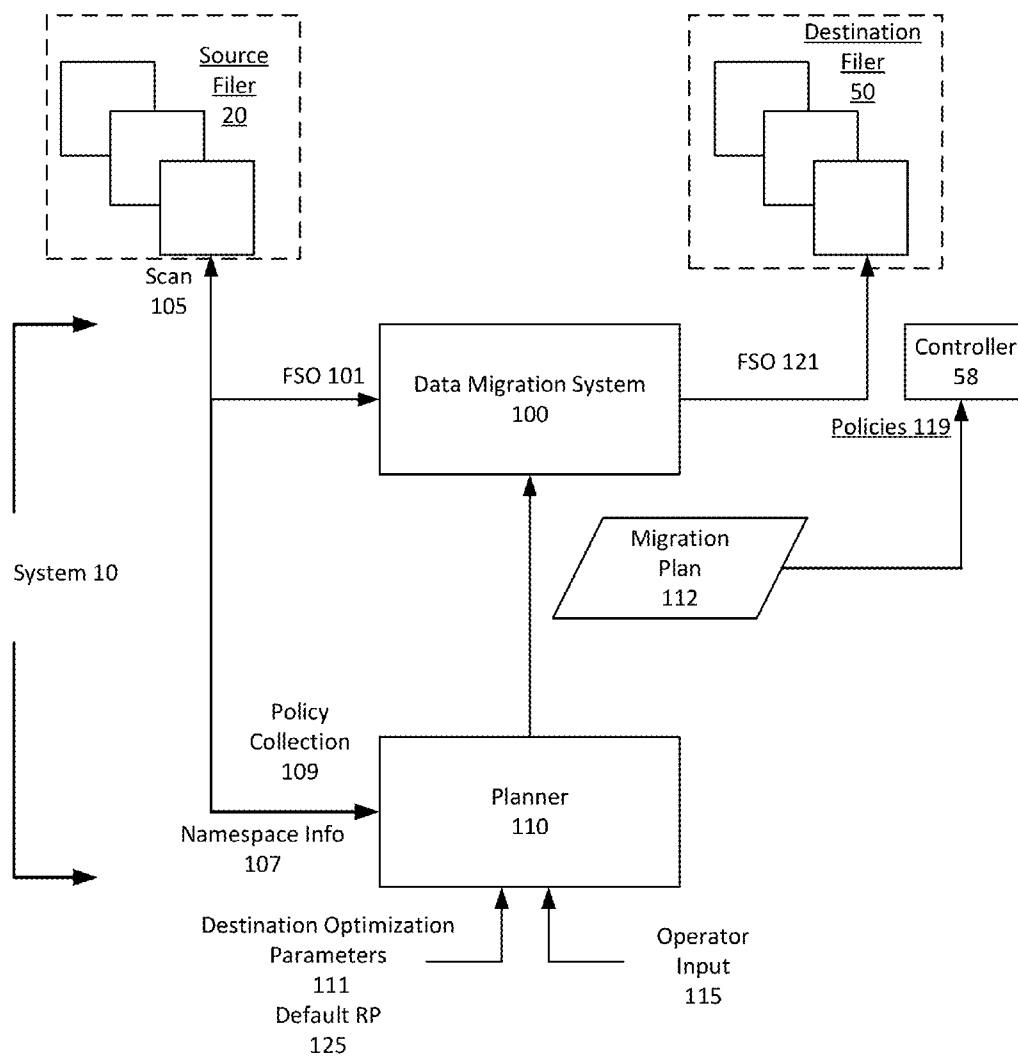


FIG. 1

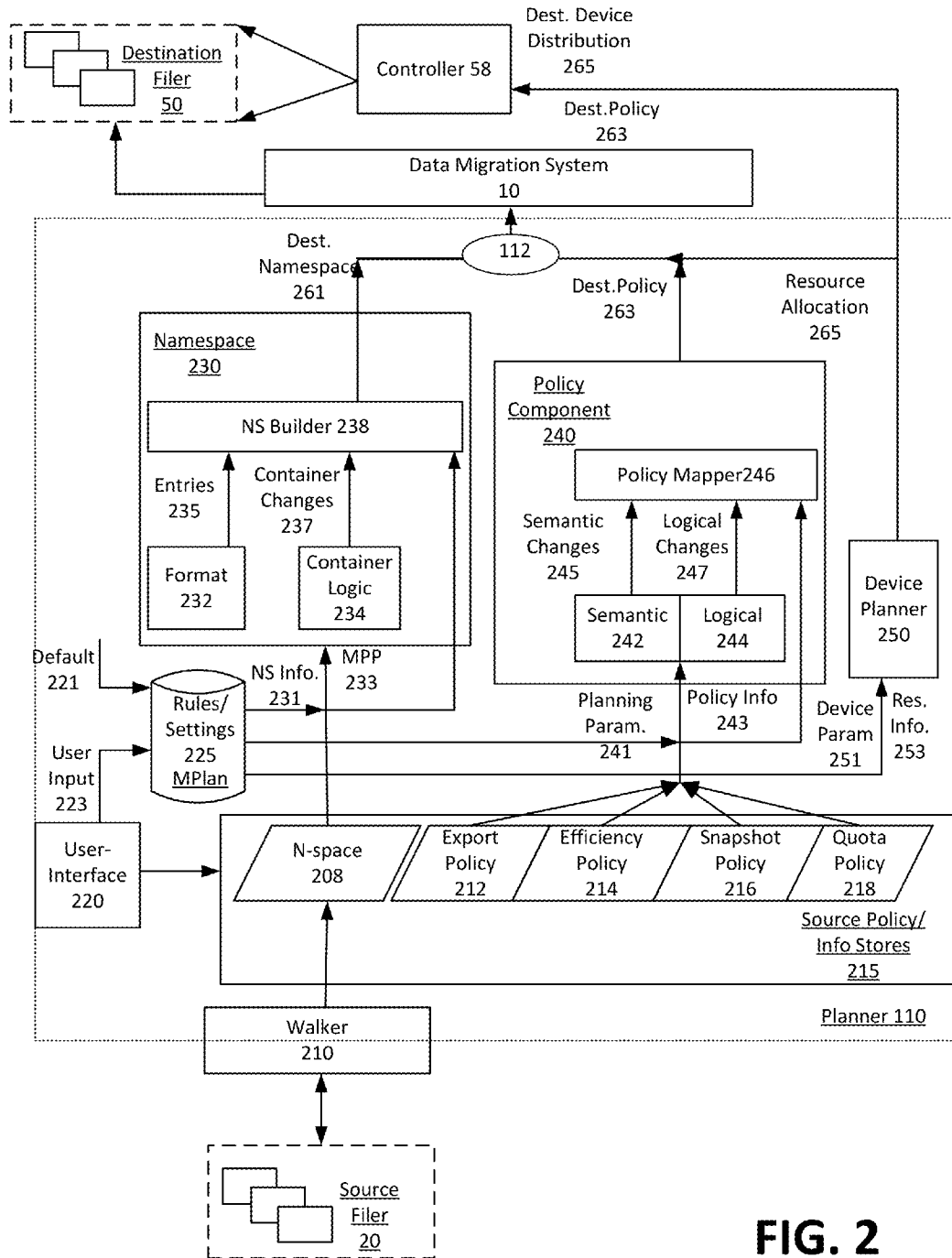
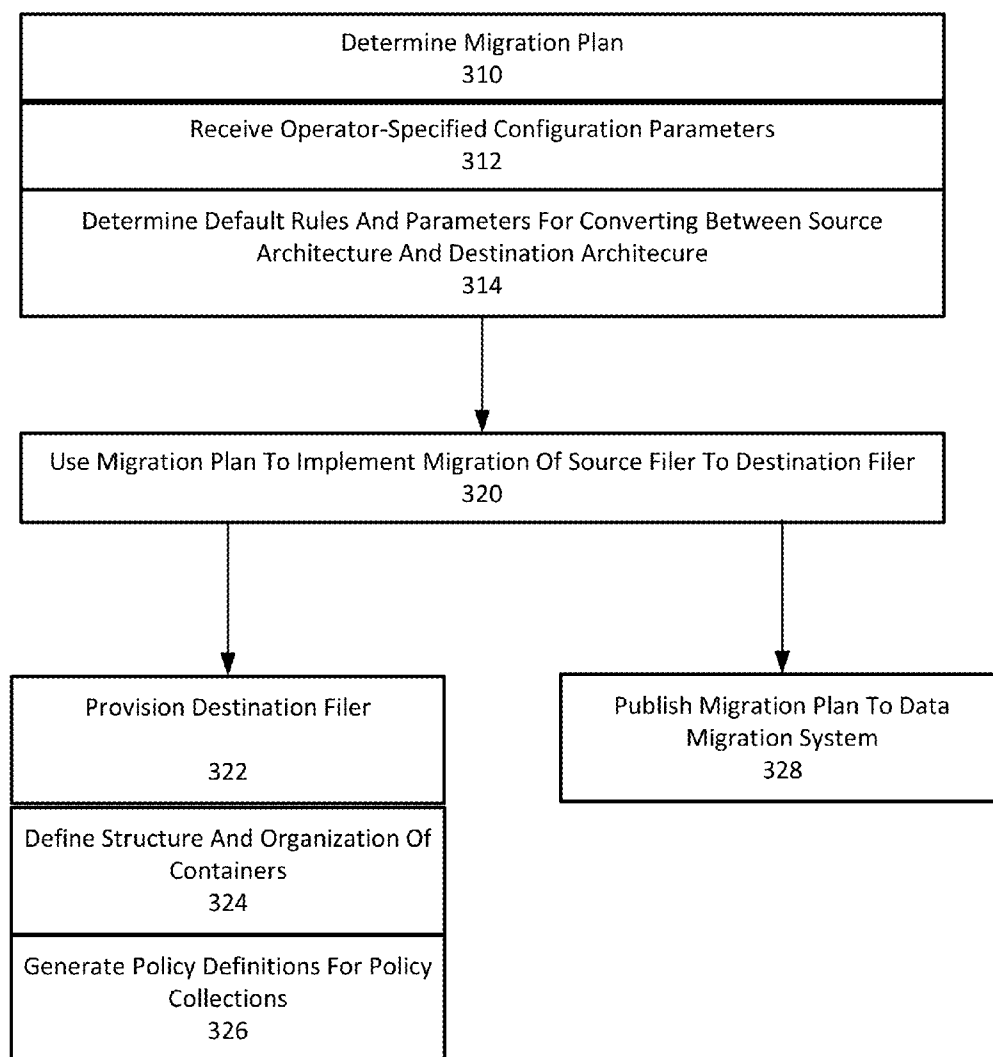


FIG. 2

**FIG. 3**

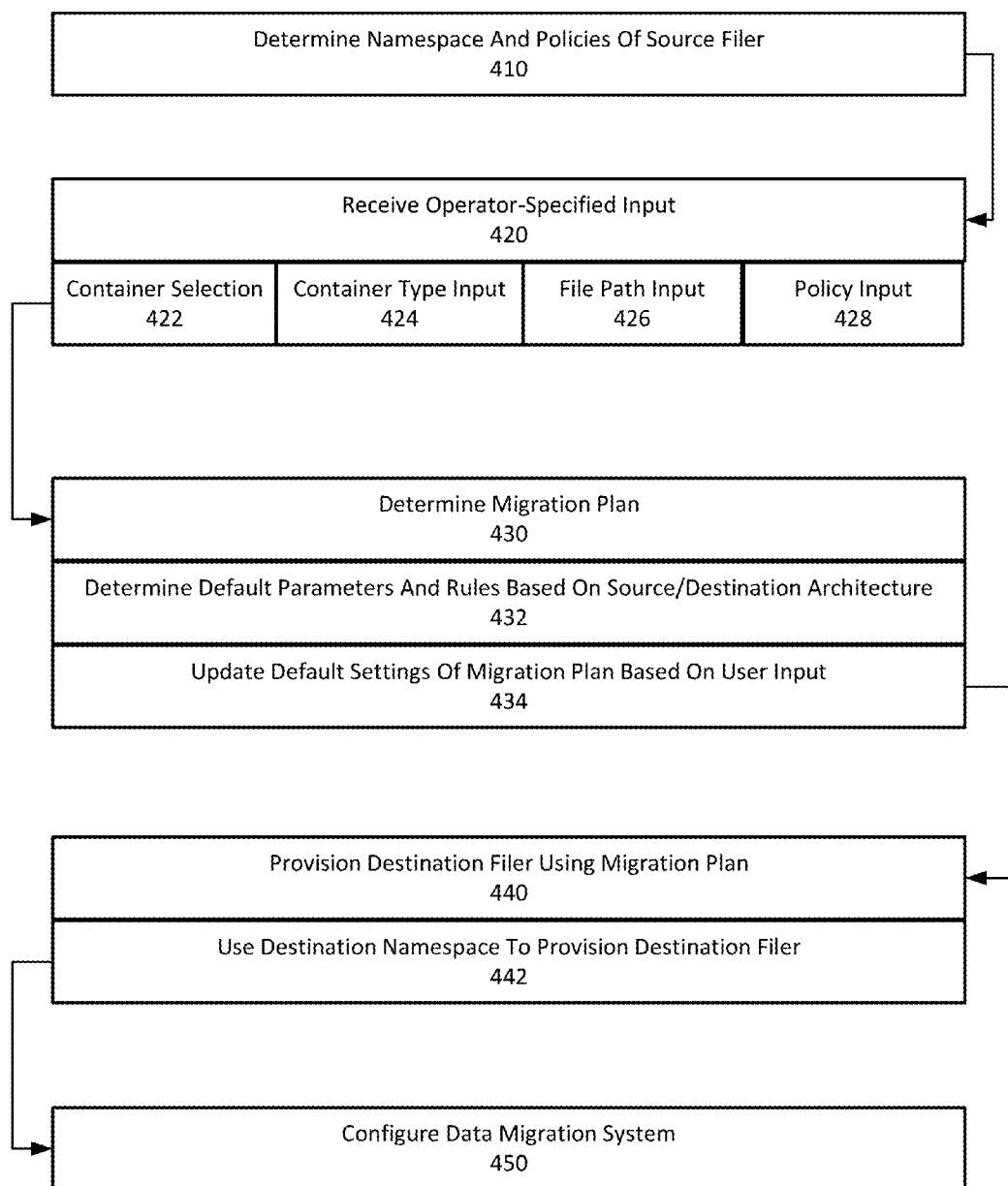


FIG. 4

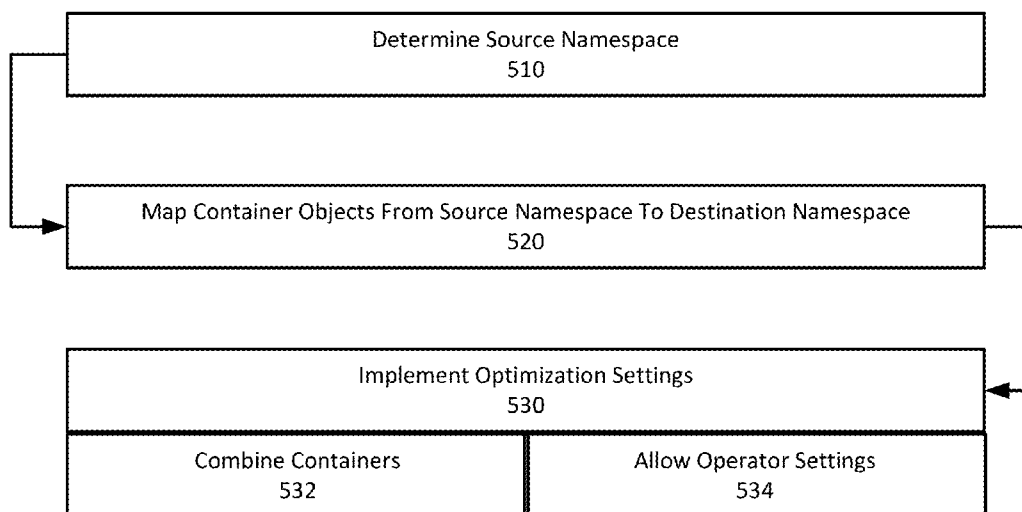


FIG. 5

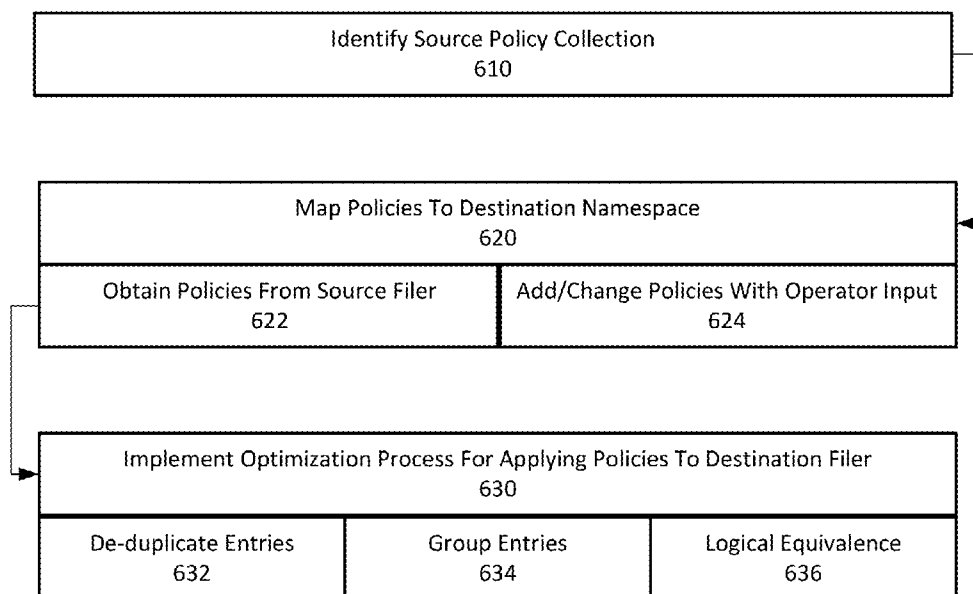


FIG. 6

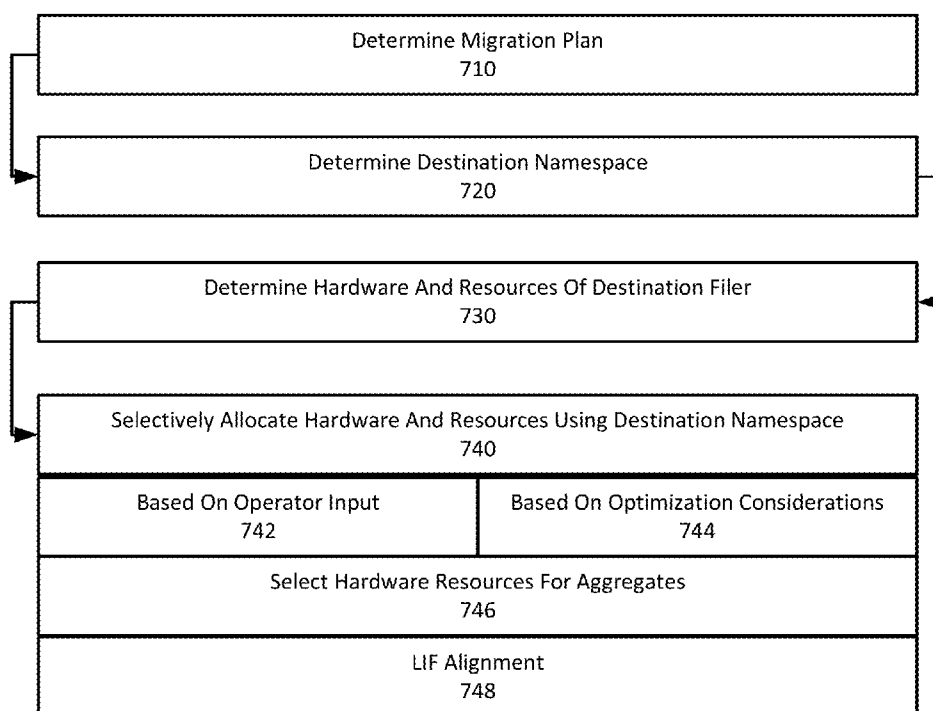


FIG. 7

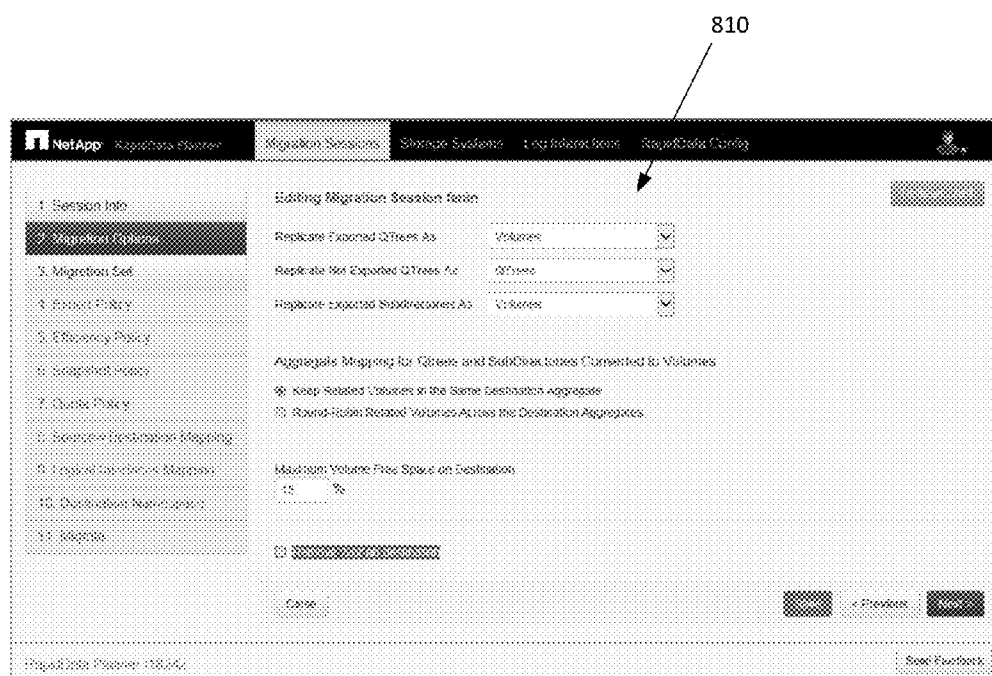


FIG. 8A

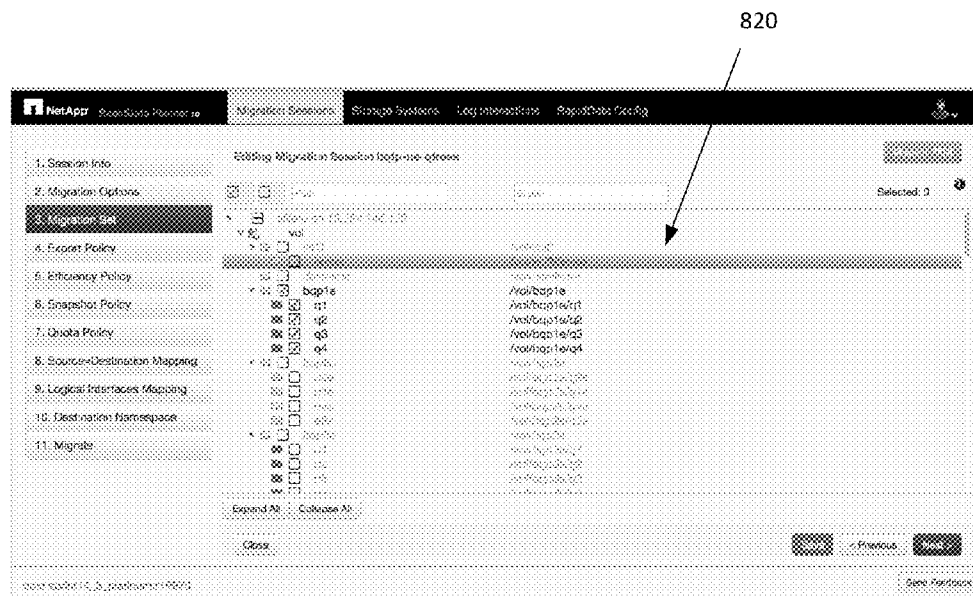


FIG. 8B

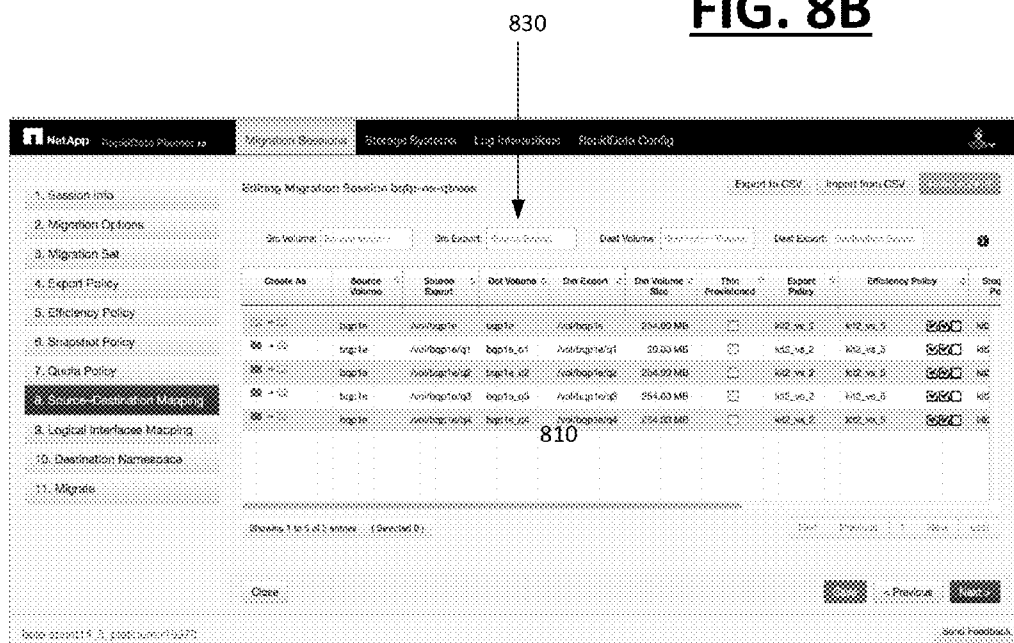


FIG. 8C

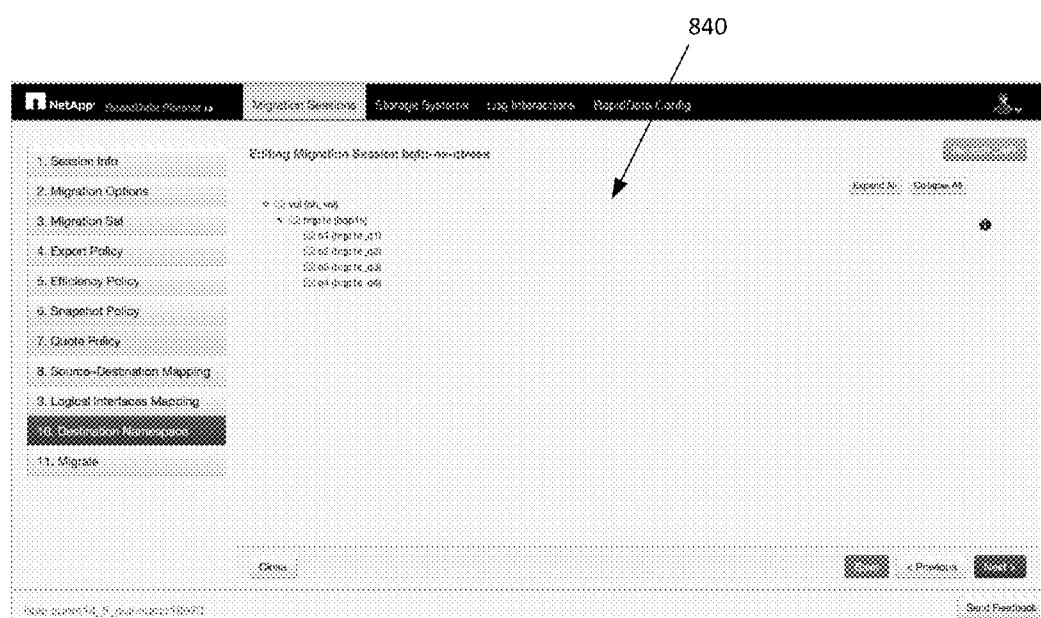


FIG. 8D

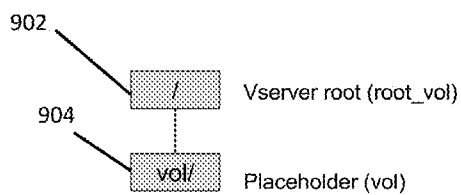


FIG. 9A

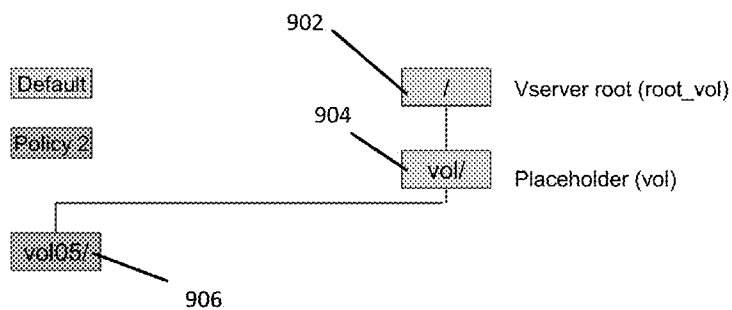


FIG. 9B

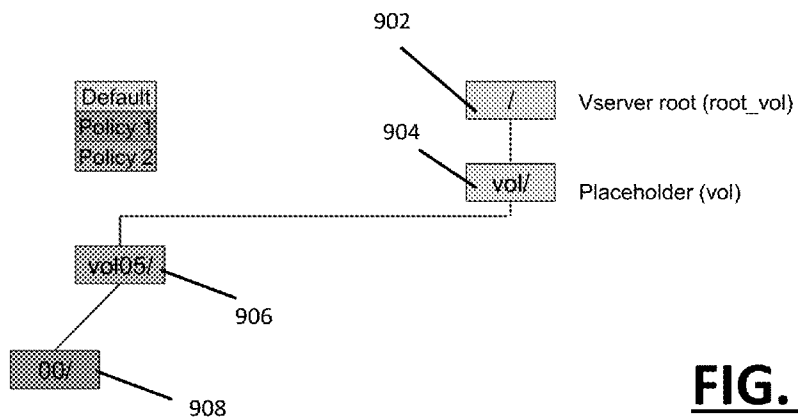


FIG. 9C

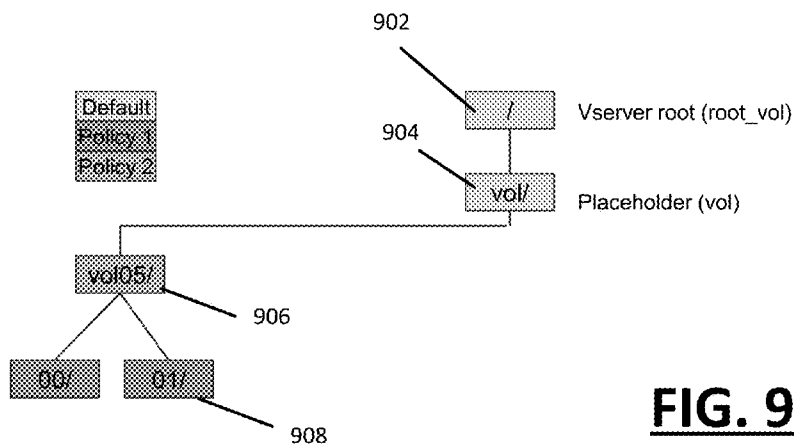


FIG. 9D

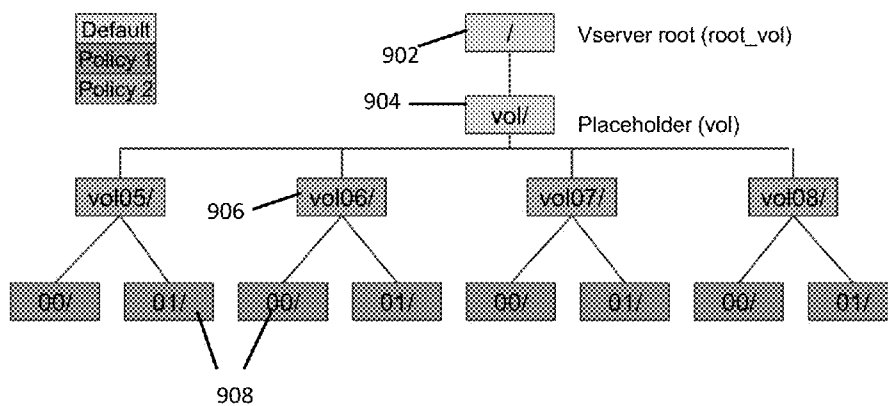
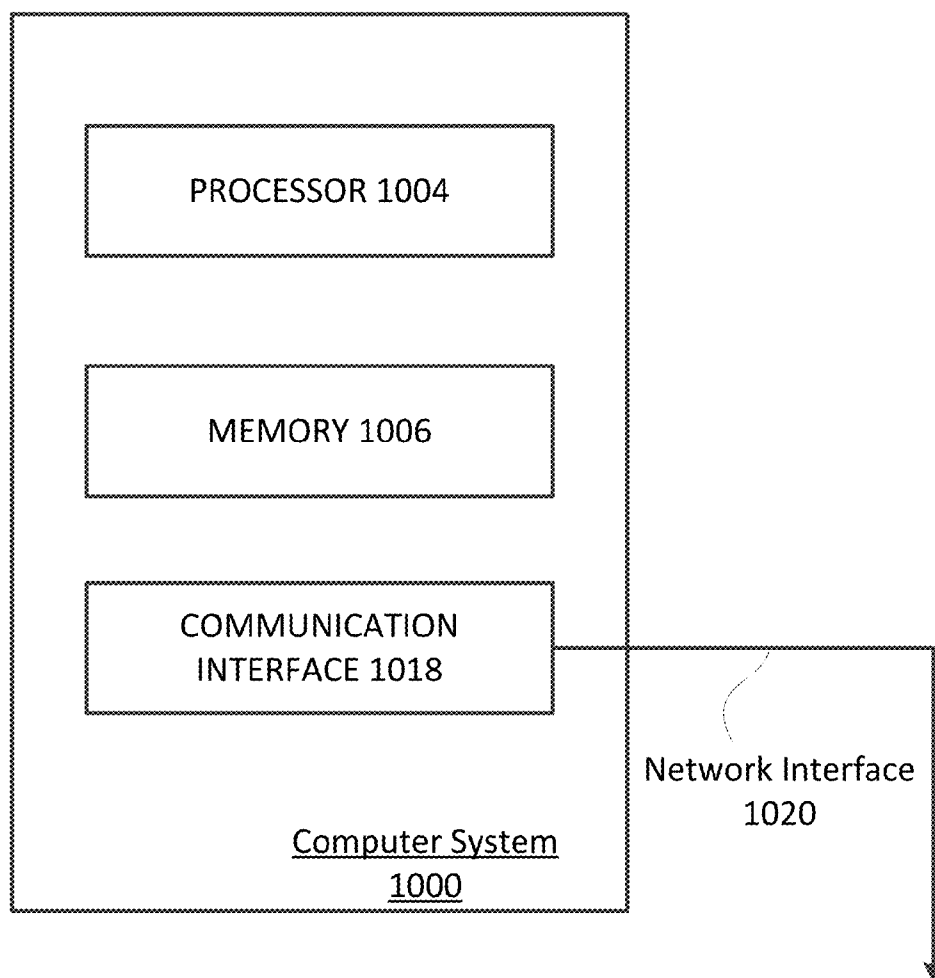


FIG. 9E

**FIG. 10**

SYSTEM AND METHOD FOR DEVELOPING AND IMPLEMENTING A MIGRATION PLAN FOR MIGRATING A FILE SYSTEM

TECHNICAL FIELD

[0001] Examples described herein relate to network-based file systems, and more specifically, to a system and method for planning and configuring migrations amongst file systems.

BACKGROUND

[0002] Network-based file systems include distributed file systems which use network protocols to regulate access to data. Network File System (NFS) protocol is one example of a protocol for regulating access to data stored with a network-based file system. The specification for the NFS protocol has had numerous iterations, with recent versions NFS version 3 (1995) (See e.g., RFC 1813) and version 4 (2000) (See e.g., RFC 3010). In general terms, the NFS protocol allows a user on a client terminal to access files over a network in a manner similar to how local files are accessed. The NFS protocol uses the Open Network Computing Remote Procedure Call (ONC RPC) to implement various file access operations over a network.

[0003] Other examples of remote file access protocols for use with network-based file systems include the Server Message Block (SMB), Apple Filing Protocol (AFP), and NetWare Core Protocol (NCP). Generally, such protocols support synchronous message-based communications amongst programmatic components.

[0004] Commercially available network-based file systems are implemented through software products, such as operating systems marketed under DATA ONTAP 7G, GX, or 8 (sometimes referred to as "7-MODE") and CLUSTERED DATA ONTAP (sometimes referred to as "cDOT"), which are manufactured by NETAPP INC. Other commercial examples of network based file systems include ISILON and VNX, which are manufactured by EMC.

[0005] With changes to business needs and development in technology, enterprise operators sometimes elect to change their network file system. For example, an operator may elect to do a technical refresh of hardware resources on a network file system, and in the process, elect to implement a different file system architecture for the updated hardware resources. Under conventional approaches, tools are available to assist an operator of a network file system in migrating to a network file system architecture. Many approaches require the source file system to incur some downtime as the clients are redirected to the new file system. Typically, file system migration amongst file systems of different architecture are very labor-intensive, as administrators are required to implement various tasks and processes in order to preserve the original namespace and the various policies (including export policies) in the new destination.

BRIEF DESCRIPTION OF THE DRAWINGS

[0006] FIG. 1 illustrates a system for implementing migration of a source file system to a destination file system using a migration plan, according to an embodiment.

[0007] FIG. 2 illustrates a migration planning system for use in migrating file system objects from a source filer to a destination filer, according to an embodiment.

[0008] FIG. 3 illustrates an example method for using a migration plan to migrate a file system onto a destination filer, according to an embodiment.

[0009] FIG. 4 illustrates an example method for developing a migration plan using operator input, according to an embodiment.

[0010] FIG. 5 illustrates an example method for implementing a selective migration of a source filer with alterations to how container objects are migrated, according to an embodiment.

[0011] FIG. 6 illustrates a method for mapping policies of a source filer to that of a destination filer, according to an embodiment.

[0012] FIG. 7 illustrates a method for allocating resources of a destination filer based on a migration plan, according to an embodiment.

[0013] FIG. 8A through 8D illustrate example interfaces that are generated for an operator to enter input for configuring or defining a migration plan, according to one or more embodiments.

[0014] FIG. 9A through FIG. 9E illustrate the provisioning of a destination filer in which changes are made to container types.

[0015] FIG. 10 is a block diagram that illustrates a computer system upon which embodiments described herein may be implemented.

DETAILED DESCRIPTION

[0016] Embodiments described herein provide for a computer system and method for creating a migration plan for migrating a source file system to a destination file system.

[0017] Among other uses, the migration plan can incorporate operator input for configuring the structure, organization and policy implementation of the destination file system. In some embodiments, a planner operates to generate a user interface for enabling operator input to specify structure, organization and policy input for file system objects that are to be migrated to the destination filer.

[0018] Still further, in some embodiments, the migration plan facilitates the migration of file system objects as between source and destination file systems that have different architectures. Furthermore, a migration plan can be created that enables the migration to be optimized for the architecture of the destination file system.

[0019] According to one aspect, a migration plan is created that is based at least in part on an operator input. The resources of a destination file system are provisioned based on the migration plan. One or more processes to migrate the source file system for the provisioned resources of the destination file system are then configured based on the migration plan.

[0020] In still another embodiment, a migration plan is determined for a file system migration. The migration plan maps each of (i) a set of file system objects from the source file system to a corresponding set of file system objects at the destination file system, and (ii) a source policy collection for the set of file system objects to a destination policy collection for the corresponding set of file system objects as the destination file system. Additionally, the migration plan includes a set of parameters that are based at least in part on an operator input. The migration of the file system objects can be implemented using the migration plan, with the migration plan specifying, for at least a first object container of the corresponding set of file system objects at the destination file

system, at least one of a file contextual path, type, or policy implementation relating to the first object container, that is different as compared to a file path, type or policy implementation of a corresponding source object container of the set of file system objects of the source file system.

[0021] In still another embodiment, a migration plan is determined for the file system migration. The migration plan includes a set of parameters and a set of rules, including one or more parameters or rules that are based on an operator input. A namespace is determined for the source file system. The source namespace identifies a set of file system objects and a file path for each of the file system objects in the set of file system objects. A collection of source policies that are implemented for file system objects identified in the namespace of the source file system is also determined. The migration plan is used to create a namespace for the destination file system based at least in part on the namespace of the source file system. Additionally, the migration policy is used to determine a destination collection of policies for implementation on the destination file system. The destination collection of destination policies can be based at least in part on the collection of source policies. At least one of the namespace or the collection of destination policies include an operator-specific configuration that is specified by the one or more rules or parameters of the operator input.

[0022] As used herein, the terms “programmable”, “programmatically” or variations thereof mean through execution of code, programming or other logic. A programmable action may be performed with software, firmware or hardware, and generally without user-intervention, albeit not necessarily automatically, as the action may be manually triggered.

[0023] The term “optimal” (or variants such as “optimized” or “optimum”) means through the use of intelligent considerations to make more optimal than otherwise without such considerations. Thus, the use of the term “optimize” or “optimized”, for example, in reference to a given process or data structure does not necessarily mean a process or structure that is the most optimal or the best. The basis for optimum can include performance, efficiency, effectiveness (e.g., elimination of redundancy) and/or cost.

[0024] An “architecture” of a network file system can be characterized by an operating system level component that structures the file system and implements the protocol(s) for operating the file system. Among other tasks, the operating software of a given architecture structures the namespace and the policies that affect the namespace. In the context provided, the term “structuring” and variants thereof refers to both a format and a logical structure.

[0025] One or more embodiments described herein may be implemented using programmatic elements, often referred to as modules or components, although other names may be used. Such programmatic elements may include a program, a subroutine, a portion of a program, or a software component or a hardware component capable of performing one or more stated tasks or functions. As used herein, a module or component can exist in a hardware component independently of other modules/components or a module/component can be a shared element or process of other modules/components, programs or machines. A module or component may reside on one machine, such as on a client or on a server, or may alternatively be distributed among multiple machines, such as on multiple clients or server machines. Any system described may be implemented in whole or in part on a server, or as part of a network service. Alternatively, a system such as

described herein may be implemented on a local computer or terminal, in whole or in part. In either case, implementation of a system may use memory, processors and network resources (including data ports and signal lines (optical, electrical etc.)), unless stated otherwise.

[0026] Furthermore, one or more embodiments described herein may be implemented through the use of instructions that are executable by one or more processors. These instructions may be carried on a non-transitory computer-readable medium. Machines shown in figures below provide examples of processing resources and non-transitory computer-readable mediums on which instructions for implementing one or more embodiments can be executed and/or carried. For example, a machine shown for one or more embodiments includes processor(s) and various forms of memory for holding data and instructions. Examples of computer-readable mediums include permanent memory storage devices, such as hard drives on personal computers or servers. Other examples of computer storage mediums include portable storage units, such as CD or DVD units, flash memory (such as carried on many cell phones and tablets) and magnetic memory. Computers, terminals, and network-enabled devices (e.g. portable devices such as cell phones) are all examples of machines and devices that use processors, memory, and instructions stored on computer-readable mediums.

[0027] System Overview

[0028] FIG. 1 illustrates a system for implementing migration of a source file system (“source filer”) to a destination file system (“destination filer”) using a migration plan, according to an embodiment. Among other benefits, an example of FIG. 1 enables a migration to be performed in which a resulting destination filer is optimized with respect to characteristics such as organizational structure, policy application, and resource allocation. In particular, an example of FIG. 1 enables migration of source filer to the destination filer when the source and destination filers operate under different architectures. According to an example of FIG. 1, programmatic components operate to automatically identify and optimize the manner in which file system objects are mapped from a source filer to a destination filer, so to account for semantic and logical characteristics of the destination filer. As an addition or alternative, an example of FIG. 1 enables an operator to further structure and configure the destination filer to implement changes based on need or preference in order to account for features of the destination filer and/or resources of the destination filer.

[0029] With further reference to FIG. 1, a system 10 includes a data migration system 100 and a planner 110. The data migration system 100 and planner 110 operate to migrate file system objects 101 from a source filer 20 to a destination filer 50. Among other functions, the planner 110 operates to (i) discover structure, organization and policies of the source filer 20 from which migration is to be initiated, (ii) provision the destination filer 50 for structure and organization, such as to type and location of object containers, and (iii) implement a policy collection 119 on the destination filer 50 that is based on the discovered policies of the source filer 20. Additionally, the planner 110 can publish a migration plan for use by the data migration system 100 to perform the migration as session-based tasks.

[0030] In some embodiments, the planner 110 provides a mechanism to enable operator input to (i) select portions of the source filer for migration, (ii) configure or after the provisioning of the destination filer, such as by way of changing

container types or file paths, and (iii) altering the destination's policy collection **119** relative to a policy collection **109** of the source filer **20**.

[0031] Still further, the planner **110** can include rules and other logic to optimize the destination filer **50**. In particular, the resources of the destination filer can be provisioned in a manner that optimizes implementation of the migrated file system to account for characteristics and functionality of the destination filer **50**, as well as preferences and needs of the operator. As described with some examples, the source filer **20** and destination filer **50** can optionally operate under different system architectures. In such implementations, the planner **110** can provision the destination filer **50** to, for example, change the type of select containers when the migration is performed in order to leverage additional configurability and functionality available to the particular container type on the architecture of the destination filer **50**, enable more operator configurability. The planner **110** can also define policies of a destination policy collection **119** in order to account for the architecture of the destination filer **50**.

[0032] In more detail, the planner **110** receives, as input, a namespace **107** and policy collection **109** of the source file system **20**. The namespace **107** can be determined from, for example, one or more programmatic resource that perform discovery on the source filer **20**. In the example of FIG. 1, the data migration system **100** can include one or more components that implement operations to discover file system objects, including interfacing with log files and other resources of the source file system **20** in order to determine the namespace **107**. In a variation, the planner **110** can execute processes to interface with export files and other policy stores or resources of the source filer **20** in order to determine the source policy collection **109**.

[0033] The planner **110** can also receive operator input **115** (e.g., via a user-interface), which identifies preferences of an operator (e.g., administrator). As described below, the operator input **115** can include input to (i) select containers or portions of the source filer **20** to migrate, (ii) specify a container type and/or organization for a select or global number of containers in the destination filer **50**, and/or (iii) specify policy configurations and changes for the file system when migrated onto the destination filer **50**. Still further, in some embodiments, the planner **110** can include a default set of rules and parameters **125**, which are selected or otherwise based on the architecture of the source filer **20** and the architecture of the destination filer **50**. The default set of rules and parameters **125** can determine how file system objects from the source filer **20** map to the destination filer **50**. The default set of rules and parameters **125** can include considerations for optimizing the destination architecture. In particular, the default set of rules and parameters **125** (i) determine the structure and organization of the migrated file system objects on the destination filer **50**, and (ii) define the policies of the destination policy collection **119** for application on the file system objects of the destination filer **50**. Accordingly, the default set of rules and parameters **125** can include logic for provisioning the destination filer **50** and for implementing the migration as between the source filer **20** and the destination filer **50** when the source and destination utilize different architectures.

[0034] In this way, planner **110** operates to generate the migration plan **112**, which in turn is used to provision the destination filer **50**. Once the migration is complete, the structure and organization of the file system objects in the desti-

nation filer **50** can be dictated by the manner in which the destination filer **50** was provisioned through use of the migration plan **112**.

[0035] As described with other examples, the policies of the destination policy collection **119** can be formatted and logically structured based on operator input **115** and/or optimization considerations of the destination filer **50**. The optimization considerations for the destination policy collection **119** can be based in part on optimization characteristics of the architecture of the destination filer **50**. Thus, the destination policy collection **119** can differ in format and structure from the corresponding policy collection **109** of the source filer **20**. By way of example, the policy collections **109**, **119** of the respective source and destination filers **20**, **50** can include policy sets for exports, quotas, storage efficiency and backup.

[0036] In operation, the planner **110** operates to generate the migration plan **112** based on the default set of rules and parameters **125** and/or the operator input **115**. In one implementation, the migration plan **112** is communicated in whole or in part to the data migration system **100**. The data migration system **100** uses the migration plan to implement the migration. In one implementation, the migration plan **112** is published for the data migration system **100**, and the data migration system replicates file system objects for inclusion in containers that are identified or specified by the migration plan. The data migration system **100** can operate on a session-basis that is defined in part by specific containers provisioned on the destination filer **50**. The sessions of the data migration system **100** can also be automated and/or sequenced, so that some or all of the migration can be performed in autonomous fashion once the migration plan **112** is published.

[0037] Among other tasks, the planner **110** and data migration system **100** combine to allocate aggregates of the source filer **20** as being migrated amongst hardware resources in accordance with default settings or preferences of the user. For example, the data migration system **100** can determine the logical interfaces (LIF) for accessing aggregates of the source file system **20** being migrated. The LIF determination can specify routes and resources for aggregates based on considerations such as expected traffic for aggregates and minimizing communication hops.

[0038] The data migration system **100** can replicate file system objects **101** of the source filer **20** as destination file system objects **121**. According to some embodiments, the data migration system **100** uses information from the migration plan **112** (e.g., destination namespace) in order to replicate file system objects for the containers of the destination filer **50**. In this way, the replication of the file system objects **101** results in the generation of destination file system objects **121** which are stored in containers of type and organization specified by the migration plan **112**.

[0039] In one embodiment, the object containers of the destination file system **50** can include volumes, quota trees ("q-tree") or directories. In one implementation, the object containers map in type and organization to that which is discovered from the source filer **20**, unless operator input **115** is recorded to alter the default set of rules and parameters **125** of the migration plan **112**. In one implementation, the operator-input can change both type and relative location of individual containers on the destination filer **50**, as compared to provisioning using the default set of rules and parameters **125**. Still further, in one implementation, the destination filer **50** is provisioned with the destination namespace, as provided by the migration plan **112**. Additionally, the data migration

system **100** uses the object containers of the destination filer **50** to replicate file system objects onto the destination filer.

[0040] Other organization aspects of file system objects **101** can also be determined on the destination file system by either default or by operator input. For example, absent operator input, the containers of the destination filer **50** can be structured and organized to preserve the granularity and organization of the source filer **20**. The organizational aspects of the containers in the destination filer **50** can be based on their respective file path, as provided on the source filer **20**. The destination filer **50** can be provisioned to generate the containers so that they include equivalent context file paths, meaning that the file paths provided with the container objects of the destination filer **50** include a common segment (which includes the leaf node of the file path) with the file path of the counterpart object container at the source filer **20**. The migration plan **112** can use operational input **115** or other input in order to alter the context file paths of the destination containers. For example, when the file path of an object container at the source filer **20** is shown to embed that object container within another container type, operator input **115** can specify that that the two containers occupy the same hierarchy level when provisioned on the destination filer **50**.

[0041] The data migration system **100** can also implement the policy collection specified by the migration plan **112** using for example, one or more controllers (shown as “controller **58**”) of the destination filer **50**. The controller **58** can represent one or multiple hardware and/or software components for controlling aspects of the destination filer **50**. For example, in an implementation in which the destination filer **50** is a cDOT architecture, the policy management may be conducted through use of a Storage Virtual Machine (or “SVM”). Accordingly, one implementation provides for the migration plan **112** to be communicated to the controller **58** of the destination filer **50** in order to create the policy rules and statements that comprise the policy collection **119**, specified by the migration plan **112** for the destination filer **50**. When the migration is performed by the data migration system **100**, the controller **58** applies the destination policy collection **119**.

[0042] FIG. 2 illustrates a migration planning system for use in migrating file system objects from a source filer to a destination filer, according to an embodiment. In particular, FIG. 2 provides example components for implementing planner **110**, as also described with an example of FIG. 1, the planner **110** includes a user interface **220**, a namespace component **230**, a policy component **240** and a device or resource planner **250**.

[0043] A walker **210** can operate as part of, or with the planner **110**, in order to determine source information **215**. The walker **210** can execute a process that interfaces with various resources of the source filer **20** in determining the source information stores **215**. In one implementation, the walker **210** can implement a REST interface to access log or health files, such as generated through “autosupport” (or “ASUP”) mechanisms of ONTAP-based network file systems. By way of example, the walker **210** can utilize log or health files to identify file system objects and structural (e.g. type) or organizational information (e.g., file paths) about the file system objects of the source filer **20**. As an addition or alternative, the walker **210** interfaces with policy servers and stores of the source file system **20** in order to determine the policies of the file system objects being migrated. By way of example, in one implementation, the walker **210** includes a process that executes through the planner **110** and executes a

Zephyr Application Program Interface (ZAPI) to locate and extract export files of the source filer **20**. The export files can identify policies that exist on the source filer for file system objects and portions (e.g., volumes) of the source filer **20**.

[0044] In one implementation, the source information stores **215** can include a source namespace **208**, an export policy store **212**, an efficiency policy store **214**, a snapshot policy store **216** and a quota policy store **218**. The namespace **208** identifies file system objects (e.g., by inode) and their respective type, as well as file path information for the objects. In one implementation, the information of the namespace is determined primarily from log health files of the source filer **20**. Information for the export policy store **212**, efficiency policy store **214**, snapshot policy store **216** and quota policy store **218** can be obtained from, for example, the walker **210** locating and obtaining export files and related data from controllers and policy resources of the source filer **20**.

[0045] In one implementation, the planner **110** includes a plan store **225** for aggregating parameters and data corresponding to a migration plan. Some of the parameters and data of the plan store **225** can be determined by default, based on, for example, the source and destination filers **20**, **50**. Other parameters and data of plan store **225** can be obtained from operator input **223**, provided through, for example, the user interface **220**. In one implementation, the user interface **220** utilizes source information **215**, such as information from the namespace store **208**, in order to generate prompts and contextual information for enabling operator input **223**. The operator input **223** can be stored as parameters in the plan store **225**. Among other applications, plan store **225** can be used to (i) generate a destination namespace, (ii) create and configure or modify file system objects (e.g., containers) identified for the destination filer **50**, (iii) create policy statements for application of a policy collection on file system objects at the destination filer **50** (as identified in the destination namespace), and/or (iv) provision the destination filer **50** and/or its resources for migration of file system objects from the source filer **20**.

[0046] In an embodiment, the namespace component **230** of the planner **110** receives source namespace information **231** from the namespace store **208**. Additionally, the namespace component **230** receives migration planning parameters **233** from the plan store **225**. The namespace component **230** can include programming and logic to enable establishment of a destination namespace **261** for the destination filer **50** using the source namespace information **231** (e.g., namespace entries). In one implementation, the destination namespace **261** is generated for a migration in which the destination filer **50** is of a different architecture than that of the source filer **20**.

[0047] In one embodiment, the namespace component **230** can include a format component **232**, container logic **234** and namespace builder **236**. The format component **232** implements formatting rules for reformatting namespace entries specified in the source namespace information **231** into namespace entries **235** that have a format of the destination architecture. In one implementation, an individual source namespace entry includes a file path and identifier in a first format (e.g., 7-MODE) of the source architecture, and the format component **232** restructures the file path and identifier into a second format of the destination architecture (shown by entries **235**). The reformatting of the file path and identifiers can include both syntax and semantic changes.

[0048] The container logic 234 includes rules and other logic for specifying a change to an object container. In particular, the container logic 234 can be configured by parameters to change container types of select object containers identified in the source namespace information 231. For example, a container as identified by the source namespace information 231 can be changed in type based on planning parameters 233, which can be determined by the operator input 223. In one implementation, operator-selected object containers specified in the source namespace information 231 can be converted from q-trees or directories to volumes. The parametric configuration of the container logic 234 can be set by default (e.g., container logic 234 keeps original container types) or by operator input 223. In one implementation, the operator input 223 can, for example, specify specific (e.g., container-specific) or global changes to container types (e.g., change some or all q-trees to volumes). Accordingly, in one implementation, parameters that specify conversion of select object containers are received by user input provided through the user interface 220. Alternatively, the parameters can correspond to a global selection that applies to object containers of a specific type in a particular source.

[0049] In one implementation, the namespace builder 238 receives (i) reformatted namespace entries 235 provided by the format component 232, and (ii) identification of object containers and type from the container logic 234. The reformatted namespace entries and object containers can be determined in part from the source namespace information 231, using the namespace migration parameters 233 as provided from the plan store 225. Based on the input, the namespace builder 238 can generate a destination namespace 261. The namespace builder 238 can also include logic to optimize the destination namespace 261 by structure or formatting.

[0050] The destination namespace 261 can be used to provision the destination filer 50. Additionally, the destination namespace 261 can configure the data migration system 100 in populating the containers of type and organization specified in the destination namespace 261.

[0051] According to one example, the policy component 240 of the planner 110 includes a format component 242, a logical component 244 and a policy mapper 246. The policy component 240 receives policy information 243 from the source information 215. Additionally, the policy component 240 receives planning parameters 241 from the plan store 225. The planning parameters 241 can be based on user input 223, received through the user interface 220. As an addition or variation, the planning parameters 241 can correspond to a default set of parameters.

[0052] The policy component 240 generates policy collections 263 of different types for the destination filer 50. For example, the policy collections 263 can include export policies, efficiency policies, snapshot policies and quota policies. The architecture of the destination filer 50 can require different formatting and logical structures for each kind of policy. For some types of policies, the format and logical structure as between the source and destination architectures may be the same, and for other policy types, the format and logical structure may be different.

[0053] The policy format component 242 operates to restructure and format policies 243 as provided from the source information 215. The policies 243 can be reformatted or structured for syntax and semantic structure. The logical component 244 can apply cluster or group operations on select kinds of policy collections in order to determine logi-

cally equivalent policy sets. In particular, the logical component 244 can include logic to combine, for example, equivalent or identical policies to reduce, thereby reducing the policy count for the destination filer 50. For example, a cluster of export policy statements can be represented by a single policy statement in the corresponding policy collection 263. The policy mapper 246 can store the policy collections 263 on the destination filer 50 when, for example, the destination filer 50 is provisioned.

[0054] The resource planner 250 can plan for hardware and resource allocation of the destination filer 50, given resource information 253 and operator-specified allocation parameters 251. The resource information 255 can include usage information, which can be determined or estimated from, for example, the operator for aggregates of the file system being migrated. For example, the resource information 255 can be determined from health logs of the source filer 20 or from manual input (e.g., operator identifies aggregates of the source filer with the most traffic or least traffic). The allocation parameters 251 can include specified by the operator via the user interface 220 (operator input 223). As an addition or alternative, the allocation parameters 251 can include default parameters or rules which can be provided by the plan store 225.

[0055] The resource planner 250 can generate resource allocation data 265 (e.g., instructions, parameters) to allocate and configure hardware and logical resources on the destination filer 50. The allocation and configuration of hardware resources can be to optimize performance and balance loads. In one implementation, the resource allocation data 265 assign specific objects to aggregates of the file system being migrated based on expected traffic. For example, those objects on the source filer 20 with the most traffic can be assigned to high-performance resources when migrated to the destination filer 50, while other objects that had history of less demand receive low cost resources on the destination filer 50.

[0056] As an addition or alternative, the allocation instructions 265 can also generate a topology that identifies the location of aggregates on nodes of the destination filer 50. The topology can specify hardware driven paths to aggregates of the destination filer, which can be determined by the presence of logical interfaces (LIFs) amongst the nodes of the destination architecture. Specifically, each LIF on the destination filer 50 provides an address (e.g., IP address) to a physical port, and the allocation instructions 265 can select paths amongst physical ports of the destination filer through LIF selection. By way of example, the network paths can be shortened (e.g., fewer hops), for file system objects and containers that are most heavily used.

[0057] As shown by an example of FIG. 2, the planner 110 operates to provision and ready the destination filer 50, and to configure the data migration system 100. As a result, the destination filer 50 is intelligently configured when put in use after migration, with minimal effort from the operator.

[0058] With respect to examples such as described with FIG. 1 and FIG. 2, the data migration system 100 can operate asynchronously and seamlessly, so as to support multiple clients what actively use file system objects being migrated while those file system objects are replicated from the destination filer 50. According to some embodiments, the data migration system 100 can be implemented in accordance with migration systems such as described with U.S. patent application Ser. Nos. 14/011,699, 14/011,696, 14/011,719, 14/011,718 and 14/011,723; all of the aforementioned appli-

cations being owned by the assignee of this application and further being incorporated by reference in its entirety.

[0059] Methodology

[0060] FIG. 3 illustrates an example method for using a migration plan to migrate a file system onto a destination filer, according to an embodiment. FIG. 4 illustrates an example method for developing a migration plan using operator input, according to an embodiment. FIG. 5 illustrates an example method for implementing a selective migration of a source filer with alterations to how container objects are migrated, according to an embodiment. FIG. 6 illustrates a method for mapping policies of a source filer to that of a destination filer, according to an embodiment. FIG. 7 illustrates a method for allocating resources of a destination filer based on a migration plan, according to an embodiment. Methods such as described with examples of FIG. 2 or FIG. 3 can be implemented using, for example, a system such as described with an example of FIG. 1 or with FIG. 2. Accordingly, examples described by FIG. 3 through FIG. 7 can be implemented in context such as described with FIG. 1 or FIG. 2, including an example in which a source and destination filer operate under different architectures. In describing examples of FIG. 2 through FIG. 7, reference may be made to examples of FIG. 1 or FIG. 2 for purpose of illustrating a suitable component for performing a step or sub-step being described.

[0061] With reference to FIG. 3, a migration plan is determined (310). The planner 110 can generate the migration plan 112 using a combination of default rules and logic, as well as operator-specified input. The migration plan 112 can also be specific to the architecture type of the source and destination filers 20, 50, to enable the migration to occur between architectures of different types.

[0062] In one implementation, the planner 110 includes a user interface 220 that prompts for and receives operator input (312). FIG. 8A through FIG. 8D illustrate example of a user interface that can enable the operator to specify planner input that is object specific or global. As an addition or alternative, the planner 110 can utilize default rules and parameters for the migration plan 112, which can be specific to the destination filer 50 and/or the conversions needed as between the source and destination architectures (314). The operator input can change the default rules and parameters.

[0063] The migration plan 112 can then be used to implement the migration from the source filer 20 to the destination filer 50 (320). In one implementation, the migration plan 112 is used to provision the destination filer 50, and the data migration system 100 populates the destination filer 50 based on the provisioning (322). In provisioning the destination filer 50, the migration plan can define structures and organization for containers at the destination filer (324). In one implementation, the type of container can be changed (e.g., from q-tree to volume or directory). Still further, the context portion of file path of the containers at the destination filer 50 can be modified based on the settings of the migration plan 112. Thus, for example, an operator can promote a q-tree to a volume, then move the promoted volume relative to other containers.

[0064] Additionally, the migration plan can generate policy definitions for policy collections of the destination filer (326). For example, the policy component 240 can generate policy collections 263 which provides for intelligently grouping or clustering policy entries of the source filer into policy definitions of the destination filer 50.

[0065] Additionally, the migration plan 112 can be published to the data migration system 100, so as to configure operations of the data migration system (328). With the migration plan 112, the data migration system 100 can create tasks which serve to populate containers of the destination namespace (as provided with the migration plan 112) with file system objects of the source filer 20. The implementation of the task approach enables the data migration system 100 to access containers that have been modified by type and path when replicating the file system objects of those containers. Additionally, the migration plan enables the data migration system 100 to automate, time and/or sequence (including implement in parallel or back-to-back) initiation of tasks.

[0066] With reference to FIG. 4, one or more discovery processes are implemented to discover a namespace of the source filer and the policy collections of the source namespace (410). As described with an example of FIG. 2, the planner 110 can implement one or more walker processes 210 that can be utilized in the discovery process. The discovery process can identify the source namespace and the policy collections that are implemented on the source namespace.

[0067] The planner can provide a user interface for the operator to specify input that configures the migration plan 112 (420). In one implementation, information from the source namespace is used to prompt the operator for intelligent input that configures the migration plan 112. For example, as shown with examples of FIG. 8A through FIG. 8D, a user interface is generated to enable the user to provide input for creating and configuring the migration plan 112. The user interface can be generated from the source namespace, which can be discovered through one or more walker processes 210. The user interface can display container objects (e.g., volumes, q-trees and directories) from the information of the source namespace.

[0068] One embodiment enables the user to select through the user interface a specific container from the source namespace (422). The selection can correspond to a juncture within the namespace, and serves to make the migration of the source filer selective. With the selection, the migration may be limited to file system objects that are embedded within the selected container. In a variation, the user can provide input through the interface to enable the operator to provide exclusion input, which identifies file system objects that the operator does not want to migrate.

[0069] In another implementation, the operator can provide input that specifies container input, and specifically input to change an existing container type to a different container type (424). For example, in cDOT, there is limited ability to specify independent policies for embedded q-trees. Should the user wish to specify a policy for the q-tree container, he can use the user interface to provide input that promotes the q-tree to a volume. As a volume, the cDOT architecture permits specification of, for example, select policies.

[0070] Still further, the operator can specify input that changes the path of a given object container or other file system object when migration is performed (426). In one implementation, the user interface can generate an organization that reflects the organization and structure of the source namespace. Absent input from the user, the structure and organization of the source namespace is replicated for the destination namespace, meaning the object containers identified in the destination namespace include file path segments (context file paths) which are the same as corresponding containers of the source namespace. With input, the file paths

of containers or other objects in the destination namespace **261** can be modified as compared to file paths that would otherwise be used with the default settings (e.g., to create one-to-one mapping with the source namespace). For example, in the preceding example where the container type is modified, the operator can elect to un-embed a volume that has been converted from a q-tree. The corresponding portion of the destination namespace **261** (i.e., context file path) can be modified based on the input.

[0071] Additionally, the user interface enables the operator to specify policy changes and additions for portions of the destination namespace (**428**). As shown with examples of FIG. 8A through 8D, policies can be displayed in interfaces that reflect existing source policy collections, and the interface can include rules for enabling the user to specify or configure policies differently for the destination namespace. For example, in the source filer running 7-MODE, an embedded q-tree can be associated with the security policy of the volume in its junction. For the destination running cDOT, the operator can provide input to promote the q-tree and further a new security policy that is specific for the promoted volume.

[0072] With operator input, the migration plan is determined (**430**). At least a portion of the migration plan **112** can be based on default parameters (**432**). In one implementation, for example, the default parameters can select to migrate file system objects with the same structure, organization, and granularity. Additionally, the policies can be replicated from the source filer **20** to the destination filer **50** with the default parameters set to achieve the same granularity (e.g., 1:1 mapping), in terms of logical equivalence, as that provided with the source filer **20**.

[0073] The operator input can be received to after the default parameters, thereby enabling the migration plan **112** to be configured per preferences and need of the operator (**434**). The alteration to the migration plan **112** can select junctures or containers for migration, specify alternative container types and paths for migrated containers. Additionally, the alteration to the migration plan **112** can select or modify policies from the default setting. In particular, the operator can specify container-specific policies with ability to change container types and position in order to achieve desired policies for select containers of the file system.

[0074] The migration plan **112** can be used to provision the destination filer **50** and configure the data migration system **100** (**440**). In one implementation, a destination namespace **261** is generated and used for provisioning the destination filer **50** (**442**). The destination namespace can define the structure and organization of the destination filer **50**. Additionally, as described in greater detail with an example of FIG. 6, policy collections can be implemented on the policy server and resources of the destination filer **50** based on default settings and/or operator-specified parameters

[0075] Additionally, the data migration system **100** can be configured to execute sessions where junctures or other portions of the source filer **20** are replicated on the destination filer **50** to populate the corresponding containers (which may have been modified by file path or type) (**450**). Additionally the data migration system **100** can be configured by the migration plan to sequence and queue the sessions automatically so that the destination filer is populated by junctures over time.

[0076] With reference to FIG. 5, the source namespace is discovered (**510**). For example, one or more walker processes

210 can discover the containers and junctures of the source filer **20**, from which the remainder of the source namespace is determined.

[0077] The migration plan **112** can be created in part by mapping the containers of the source namespace to a destination namespace (**520**) that is under development. The planner **110** can create the migration plan with default settings that create the destination namespace using the same or equivalent container structure, organization and granularity as present with the source namespace. For example, the planner **110** can implement the migration with default settings that by default, substantially replicates the source namespace as the default namespace.

[0078] However, examples recognize that operators typically desire to upgrade or change the technological resources of the filer, in which case the source namespace is likely not optimal for the destination filer. In one implementation, the planner **110** can generate the migration plan **112** to include default settings that optimize the destination namespace (**530**). For example, containers on the source namespace can be combined or consolidated (e.g., multiple q-trees can be consolidated into one volume) (**532**). Additionally, the planner **110** can include operator settings to enable greater adjustments to the migration plan **112**, such as changes to container types and file paths (**534**).

[0079] With reference to FIG. 6, the collection of policies on the source filer **20** can be identified in the context of the source namespace (**610**). As described with an example of FIG. 2, one or more walker processes **210** can be implemented to determine various kinds of policies for the source namespace, including export policy, efficiency policy, snapshot policy and quota policy, as well as other policies (e.g., security).

[0080] The policy component **240** of the planner **110** can map the collection of policies to the destination namespace using default rules and/or operator-specified parameters (**620**). For example, under default, the collection of policies for the source namespace can be formatted and converted into logically equivalent policy entries, which are subsequently implemented on the destination namespace. The default parameters can be set to achieve the same granularity (e.g., 1:1 mapping), in terms of logical equivalence, as that provided with the source filer **20**. The operator input can permit configurations, selections and other changes to the policy collection, as provided by other examples described herein.

[0081] Once mapped, the policy component **240** can implement one or more optimization processes in order to configure the manner in which policies are stated and implemented, given, for example, the architecture of the destination filer **50** (**630**). As shown by examples provided below, the optimization can include de-duplicating policy entries present for the source filer **20** in order to create an equivalent set of policy statements (**632**). The de-duplication can be based on the architecture of the destination filer, which can differ by format, structure and syntax as to how policy entries are applied to defined file system objects. Likewise, some types of policy entries from the source namespace can be grouped as a result of the destination architecture permitting (or requiring) logically different structures to the policy entries (**634**). Still further, the policy entries of the source filer can be individually or by group restated using policy entries of the destination architect which are logically equivalent (**636**).

[0082] The following examples illustrate examples of how policy entries for a destination collection can be reformatted

or structured by the planner 110 in order to optimize a corresponding subset of policy collections.

[0083] The following example illustrates two export policy entries for q-trees in 7-MODE, in which the same policy is applied to two different q-trees.

```
/vol/vol01/p17d4 -
sec=sys,rw=.mobstor.sp2.yahoo.com:ymdb.sp2.yahoo.com,anon=0
/vol/vol01/p17d5 -
sec=sys,rw=.mobstor.sp2.yahoo.com:ymdb.sp2.yahoo.com,anon=0
```

[0084] A policy definition in cDOT consists of defining the policy (first line in following paragraph) and then adding in the access rules (last 2 lines). Accordingly, when converting to cDOT, the policy component 240 of the planner 110 can recognize the application of identical policies to different containers, and further carry application of the policy to corresponding volumes (if the q-trees are promoted). This results in the creation of one policy on the destination that applies to both volumes.

```
export-policy create -policyname secpol-1 -vserver ausvs
export-policy rule create -policyname secpol-1 -clientmatch
.mobstor.sp2.yahoo.com -rorule sys -rwrule sys -anon pcuser -vserver
ausvs
export-policy rule create -policyname secpol-1 -clientmatch
.ymdb.sp2.yahoo.com -rorule sys -rwrule sys -anon pcuser -vserver ausvs
```

[0085] An example of another type of policy collection is snapshot policies. For the source filer 20 implemented under 7-MODE, policy entries for snapshot scheduling are defined per volume. Conversely, in cDOT, a single policy entry for snapshot scheduling can be applied to all volumes that are to implement the policy, and the policy entry can carry additional information for implementing the policy to the volumes. The following example illustrates how planner 110 can generate the destination policy collection for such policies with logical equivalences that are more optimal for the destination architecture:

[0086] 7-MODE snap sched vol0

[0087] cDOT Volume vol0: 0 2 6 volume snapshot policy
create—vserver ausys—policy sspol—enabled true—
schedule1 hourly—count1 6—schedule2 daily—count2
2—schedule3 weekly—count3 0

[0088] Other snapshot policies as between 7-MODE and cDOT can differ by syntax, rather than application. For example, a snapshot reserve policy can differ by syntax but not logic.

[0089] 7-MODE snap reserve vm_images

[0090] Volume vm_images: current snapshot reserve
is 20% or 41943040 k-bytes.

[0091] cDOT volume modify—vserver ausys—volume
vol01—percent-snapshot—space 20

[0092] Storage efficiency policies are generally governed by a policy that is applied to a volume. In 7-MODE, for example, a storage efficiency policy can include a deduplication process that is applied to a volume on a scheduled basis. In order to replicate such a policy in cDOT, the policy component 240 can specify a job that performs the efficiency task on a schedule, and then assign the job to a particular volume.

[0093] Quota policy as between 7-MODE and cDOT provides an example in which the mapping onto the destination filer 50 require policy generation and considerations which

are not present for the source filer 20. In 7-MODE, for example, a new default user quota is derived on a per volume transaction. Conversely, in cDOT, the quota policy is a collection of quota rules for all volumes of a virtual server. For example, the SVM can have 5 policies, of which 1 is active, and the activation of the quota policy is on a volume-by-volume basis. Accordingly, the policy component 240 can generate the policy collection for quotas on the destination filer 50 by (i) generating additional policies, (ii) specifying values for the generated policies, and (iii) selectively activating a policy. The steps of generating quota policy for the destination filer in this manner can be performed programmatically and/or with user input (such as activation parameter or quota value).

[0094] With reference to FIG. 7, a migration plan 112 is determined (710), and the migration plan is used to determine the destination namespace (720). Additionally, the resources of the destination filer 50 can be determined through, for example, operator input (730). The identified resources can include the tier or level of hardware components for implementing the storage environment, as well as the arrangement of logical interfaces (LIFs) for physical ports that interconnect the hardware resources.

[0095] The destination namespace can be a basis for allocating the identified resources for implementing the destination filer (740). The resource selection can be based on operator input (742), and/or based on optimization parameters (744). In one implementation, the operator provides input that identifies the aggregates that are to have, for example, the best or lowest performance, based on expected traffic or usage (746). Alternatively, the determination can be made programmatically by resource planner 250, which can access, for example, health logs or usage data in the source information 215. In similar fashion, those volumes or junctures that have highest use can be interconnected across resources with relatively shortened paths based on LIF alignment (748). For example, an active portion of a namespace can be located on hardware resources that are co-located or separated by a single LIF distance.

[0096] User Interface

[0097] FIG. 8A through 8D illustrate example interfaces that are generated for an operator to enter input for configuring or defining a migration plan, according to one or more embodiments. The interfaces provided with examples of FIG. 8A through FIG. 8D can be generated by, for example, the user interface 220 of the planner 110 (as shown in FIG. 2). The interfaces of FIG. 8A through FIG. 8D can be created using the source namespace, which can be discovered using walker processes 210.

[0098] FIG. 8A illustrates an example interface 810 that enables the operator to specify global parameters for configuring the migration plan. The global parameters can, for example, enable the operator to globally specify conversions of containers by type, and also to specify which aggregate promoted volumes should occupy (e.g., same). The example assumes that, for example, a q-tree that is being promoted to a volume may need a different aggregate as it is likely to become more significant. A pull down menu can be generated which identifies the conversions possible as between the source and destination architecture.

[0099] FIG. 8B illustrate an example interface 820 that enable an operator to select the volumes or portions of the source filer 20 that is to be migrated. The interface 820 can be generated in part based on the source namespace, so as to

identify actual containers of the source namespace, including the organization of the containers on the source filer **20**. Thus, the interfaces of FIG. **8B** enables the operator to specify selective migration, based on knowledge of the source namespace. According to some variations, the planner **110** can also incorporate migration of multiple source filers into one destination filer, and an example interface of FIG. **8B** enables the operator to specify portions of multiple sources to aggregate.

[0100] FIG. **8C** illustrates an interface **830** that illustrates how objects of the source namespace map to objects of the destination namespace, based on global policies (which can be selected or set by default). FIG. **8D** illustrates an interface for enabling the user to view a destination namespace. Based on the interfaces **830**, **840**, the user can select policies and/or provide input parameters.

EXAMPLES

[0101] FIG. **9A** through FIG. **9E** is a representative flow for the provisioning of a destination filer in which changes are made to container types. In a specific example provided, a 7-MODE source namespace is identified which includes four volumes under a root volume **902**, and each volume includes two embedded q-trees (not shown). The example assumes that the destination namespace is for cDOT implementation that promotes the q-trees to volumes. In FIG. **9A**, the provisioning of the destination filer **50** includes generating a placeholder volume **904** under the root volume **902**, which in FIG. **9A** can be implemented using the destination namespace entry:

```
[0102] volume create—volume vol—aggregate n02_
      sata01—vserver ausvs—junction-path/vol
```

[0103] In FIG. **9B**, the volumes **906**, which in the source namespace were previously mounted under the root node **902**, are mounted under the placeholder volume using the following namespace entries:

```
volume create vol05 -aggregate n02_sata01 -size 100GB -vserver
ausvs -autosize
true -junction-path /vol/vol05 -policy vol-pol2
volume create vol06 -aggregate n02_sata01 -size 100GB -vserver
ausvs -autosize
true -junction-path /vol/vol06 -policy vol-pol2
volume create vol07 -aggregate n02_sata01 -size 100GB -vserver
ausvs -autosize
true -junction-path /vol/vol07 -policy vol-pol2
volume create vol08 -aggregate n02_sata01 -size 100GB -vserver
ausvs -autosize
true -junction-path /vol/vol08 -policy vol-pol2
```

[0104] In FIG. **9C** and FIG. **9D**, the remaining q-trees present in the source namespace are now mounted within the placeholder volume structure as follows (note that only the first two q-trees are shown for brevity):

```
volume create vol05-00 -aggregate n02_sata01 -size 100GB -vserver
ausvs -autosize true -junction-path /vol/vol05/00 -policy vol-pol
volume create vol05-01 -aggregate n02_sata01 -size 100GB -vserver
ausvs -autosize true -junction-path /vol/vol05/01 -policy vol-pol
```

[0105] FIG. **9E** shows the result. Where in the source namespace, 4 volumes includes 8 embedded q-trees, now in the destination namespace q-trees are modified to volumes **908**, with the addition of the placeholder volume. Thus, the

example provided converts the source namespace with 12 containers (4 volumes and 8 q-trees) into a destination namespace with 13 containers (all volumes).

[0106] Computer System

[0107] FIG. **10** is a block diagram that illustrates a computer system upon which embodiments described herein may be implemented. For example, in the context of FIG. **1** or FIG. **2**, planner **110** may be implemented using one or more computer systems such as described by FIG. **10**. Still further, methods such as described with FIG. **3** through FIG. **7** can be implemented using a computer such as described with an example of FIG. **10**.

[0108] In an embodiment, computer system **1000** includes processor **1004**, memory **1006** (including non-transitory memory), and a communication interface **1018**. Computer system **1000** includes at least one processor **1004** for processing information. Computer system **1000** also includes a memory **1006**, such as a random access memory (RAM) or other dynamic storage device, for storing information and instructions to be executed by processor **1004**. The memory **1006** also may be used for storing temporary variables or other intermediate information during execution of instructions to be executed by processor **1004**. Computer system **1000** may also include a read only memory (ROM) or other static storage device for storing static information and instructions for processor **1004**. A storage device **1010**, such as a magnetic disk or optical disk, is provided for storing information and instructions. The communication interface **1018** may enable the computer system **1000** to communicate with one or more networks through use of the network link **1020** (wireless or wireline).

[0109] In one implementation, memory **1006** may store instructions for implementing functionality such as described with an example of FIG. **1** or FIG. **2**, or implemented through an example method such as described with FIG. **3** through FIG. **7**. Likewise, the processor **1004** may execute the instructions in providing functionality as described with FIG. **1** or FIG. **2**, or performing operations as described with an example method of FIG. **3** through FIG. **7**.

[0110] Embodiments described herein are related to the use of computer system **1000** for implementing the techniques described herein. According to one embodiment, those techniques are performed by computer system **1000** in response to processor **1004** executing one or more sequences of one or more instructions contained in the memory **1006**. Such instructions may be read into memory **1006** from another machine-readable medium, such as storage device **1010**. Execution of the sequences of instructions contained in memory **1006** causes processor **1004** to perform the process steps described herein. In alternative embodiments, hard-wired circuitry may be used in place of or in combination with software instructions to implement embodiments described herein. Thus, embodiments described are not limited to any specific combination of hardware circuitry and software.

[0111] Although illustrative embodiments have been described in detail herein with reference to the accompanying drawings, variations to specific embodiments and details are encompassed by this disclosure. It is intended that the scope of embodiments described herein be defined by claims and their equivalents. Furthermore, it is contemplated that a particular feature described, either individually or as part of an embodiment, can be combined with other individually described features, or parts of other embodiments. Thus,

absence of describing combinations should not preclude the inventor(s) from claiming rights to such combinations.

What is claimed is:

1. A method for implementing a file system migration as between a source file system and a destination file system, the method being implemented by one or more processors and comprising:

determining a migration plan for the file system migration, the migration plan mapping each of (i) a set of file system objects from the source file system to a corresponding set of file system objects at the destination file system, and (ii) a source policy collection for the set of file system objects to a destination policy collection for the corresponding set of file system objects as the destination file system, wherein the migration plan includes a set of parameters that are based at least in part on an operator input; and

implementing the migration of the file system objects using the migration plan, the migration plan specifying, for at least a first object container of the corresponding set of file system objects at the destination file system, at least one of a file contextual path, type, or policy implementation relating to the first object container, that is different as compared to a file path, type or policy implementation of a corresponding source object container of the set of file system objects of the source file system.

2. The method of claim 1, further comprising providing a user interface to receive the operator input, the user interface including a plurality of feature sets, the user interface including at least a type feature set to enable the operator to specify a parameter for implementing a first conversion rule to migrate one or more source object containers of a first container type as a corresponding one or more destination object containers of a second container type on the destination file system.

3. The method of claim 2, wherein the parameter is a global parameter that implements the first conversion rule to migrate all source object containers of the first container type in at least a given portion of the source file system as corresponding destination file system objects of the second type on the destination file system.

4. The method of claim 2, wherein the parameter is a selective parameter that implements the first conversion rule on an input specified set of source object containers of the first type in at least a given portion of the source file system as corresponding destination file system objects of the second type on the destination file system.

5. The method of claim 1, further comprising providing a user interface to receive the operator input, the user interface including a plurality of feature sets, including at least a location feature set to enable the operator to specify a parameter for implementing a mapping rule to specify a path for a set of destination object containers as compared to a corresponding set of source object containers.

6. The method of claim 1, further comprising providing a user interface to receive the operator input, to enable an operator to specify an input that specifies (i) a first parameter to migrate at least a first source object container of a first container type as a corresponding destination object container of a second container type, and (ii) a second parameter to implement a container-specific policy on the corresponding destination object container, the container-specific policy being available on the destination file system for the second container type and not the first container type.

7. The method of claim 1, further comprising providing a user interface to receive the operator input, to enable an operator to specify an input that specifies (i) a first parameter to migrate at least a first source object container of a first container type as a corresponding destination object container of a second container type, and (ii) a second parameter for implementing a mapping rule to specify a file path for the corresponding destination object container that is contextually different than the file path of the first source object container.

8. The method of claim 1, wherein the migration plan includes a set of parameters that are implemented by default and alterable by input from an operator.

9. The method of claim 1, wherein implementing the migration of the file system objects using the migration plan includes causing a quota tree or directory container on the source file system to be migrated as a volume container on the destination file system.

10. The method of claim 6, wherein the first object container of the destination file system is a volume, and the corresponding source object container of the source file system is a quota tree or directory.

11. The method of claim 1, wherein the migration plan includes a set of conversion rules for implementing the migration as between a source architecture of the source file system and a destination architecture of the destination file system, wherein the source and destination architectures are different.

12. A method for implementing a file system migration as between a source file system and a destination file system, the method being implemented by one or more processors and comprising:

determining a migration plan for the file system migration, the migration plan including a set of parameters and a set of rules, including one or more parameters or rules that are based on an operator input;

determining each of (i) a namespace for the source file system, the namespace identifying a set of file system objects and a file path for each of the file system objects in the set of file system objects, and (ii) a collection of source policies that are implemented for file system objects identified in the namespace of the source file system;

using the migration plan to create a namespace for the destination file system based at least in part on the namespace of the source file system; and

using the migration plan to determine a collection of destination policies for implementation on the destination file system, the collection of destination policies being based at least in part on the collection of source policies; wherein at least one of the namespace or the collection of destination policies include an operator-specific configuration that is specified by the one or more rules or parameters of the operator input.

13. The method of claim 12, wherein using the migration plan to create the namespace for the destination file system includes creating the namespace of the destination file system in accordance with a destination architecture of the destination file system, the destination architecture being different than a source architecture.

14. The method of claim 13, wherein using the migration plan to create the namespace of the destination file system includes converting a set of entries for the namespace of the source file system, which individually specify one or more of a file path or policy for individual file system objects in the set

of file system objects, into a corresponding set of entries for the namespace of the destination file system.

15. The method of claim **14**, wherein using the migration plan to convert the set of entries includes converting individual entries in the set of entries from a source format and structure into a destination format and structure.

16. The method of claim **15**, wherein converting the set of entries includes determining the set of entries for the namespace of the destination file system that are logically equivalent to the set of entries for the namespace of the source file system.

17. The method of claim **14**, wherein using the migration plan to determine a collection of destination policies includes grouping multiple policy entries in the destination format into a single logically equivalent policy entry.

18. The method of claim **14**, wherein using the migration plan to create the namespace of the destination file system includes generating one or more entries in the corresponding set of entries for the namespace of the destination file system to specify that a given destination object container is of a different container type than that of a corresponding source object container.

19. The method of claim **14**, wherein using the migration plan to create the namespace of the destination file system includes generating one or more entries in the corresponding set of entries for the namespace of the destination file system to specify that a given destination object container has a different contextual file path or policy than that of a source object container.

20. A non-transitory computer-readable medium that stores instruction for implementing a file system migration as between a source file system and a destination file system, the instructions being executable by one or more processors of a computer system to cause the computer system to perform operations that comprise:

determining a migration plan for the file system migration, the migration plan including a set of parameters and a set of rules, including one or more parameters or rules that are based on an operator input;

determining each of (i) a namespace for the source file system, the namespace identifying a set of file system objects and a file path for each of the file system objects in the set of file system objects, and (ii) a collection of source policies that are implemented for file system objects identified in the namespace of the source file system;

using the migration plan to create a namespace for the destination file system based at least in part on the namespace of the source file system; and

using the migration plan to determine a collection of destination policies for implementation on the destination file system, the collection of destination policies being based at least in part on the collection of source policies; wherein at least one of the namespace or the collection of destination policies include an operator-specific configuration that is specified by the one or more rules or parameters of the operator input.

* * * * *