US 20140006858A1

(19) United States

(12) Patent Application Publication

(10) Pub. No.: US 2014/0006858 A1

(43) Pub. Date: Jan. 2, 2014

Helfman et al.

(54) **UNIVERSAL PLUGGABLE CLOUD DISASTER RECOVERY SYSTEM**

(71) Applicants: **Noam Sid Helfman**, Redmond, WA (US); **Ken Hines**, Kenmore, WA (US); **Reid Andrew Spencer**, Mercer Island, WA (US); **Moshe Vainer**, Redmond, WA (US); **Kalpana Narayanaswamy**, Redmond, WA (US); **Przemyslaw Pardyak**, Seattle, WA (US); **Ashutosh Tiwary**, Bellevue, WA (US)

(72) Inventors: **Noam Sid Helfman**, Redmond, WA (US); **Ken Hines**, Kenmore, WA (US); **Reid Andrew Spencer**, Mercer Island, WA (US); **Moshe Vainer**, Redmond, WA (US); **Kalpana Narayanaswamy**, Redmond, WA (US); **Przemyslaw Pardyak**, Seattle, WA (US); **Ashutosh Tiwary**, Bellevue, WA (US)

(21) Appl. No.: **13/706,198**

(22) Filed: **Dec. 5, 2012**

Related U.S. Application Data

(60) Provisional application No. 61/567,029, filed on Dec. 5, 2011.

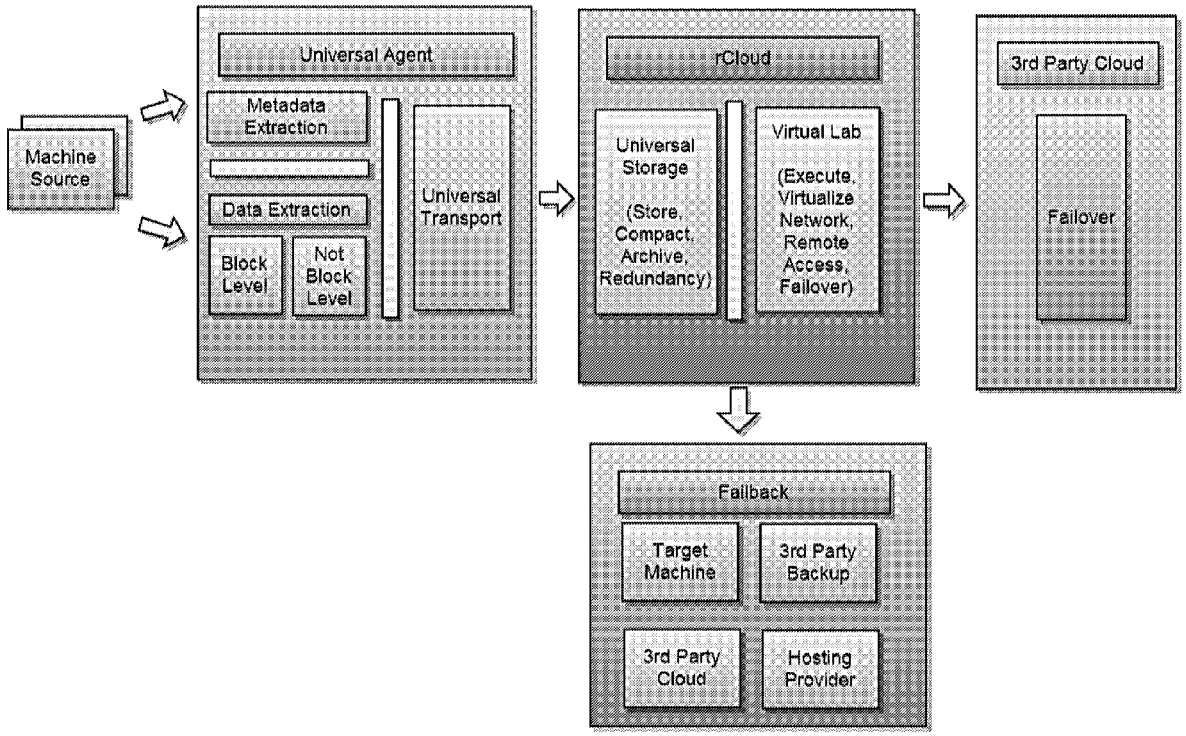
Publication Classification

(51) **Int. Cl.**
G06F 11/14 (2006.01)

(52) **U.S. Cl.**
CPC ***G06F 11/1448*** (2013.01)
USPC **714/19**

(57) **ABSTRACT**

A method, implementable in a system coupled to a display device and a network, includes generating in a first region of a screen of the display device a user-interface portion associated with a first electronic destination address. The user-interface portion is configured to receive from a second region of the screen, in response to a command by a user of the system, a first icon representing a data set. In response to the user-interface portion receiving the first icon, a copy of the data set, or the data set itself, is electronically transferred over the network to the first destination address.



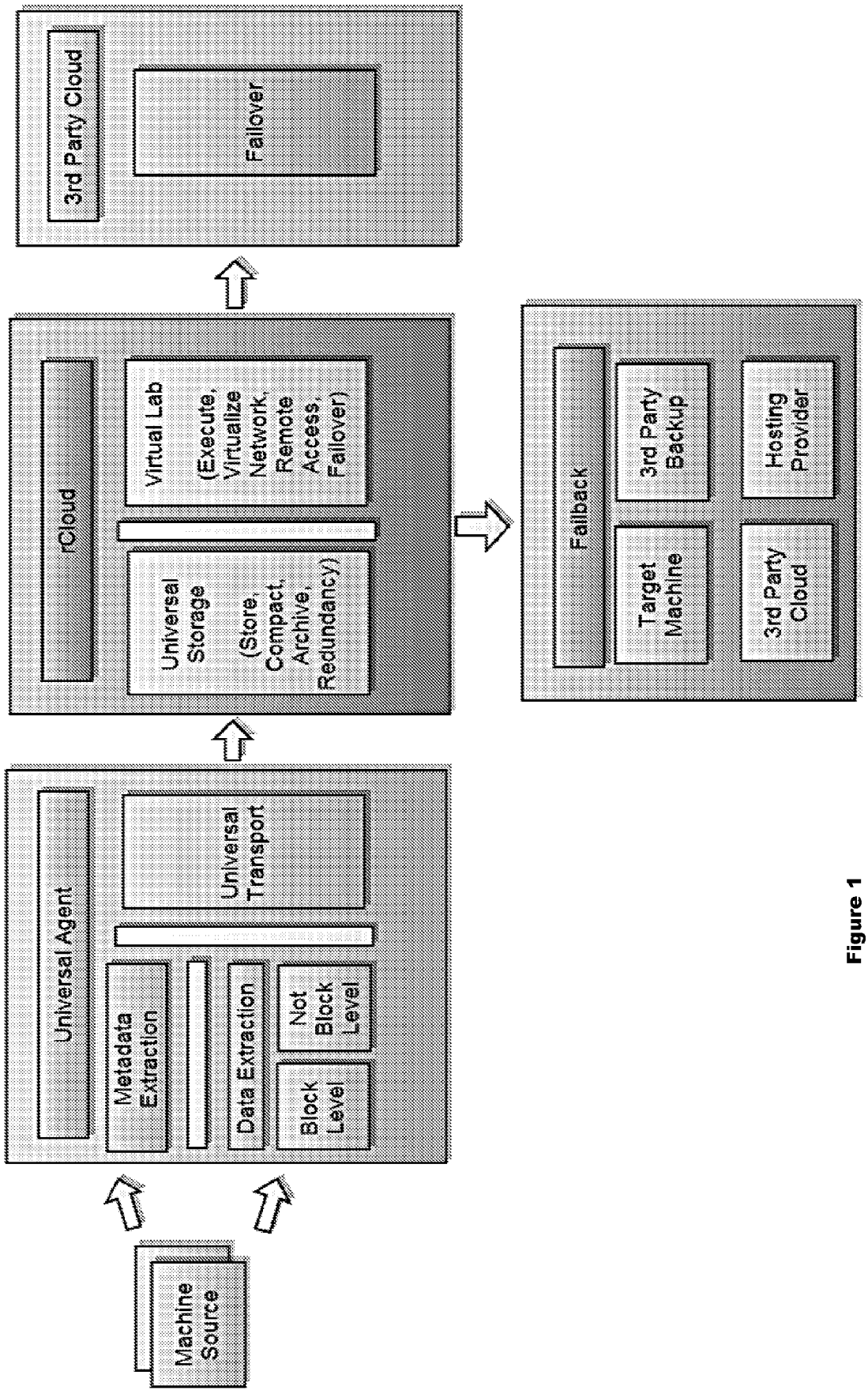

Machine Source
Universal Agent
Metadata Extraction
Data Extraction
Block Level
Not Block Level
Universal Transport
rCloud
Universal Storage (Store, Compact, Archive, Redundancy)
Virtual Lab (Execute, Virtualize Network, Remote Access, Failover)
3rd Party Cloud
Failover
Failback
Target Machine
3rd Party Backup
3rd Party Cloud
Hosting Provider

Figure 1

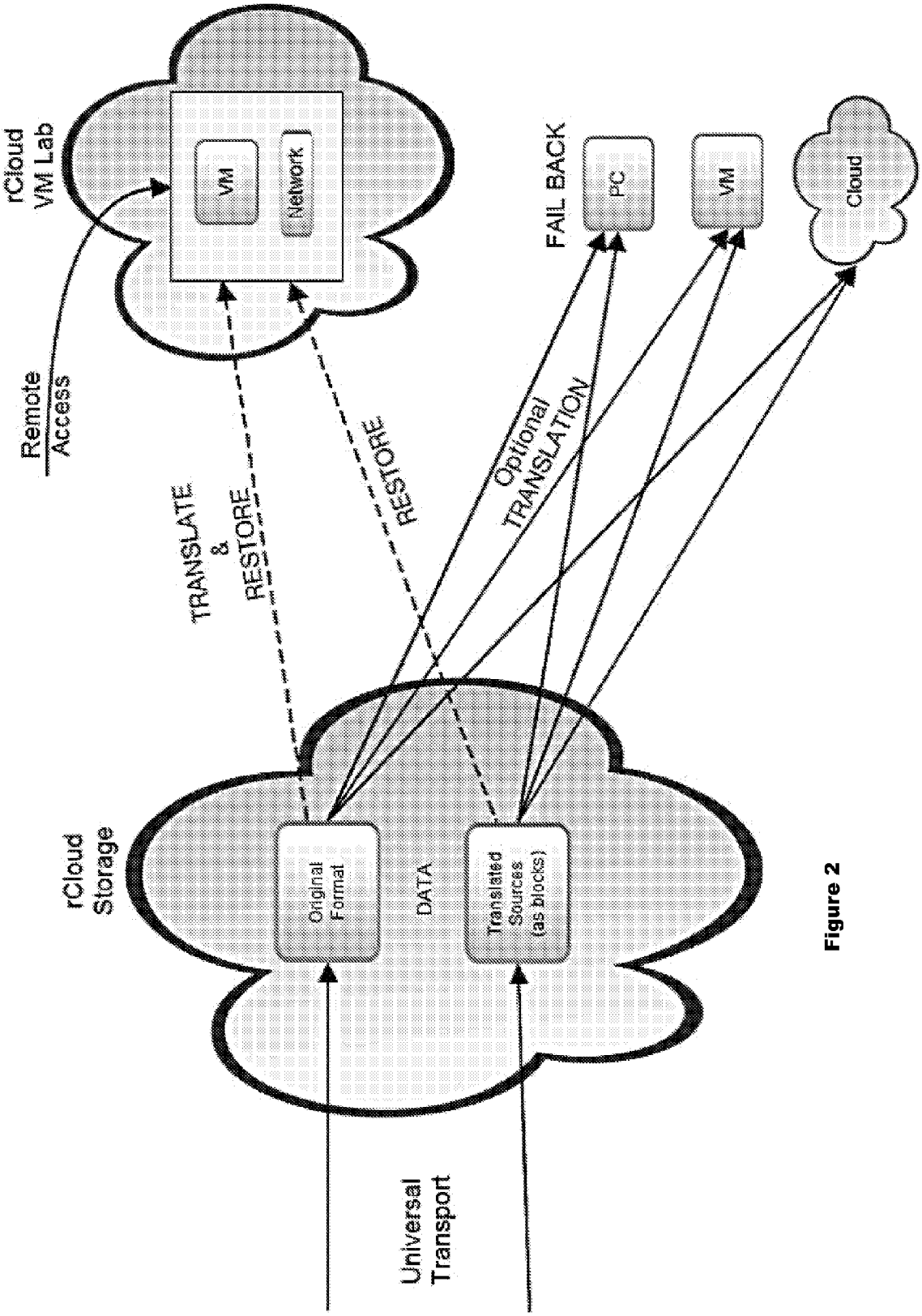

rCloud
VM Lab
VM
Network
Remote
Access
TRANSLATE
&
RESTORE
RESTORE
FAIL BACK
Optional
TRANSLATION
PC
VM
Cloud
rCloud
Storage
Original
Format
DATA
Translated
Sources
(as blocks)
Universal
Transport

Figure 2

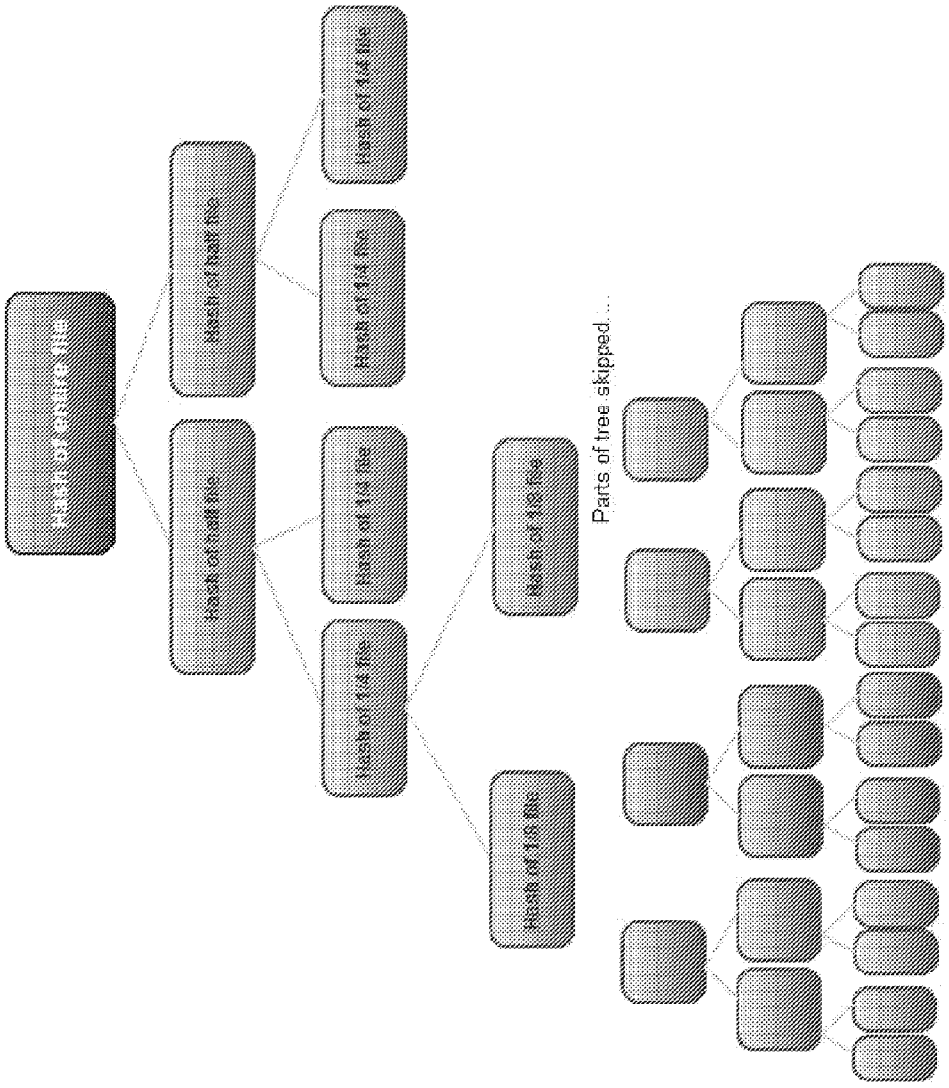

Hash of entire file
Hash of half file
Hash of half file
Hash of 1/4 file
Hash of 1/4 file
Hash of 1/4 file
Hash of 1/4 file
Hash of 1/8 file
Hash of 1/8 file
Parts of tree skipped...

Figure 3

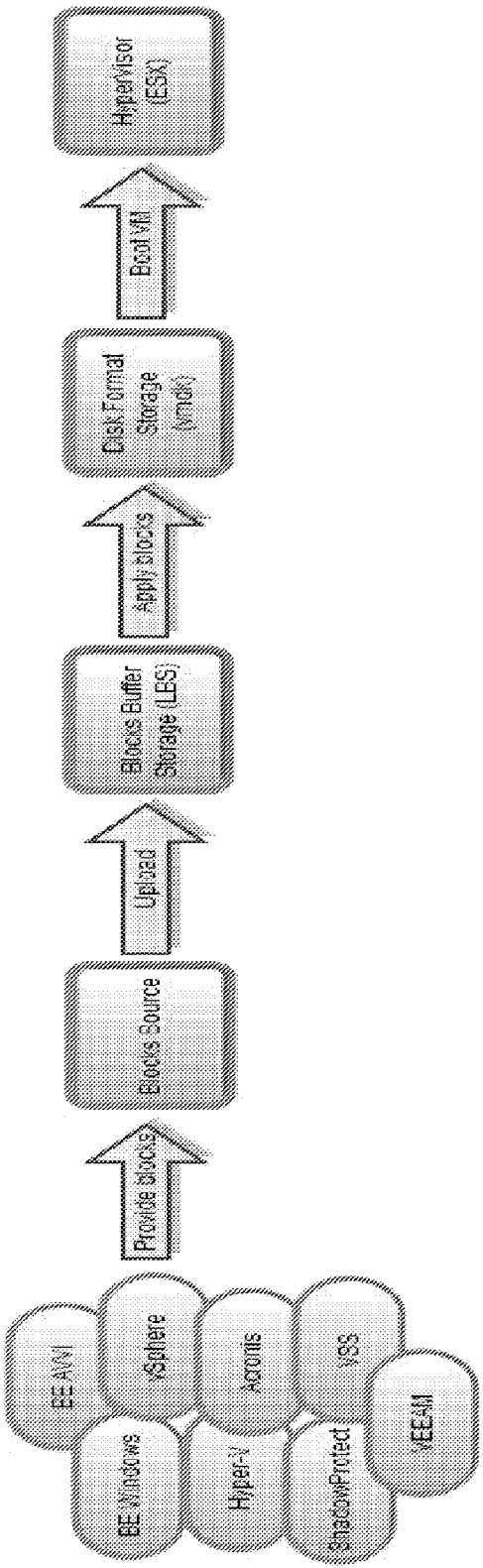

EE AWS
vSphere
Acronis
VSS
EE Windows
Hyper-V
ShadowProtect
VEEAM
Provide blocks
Blocks Source
Upload
Blocks Buffer Storage (LES)
Apply blocks
Disk Format Storage (vmdk)
Boot VM
Hypervisor (ESX)

Figure 4

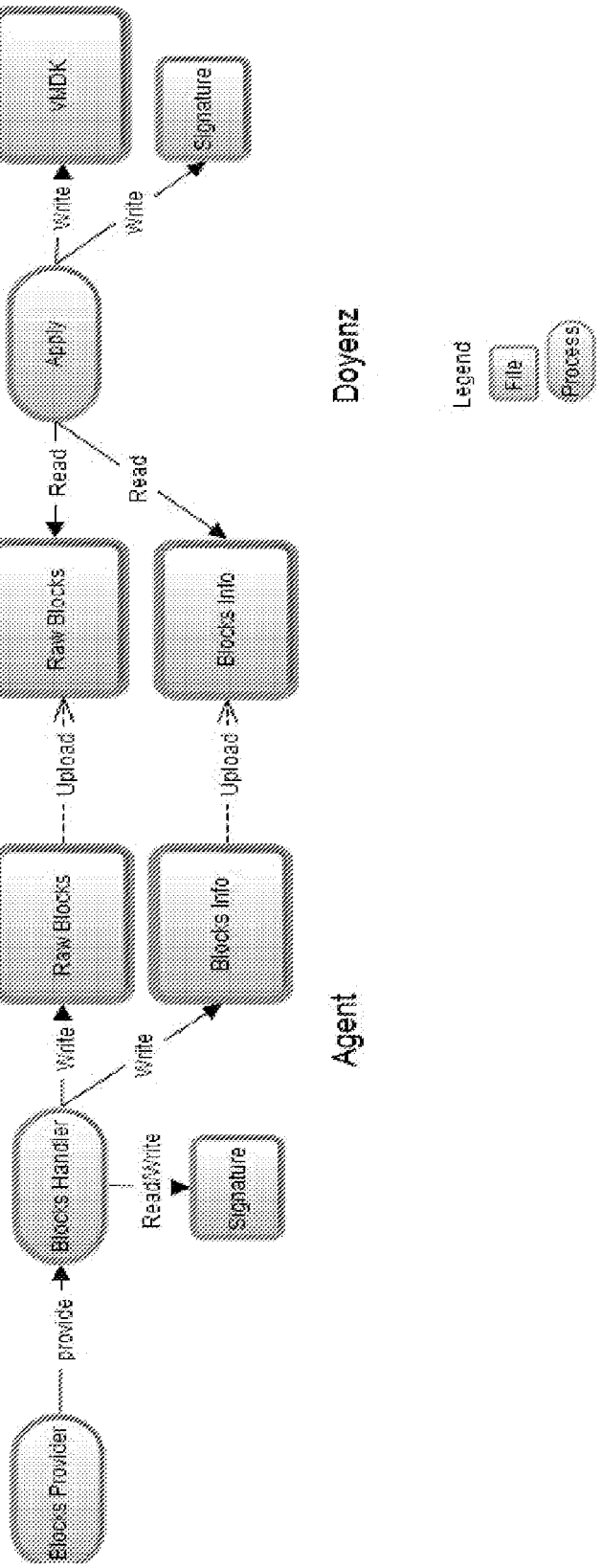

Agent
Doyenz
Legend
File
Process
Blocks Provider
provide
Blocks Handler
Read/Write
Signature
Write
Write
Raw Blocks
Blocks Info
Upload
Upload
Raw Blocks
Blocks Info
Read
Read
Apply
Write
Write
vMDK
Signature

Figure 5

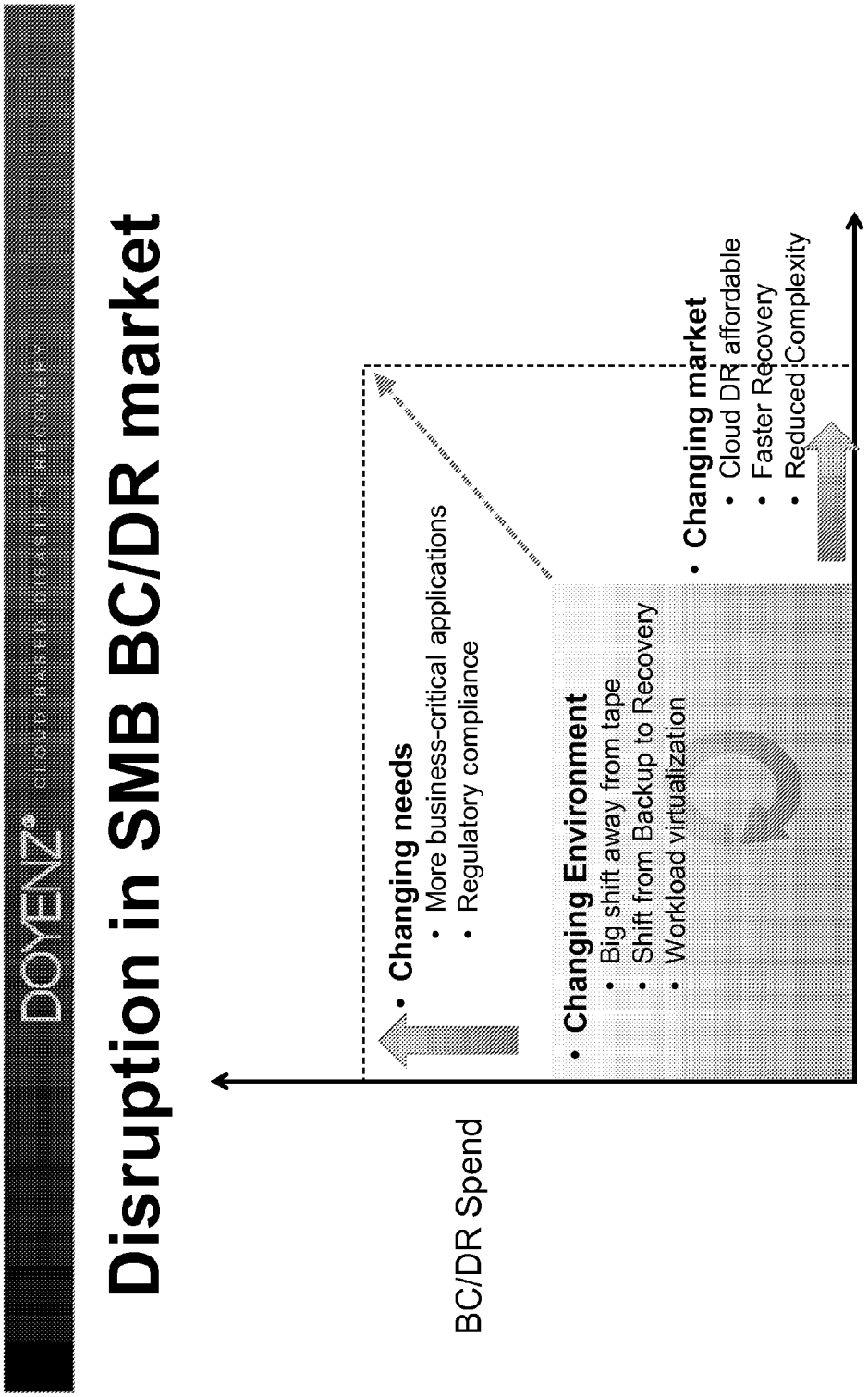

Disruption in SMB BC/DR market
DOYENZ
BC/DR Spend
# of Customers
• Changing needs
• More business-critical applications
• Regulatory compliance
• Changing Environment
• Big shift away from tape
• Shift from Backup to Recovery
• Workload virtualization
• Changing market
• Cloud DR affordable
• Faster Recovery
• Reduced Complexity

Figure 6

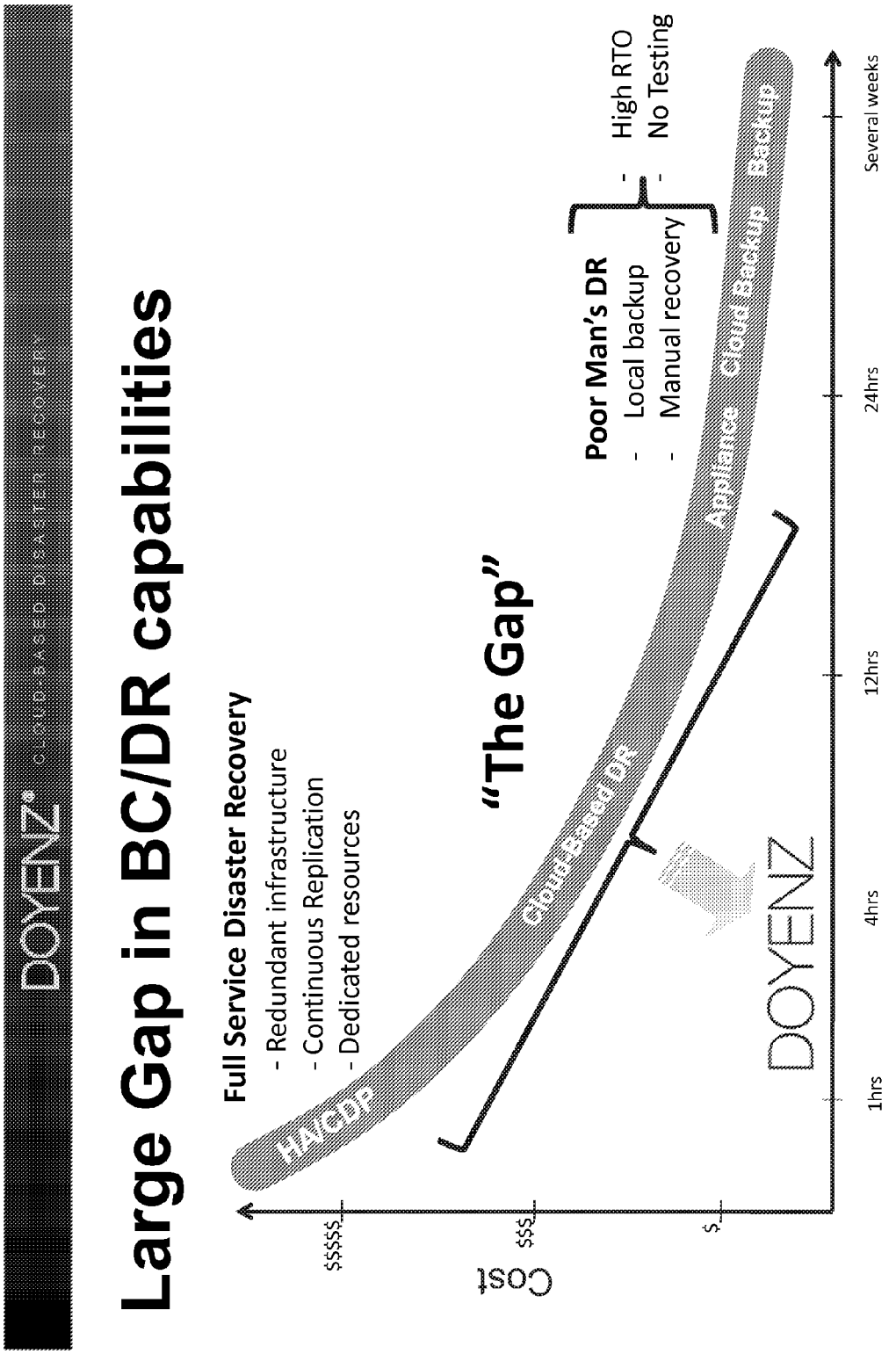
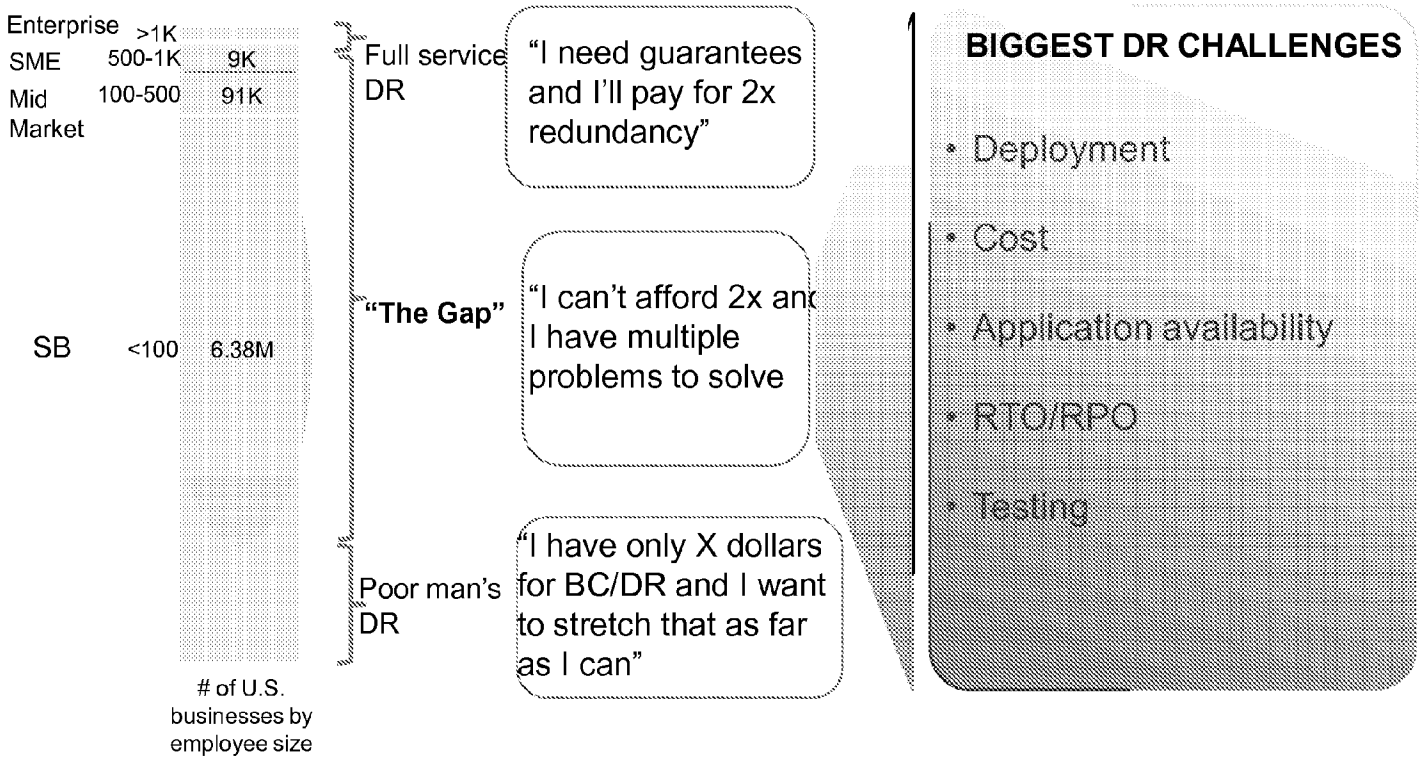

DOYENZ
Large Gap in BC/DR capabilities
Full Service Disaster Recovery
- Redundant infrastructure
- Continuous Replication
- Dedicated resources
"The Gap"
Poor Man's DR
- Local backup
- Manual recovery
- High RTO
- No Testing
HA/CDP
Cloud Based DR
Appliance
Cloud backup
Backup
DOYENZ
Cost
$$$$$
$$$
$
1hrs
4hrs
12hrs
24hrs
Several weeks
Time to Recovery (RTO)

Figure 7

DOYENZ®

SMB BC/DR Market Needs

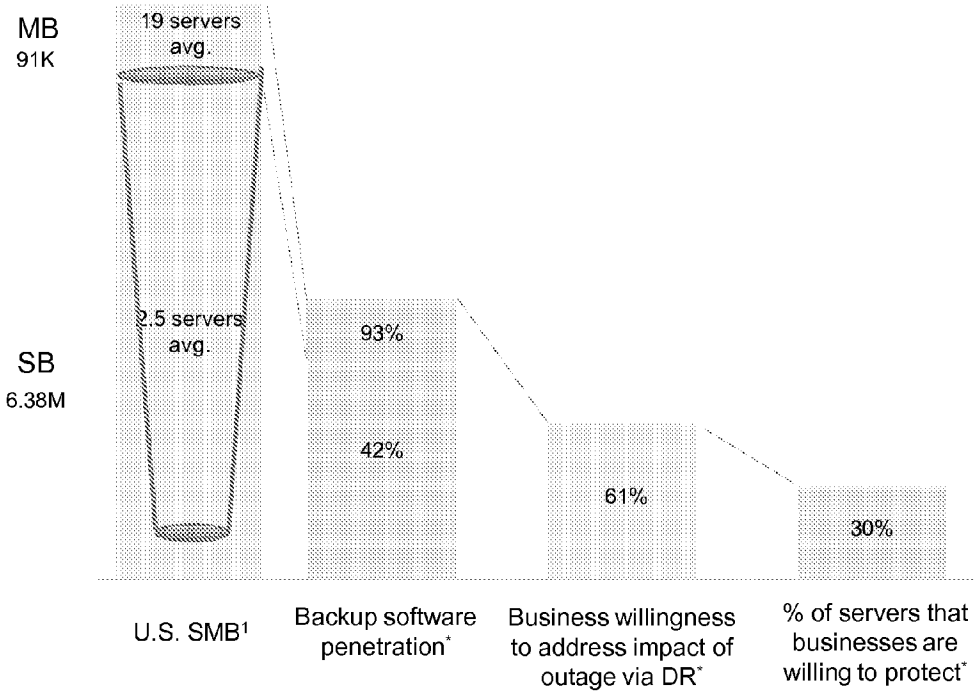


Sources: AMI Partners SMB Market Overview, industry analysts, vendor market survey

Figure 8

DOYENZ®

Size of U.S. SMB DR Market Opportunity



*Sources: AMI Partners SMB Market Overview, industry analysts, vendor market survey
Estimate based on $1.50/GB retail price and 100GB server size on average

Figure 9

= 1.55M servers

≡ $1.50/GB/month
* 100 GB/server * 12 months
* 1.55M servers

≡ $2.8B* annually

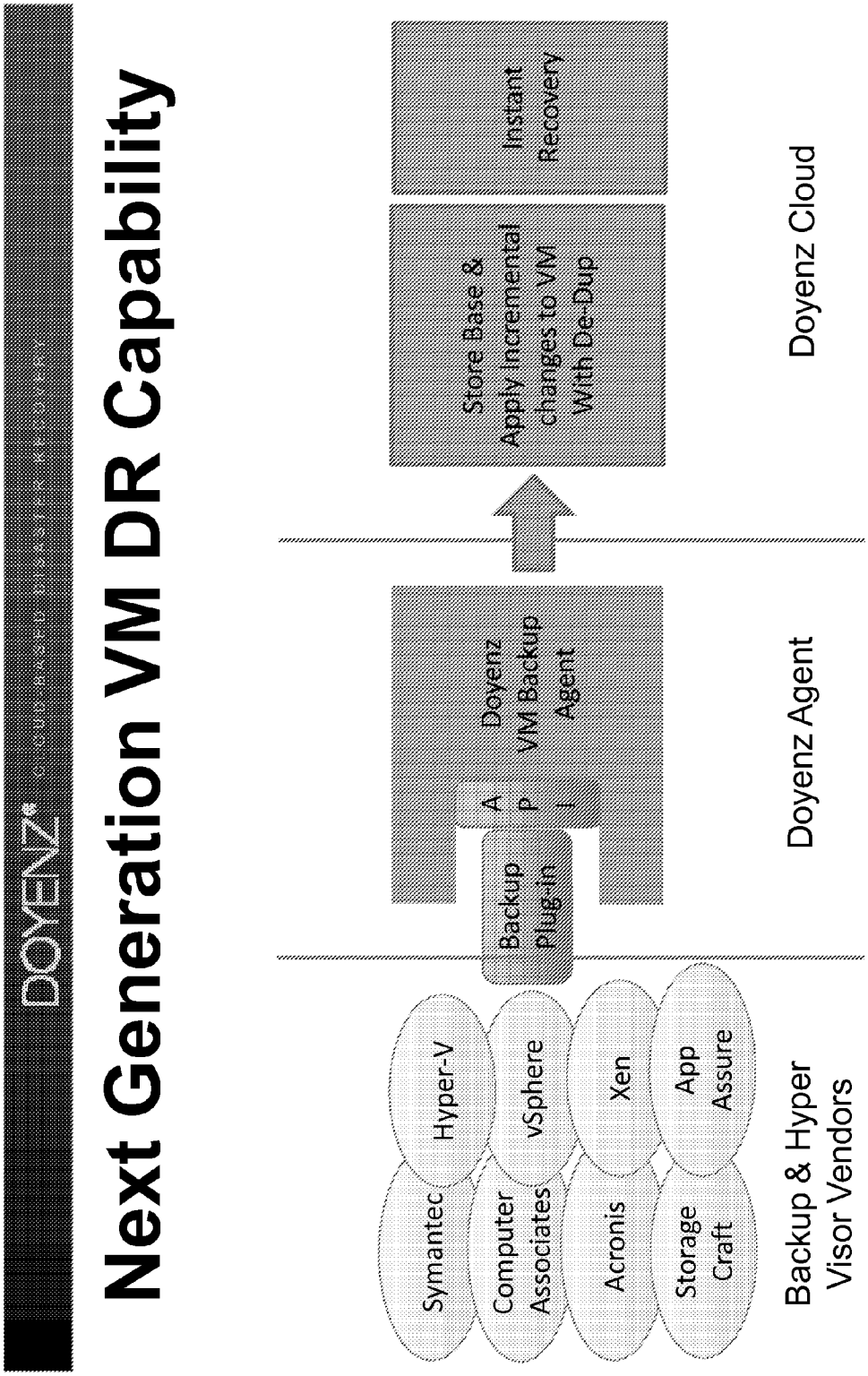

DOYENZ
Next Generation VM DR Capability
Backup & Hyper Visor Vendors
Symantec
Hyper-V
Computer Associates
vSphere
Acronis
Xen
Storage Craft
App Assure
Backup Plug-in
API
Doyenz VM Backup Agent
Doyenz Agent
Store Base & Apply incremental changes to VM With De-Dup
Instant Recovery
Doyenz Cloud

Figure 10

DOYENZ® CLOUD-BASED DISASTER RECOVERY

Cloud-based BC/DR vendors

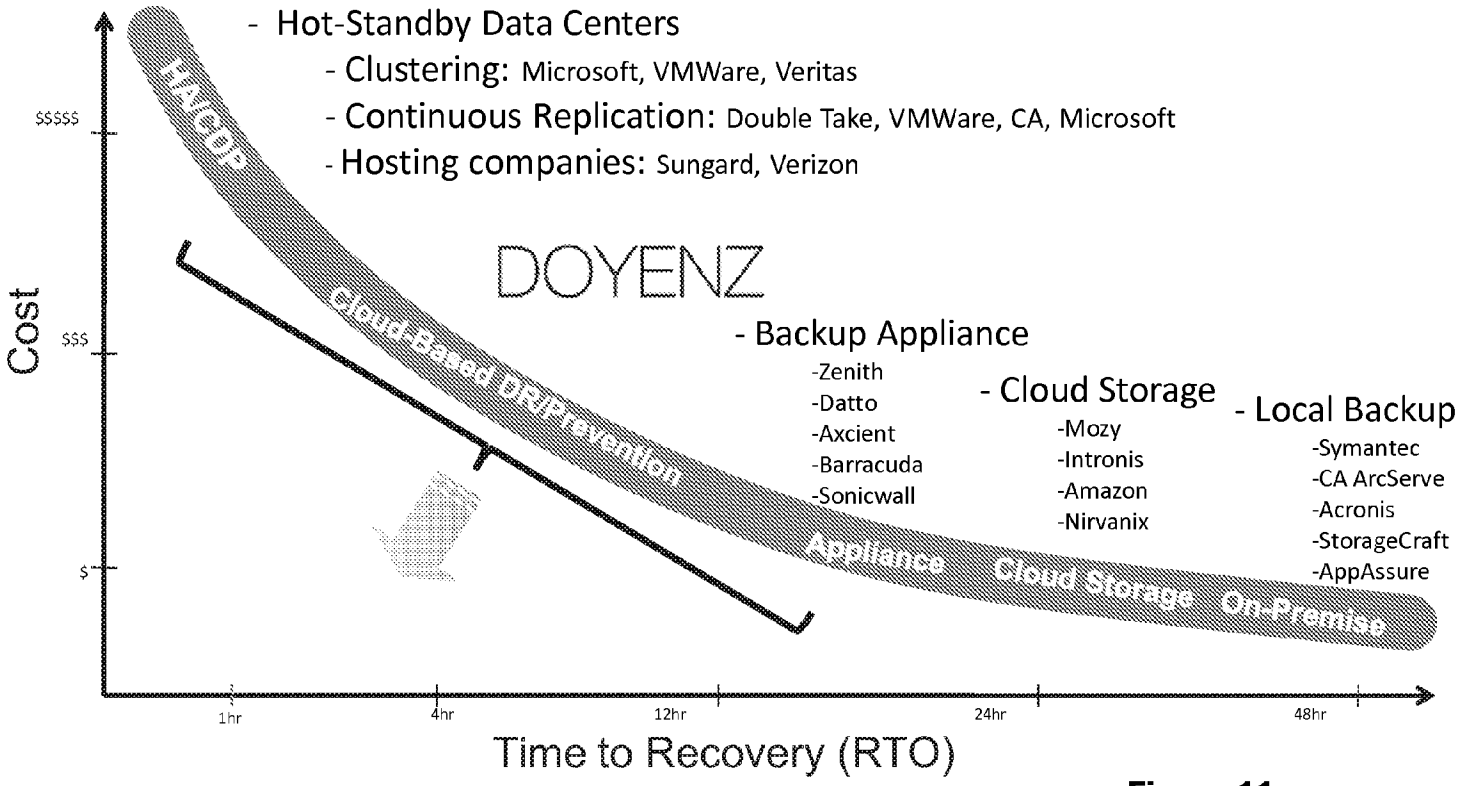


Figure 11

DOYENZ® CLOUD-BASED DISASTER RECOVERY

How it works

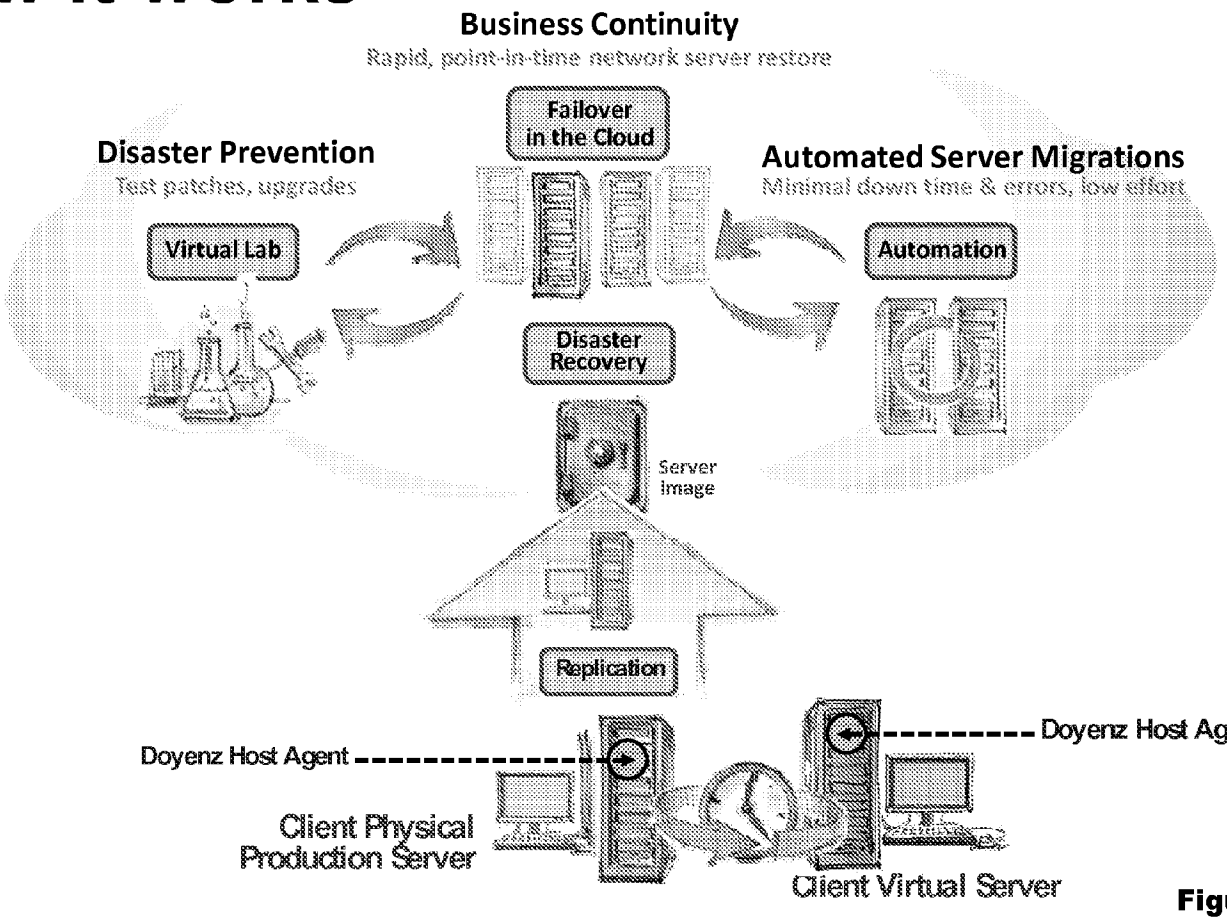


Figure 12

Value Add Service: Cloud On-Ramp

1. Cloud BC/DR is low-hanging fruit
   - Paradigm Shift: Tape → Disk → Cloud
   - Urgent need, ability to pay, build trust for the Cloud
   - Acquire customer data & deliver additional cloud services

2. From failover to always-on in the cloud
   – Build, Package, Deploy VM to managed, public or private cloud

3. Continue to do more with your bits in Cloud
   - BC/DR, Disaster Prevention, Migration, Diagnostics/Forensics

Figure 13

Patent Application Publication Jan. 2, 2014 Sheet 13 of 15 US 2014/0006858 A1

Intellectual Property

- BC/DR & Disaster Prevention **as-a-Service Provider** with Core-IP
  - Unique Technology and extensible platform:
    - Agent, automated virtualized recovery, failover, virtual lab, automation
    - Extensible platform to support multiple backup products and hyper-visors
    - Patent pending
  - Build & operate the service based on that platform:
    - Scale, performance, stability, ease of use, implementation, on-boarding
  - Design, implement & validate reference architecture for cost-effective scale-out:
    - Hardware and software components, configurations, capacity, monitoring, management
    - Commodity hardware, best-of-breed base technologies
  - Market, price, sell, on-board, and support MSPs efficiently:
    - Deep knowledge of customer base and market needs
    - Leader in SMB-IT cloud BC/DR market
- Business, Technology & Service all designed to work together efficiently
- Proven with thousands of protected servers and hundreds of paying partners

Figure 14

Patent Application Publication Jan. 2, 2014 Sheet 14 of 15 US 2014/0006858 A1

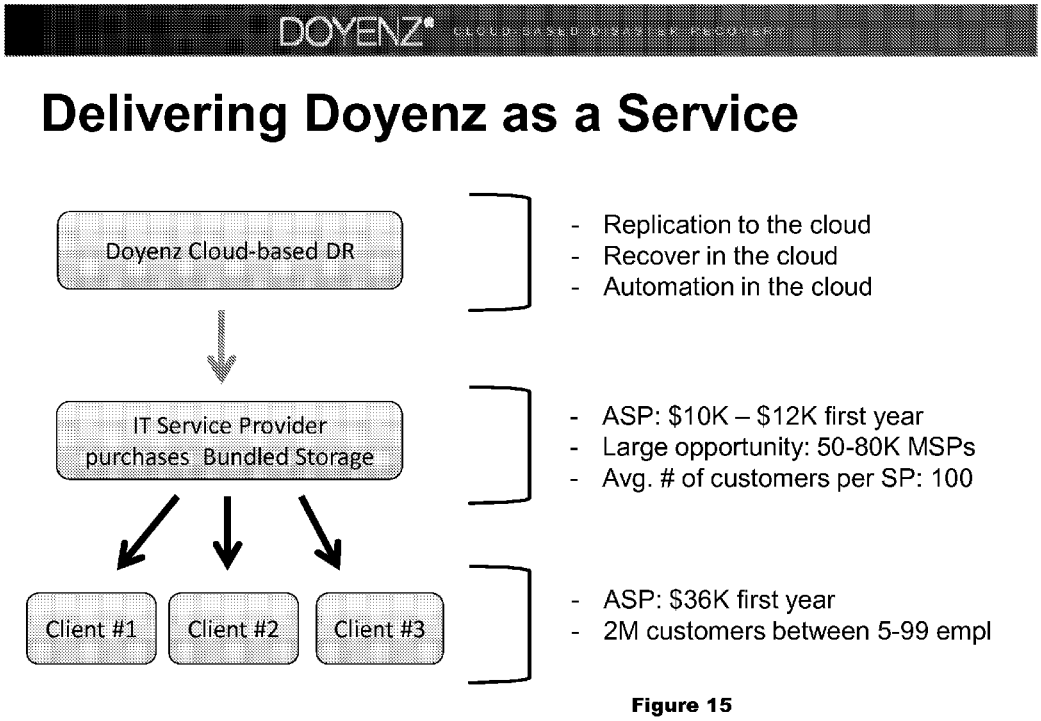

DOYENZ
Delivering Doyenz as a Service
Doyenz Cloud-based DR
- Replication to the cloud
- Recover in the cloud
- Automation in the cloud
IT Service Provider purchases Bundled Storage
- ASP: $10K – $12K first year
- Large opportunity: 50-80K MSPs
- Avg. # of customers per SP: 100
Client #1
Client #2
Client #3
- ASP: $36K first year
- 2M customers between 5-99 empl

Figure 15

UNIVERSAL PLUGGABLE CLOUD DISASTER RECOVERY SYSTEM

CROSS-REFERENCE TO RELATED APPLICATIONS

[0001] This application claims priority to U.S. Provisional Appl. No. 61/567,029 filed Dec. 5, 2011, which is hereby incorporated by reference in its entirety.

BACKGROUND OF THE INVENTION

[0002] An embodiment relates generally to computer-implemented processes.

BRIEF DESCRIPTION OF THE SEVERAL VIEWS OF THE DRAWING

[0003] Preferred and alternative embodiments of the present invention are described in detail below with reference to the following drawings.

[0004] FIGS. **1-15** illustrate elements and/or principles of at least one embodiment of the invention.

DETAILED DESCRIPTION OF THE INVENTION

[0005] This patent application is intended to describe one or more embodiments of the present invention. It is to be understood that the use of absolute terms, such as "must," "will," and the like, as well as specific quantities, is to be construed as being applicable to one or more of such embodiments, but not necessarily to all such embodiments. As such, embodiments of the invention may omit, or include a modification of, one or more features or functionalities described in the context of such absolute terms.

[0006] Embodiments of the invention may be operational with numerous general purpose or special purpose computing system environments or configurations. Examples of well known computing systems, environments, and/or configurations that may be suitable for use with the invention include, but are not limited to, personal computers, server computers, hand-held or laptop devices, multiprocessor systems, microprocessor-based systems, set top boxes, programmable consumer electronics, network PCs, minicomputers, mainframe computers, distributed computing environments that include any of the above systems or devices, and the like.

[0007] Embodiments of the invention may be described in the general context of computer-executable instructions, such as program modules, being executed by a computer and/or by computer-readable media on which such instructions or modules can be stored. Generally, program modules include routines, programs, objects, components, data structures, etc. that perform particular tasks or implement particular abstract data types. The invention may also be practiced in distributed computing environments where tasks are performed by remote processing devices that are linked through a communications network. In a distributed computing environment, program modules may be located in both local and remote computer storage media including memory storage devices.

[0008] Embodiments of the invention may include or be implemented in a variety of computer readable media. Computer readable media can be any available media that can be accessed by a computer and includes both volatile and nonvolatile media, removable and non-removable media. By way of example, and not limitation, computer readable media may comprise computer storage media and communication media. Computer storage media include volatile and nonvolatile, removable and non-removable media implemented in any method or technology for storage of information such as computer readable instructions, data structures, program modules or other data. Computer storage media includes, but is not limited to, RAM, ROM, EEPROM, flash memory or other memory technology, CD-ROM, digital versatile disks (DVD) or other optical disk storage, magnetic cassettes, magnetic tape, magnetic disk storage or other magnetic storage devices, or any other medium which can be used to store the desired information and which can accessed by computer. Communication media typically embodies computer readable instructions, data structures, program modules or other data in a modulated data signal such as a carrier wave or other transport mechanism and includes any information delivery media. The term "modulated data signal" means a signal that has one or more of its characteristics set or changed in such a manner as to encode information in the signal. By way of example, and not limitation, communication media includes wired media such as a wired network or direct-wired connection, and wireless media such as acoustic, RF, infrared and other wireless media. Combinations of the any of the above should also be included within the scope of computer readable media.

[0009] According to one or more embodiments, the combination of software or computer-executable instructions with a computer-readable medium results in the creation of a machine or apparatus. Similarly, the execution of software or computer-executable instructions by a processing device results in the creation of a machine or apparatus, which may be distinguishable from the processing device, itself, according to an embodiment.

[0010] Correspondingly, it is to be understood that a computer-readable medium is transformed by storing software or computer-executable instructions thereon. Likewise, a processing device is transformed in the course of executing software or computer-executable instructions. Additionally, it is to be understood that a first set of data input to a processing device during, or otherwise in association with, the execution of software or computer-executable instructions by the processing device is transformed into a second set of data as a consequence of such execution. This second data set may subsequently be stored, displayed, or otherwise communicated. Such transformation, alluded to in each of the above examples, may be a consequence of, or otherwise involve, the physical alteration of portions of a computer-readable medium. Such transformation, alluded to in each of the above examples, may also be a consequence of, or otherwise involve, the physical alteration of, for example, the states of registers and/or counters associated with a processing device during execution of software or computer-executable instructions by the processing device.

[0011] As used herein, a process that is performed "automatically" may mean that the process is performed as a result of machine-executed instructions and does not, other than the establishment of user preferences, require manual effort.

[0012] Embodiments of the invention may be referred to herein using the term "Doyenz rCloud." Doyenz rCloud universal disaster recovery system utilizes a fully decoupled architecture to allow backups or capture of different types of data, e.g., files, or machines, using different sources and source mechanisms of the data, and to restore them into different types of data, e.g., files, or machines, using different targets and target mechanisms for the data. rCloud may use

different types of transfer, transformation, or storage mechanisms to facilitate the process.

[0013] As applied to disaster recovery, rCloud may include but is not limited to the following functionality and application:

[0014] Support for multiple sources and formats of data, including but not limited to files, disks, blocks, backups, virtual machines and changes to all of them,

[0015] Sources may include but are not limited to full, incremental, and other forms of backups that are made at any possible level, including but not limited to, at a file level, block level, image level, application level, service level, mailbox level, etc and may come from or be related to, directly or indirectly, to any operating system, hypervisor, networking environment, or other implementation or configuration, etc.

[0016] These sources can reside on different types of media, including but not limited to disk, tape, cloud, on-premise etc,

[0017] A simple pluggable universal agent that allows Doyenz or a third party to build a provider for each source of data for a given source solution that allows us to consume that data,

[0018] The consumed data may be transported via the universal transport mechanism to the cloud where it could be (i) either stored as the source and/or incremental change, (ii) applied to a stored instance, (iii) applied to a running instance at any given point in time

[0019] An universal restore mechanism that can take the changes, apply them to the appropriate source data in the cloud and enable rapid recovery, including but not limited to machine and file level backup restore, direct replication to a live instance of the data or machine, etc.

[0020] The recovery can be used for failover, DR testing and other forms of production testing scenario

[0021] This approach allows the ability to provide a cloud-based recovery service to a much larger portion of the market segment.

[0022] While the language in this document uses Disaster Recovery, backups, uploads and cloud as specific examples, it applies equally to any system where different types of data or machines are transferred between any number of sources and targets of different types, for example, digital media instead of machine backups, or two workgroup networks within the same IT organization instead of local hosts and cloud providers.

[0023] Examples, of source and target data include physical machines, virtual machines for different hypervisors or different cloud providers, files of different types, other data of different types, backups of either physical or virtual machines or files or other date provided by backup software or other means. Source and target data may be stored on or transferred through any media.

[0024] Any word such as machine, virtual machine, physical machine, VM, backup, instance, server, workstation, computer, storage, system, data, media, database, file, disk, drive, block, application data, application, raw blocks, running machine, live machine, live data, or other similar or equivalent terms may be used interchangeably to mean either source or target or intermediate stage or representation data within the system.

[0025] Any word such as backup, import, seeding, restore, recover, capture, extract, save, store, reading, writing, ingress, egress, mirroring, copying, live data updated, continues data protection, or other similar or equivalent terms may be used interchangeably to mean adding of data into the system, moving it outside of the system, its internal transfer, representation, transformation, or other usage or representation.

[0026] Any reference to block-based mechanism, operation, or system, or similar or equivalent may be used interchangeably to mean any of the following or their combination: fixed sized block based, flexible sized block based, non block based, stream based, or other form of representation, transfer, operation, transformation, or other as applicable in the context it is used.

[0027] Any reference to block is equivalent to data, data set, subset of data, fragment of data, representation of data, or other as applicable in the context it is used.

[0028] Any reference to cloud, rCloud, system, product, Doyenz, mechanism, service, services, invention, implementation, architecture, solution, software, backend, frontend, agent, sender, receiver or other similar or equivalent term may be used interchangeably to refer to overall system and set of mechanisms being described. Doyenz rCloud may include the following functionality in its implementation:

[0029] Read or write data

[0030] Read or write metadata

[0031] Discover sources, targets, their configuration, other relevant configuration, including but not limited to networking configuration

[0032] Transport mechanism of metadata, data, and configurations

[0033] Machine execution, including but not limited to rCloud or 3rd party cloud environments, different hypervisors or other virtualization platforms, or physical machines.

[0034] Data consumption, playback, or any other form of utilization.

[0035] Backups of data, machine, media, file, database, mailbox, etc

[0036] Restore of data, machine, media, file, database, mailbox, etc

[0037] Failover of machine, service, environment, network, etc.

[0038] Failback of machine, service, environment, network, etc.

[0039] Networking, virtualized or other

[0040] Remote and local access

[0041] Storage, with optional provisions, for example, for compaction, archiving, redundancy, etc.

[0042] Transformation, including but not limited to compression, encryption, deduplication.

[0043] Conversion among different formats, including but not limited to backup software backup file formats

[0044] Maintain and use multiple versions with ability to select, delete, and use for other purposes.

[0045] Maintain and use history or logs of any operations of changes within the system, including as related to any data it maintains

[0046] Instrumentation, other form of interception, attachment, API integration, other communication, for the purpose of capturing it into the system or injecting it from the system into other systems or other purposes

[0047] Doyenz achieves flexibility by decoupling and allowing pluggable implementations that together collect and upload to the cloud info about any machine or other data itself and its configuration, including but not limited to its OS, network configuration, hardware information, disk geometry, etc, and independently allowing the translation thru utilization of plugins of block-level data from any source that rep-

resents file or block information, (see universal agent architecture), and utilizing common or specific transport of the data in to rCloud, where it is stored in the fully decoupled storage solution, thus allowing Doyenz to break the dependence between the source format, transport, storage format.

[0048] Alternatively, Doyenz stores the source data in the format it originates from (for example, local backup files stored in the cloud) and decouples the use of this data by utilization either universal restore or pluggable translation layers that translate source data in to block devices usable by decoupled hypervisors utilized by Doyenz in its rCloud solution.

[0049] When customers come to utilize their machines (e.g. in event of loss of the machine due to disaster event or hw/sw failure, virus attack, etc) stored in the rCloud, this usually means running the machine in the cloud, or failing-over machine to the cloud, or receiving the machine to customer premises, or a hosting provider of the client where the such machine will be running, or receiving the machine in format compatible with a local solution chosen by the customer, that the customer later may restore from. Since Doyenz stores one or more customer machines in the decoupled format that represents metadata about the machine(s) and format that represents customer disks that may be independent from the source format in which machine was uploaded to the cloud. Doyenz can utilize its pluggable restore architecture to construct a target machine suitable to run in Doyenz cloud or compatible to a format chosen by a customer or a format that is compatible to a 3rd party cloud, and utilizing a transport plugin to be downloaded to customer premises, or 3rd party hosting provider chosen by a customer, or 3rd party cloud, or through pluggable and decoupled Lab Manager solution run in the hypervisor of choice in Doyenz rCloud. Additionally, by utilizing decoupled network virtualization and fencing solution, Doyenz rCloud can faithfully represent a network compatible with the network described by a metadata collected from a customer by the time machine was imported or backed-up to the cloud, or a network configuration chosen by a client at the time of restore, or network configuration chosen by the client when machine is running in rCloud, or net configuration chosen by the client as a target network configuration for transporting to the 3rd party cloud, or 3rd party hosting provider, or any other place where the machine could run.

[0050] Such flexible solution or implementation, that allows any machine/source to be represented in the cloud, is called X2C (Any To Cloud).

[0051] And the solution or implementation allowing such machine representation to be executed on any target and/or transferred to any target is called C2X (Cloud To Any).

[0052] rCloud allows conversions from many formats, representations, etc. to many. For example, for backups, this may include but is not limited to

[0053] P2x—from physical to same or different form

[0054] V2x—from virtual to same or different form

[0055] C2x—from cloud to same or different form

[0056] B2x—from backup to same or different form

[0057] x2P—to physical from same or different form

[0058] x2V—to virtual from same or different form

[0059] x2C—to cloud from same or different form

[0060] x2B—to backup from same or different form

[0061] with example combinations of P2V, V2V, V2P, P2C, V2C, B2C, C2C, C2V, C2B, C2P, etc.

[0062] Blocks will be applied to a vmdk (or any disk format we would like to support) (same as storage agnostic)

Preferably, all hypervisors can encapsulate entire server or desktop environment in a file. Commonality of virtual machine disk formats enables us to support wide area of formats.

[0063] Failover to any Cloud

[0064] Doyenz's DR solution (rCloud) allows a special kind of restore—failover, where the customer's machine is made to be available and running in the cloud and accessible by the customer. rCould solution decouples backup source, storage, and virtual machine execution environment (Lab-Manager). This approach allows Doyenz a greater flexibility of failing back to any cloud solution as a target. As a result, customer machine may start its life as physical machine, P2C to Doyenz rCloud (or any other cloud-based storage, like S3) then fail-over in to the instantly created virtual machine instance in the ESX virtualized environment as an example that Doyenz cloud currently utilizes, and then fail back to customer environment as a Hyper-V appliance (C2V) or other virtual solutions.

[0065] OS Agnostics

[0066] Doyenz's DR solution works hand-in-hand with hypervisor software and therefore any virtual machine type/OS combination that is supported by a hypervisor is also supported by our solution.

[0067] Single agent for One machine/Multiple machines/Multiple types of machines One instance of the agent is capable of handling multiple machines, both physical and virtual machines, including hypervisors. In addition, multiple physical (and virtual) machines, that are backed-up by a 3rd party standalone backup agent(s), could be handled by the same Doyenz's Agent.

[0068] Storage Agnostic

[0069] Since Doyenz's backup solution is based on storing blocks of data, we are not limited by any storage provider, it could be just a SAN storage, NAS storage, any storage cloud, distributed storage solution, technically anything that is capable of storing blocks reliably

[0070] Universal Restore

[0071] Doyenz Universal Storage stores data coming from sources can be described as belonging to at least two different types of formats—

[0072] storage formats that can be directly consumed as block based devices

[0073] other possibly proprietary storage formats that for example originate from 3rd party backup providers and are stored unchanged or modified on Doyenz storage

[0074] other formats that may be translated to and from the above

[0075] The act of restoring, failing over or otherwise executing said machines in Doyenz or third party clouds may involve one or more of the following steps:

1. Configuring a virtual or physical machine in the destination lab to conform to the metadata configuration that was captured at the time of backup and describes the source machine (e.g. amount of memory, number and type of disks, bios configuration etc. . . .)

2. Exposing the stored disk data that corresponds to the restore point in time in a format that is directly readable as disk by the target lab.

Doyenz may utilize a plug-in that is aware of the target lab api (either doyenz or third party) on one hand, and metadata format stored in doyenz on another hand, and using the target

lab api can configure a virtual or physical machine that conforms to original source configuration.

Where the source data is stored on Doyenz storage as block device, the block device may be directly attached as disks to the target lab using standard lab apis and standard remote disks protocols, e.g. iSCSI, NFS, NBD etc.

Where the lab is local to doyenz, such block devices can even be represented as locally attached files, e.g. VirtualBox based lab on ZFS based storage

Where the source data is stored not as a block device, e.g., in a proprietary 3rd party format, Doyenz implements several strategies to make the source data universally accessible by the target lab including but not limited to:

[0076] 1. Using original 3rd party software to perform a 3rd party restore to a destination block device—in this case the 3rd party software is either driven through an API it makes accessible or Doyenz utilizes proprietary doyenz automation (prey. patent) to functionally drive the restore process through the UI in a specially purposed virtual machine.

[0077] 2. Where a 3rd party software provider provides mount tools that can mount a backup file to a local machine, such tools can be used to mount the backup file and represent the resulting mounted disk as a remote or local disk to the lab.

[0078] 3. Where a 3rd party backup software provider provides mount tools that can mount a backup file to a local machine, doyenz can utilize methods described in universal agent disclosure to scan the mounted disk and translate/copy the blocks to an intermediate destination block level device that is compatible with destination lab

[0079] 4. Where a third party backup software provider provides integration into a hypervisor, (e.g. storagecraft virtaulboot), doyenz can utilize a version of doyenz lab that is compatible with said 3rd party provider's choice of hypervisor and therefore make lab compatible with the source.

[0080] 5. Otherwise any form of interception, integration, or instrumentation, or similar may be used to capture the needed data and configuration

[0081] Where any transformation is performed on the stored disks, such that the target lab's hardware differs from hardware abstraction layer deployed in the guest operating system on the source machine, and the operating system does not support universal hardware (e.g. windows) a special process of adjusting said source to be run in a lab with different hardware or hypervisor is performed.

[0082] In those steps, the source disks in the target format are mounted either locally in storage or in destination virtual machine or in special virtual machine where a specially designed piece of software replaces hardware abstraction layer and installs drivers to make the machine compatible with target lab.

[0083] Where 3rd party software used in restore process already provides such functionality it can be used as part of restore process by running the restore itself on the target physical or virtual hardware to automatically convert restored disks to be compatible with target physical or virtual hardware.

[0084] Restore/recovery may be implemented for different types and formats of data or machine, including but not limited to, file level, disk, machine, running machine, virtual machine, recovery directly into a live running instance.

[0085] Universal Failback

[0086] The act of failback differs from the act of restore or failover in that Doyenz could provide a machine that is either stored in doyenz storage or is running in Doyenz lab in a target format and/or to a target destination of customer's choosing and doesn't necessarily require running the machine in Doyenz or any other lab.

[0087] In case where doyenz storage used for regular store of the machine source or used as a transient translated format for running machine in the lab is compatible with target format required by customer, the source or transient storage is then transfered to the customer or to 3rd party cloud w/o any transformation applied to the data.

[0088] Where target format is different from the format that the source is stored in the Doyenz storage, and Doyenz stores the data in block-based format, and destination

[0089] In addition, any mechanism or method that applies to a backup and restore may apply to failback.

[0090] Example transformations and usage depending on available formats.

Doyenz format	Target format	Actions
same as target	same as source	Download
3rd party mountable	3rd Party	Mount and perform backup
3rd party mountable	3rd Party mountable	Mount both and perform block-level copy
3rd party non-mountable	3rd Party mountable	Restore to a mounted 3rd party target
3rd party non-mountable	3rd Party non-mountable	Restore to Doyenz block-level storage and backup from Doyenz storage using 3rd party's backup software
Block level	Block-level with different header or other metadata	Transfrom header or metadata and download
3rd party mountable	Block level any	Mount and perform block-level copy
3rd party non-mountable	Block level any	Restore to a mounted block-level
Block level	Different block-level or mountable 3rd party	perform block-level copy
Block level	Non-mountable 3rd party	Mount and backup from Doyenz storage using 3rd party's backup software

[0091] When the destination is a block-level format (or 3rd party cloud) and as such where 3rd party software is not required to perform transformation (if any), the actual target data is not necessarily stored in Doyenz cloud but could be stream directly

[0092] as downloadable stream to customer destination

[0093] or pushed as an upload stream to 3rd party cloud,

[0094] or downloaded by Doyenz Agent as any block-level format, where the agent assumes responsibility to provision set data either to locally available physical disks or directly to the customer's hypervisor of choice

[0095] Autoverified Backups

[0096] Doyenz may apply multiple levels of verification to make sure that at any given point in time backups and or imports and or other types of uploads into doyenz or any other service that implements doyenz technology where such backups uploads or imports in any way represent a machine are recoverable back into a machine representation whether it is

a physical machine or virtual machine or a backup of such or any other machine recovery type.

[0097] All verification steps are optional. All verification steps may be performed before, during, or after the relevant other steps of system's operations. All verification steps may be performed in their entirety or partially.

[0098] Upload verification, preferably:

[0099] a. Every upload may be broken down into blocks a.k.a chunks and each chunk may be assigned a cryptographic or other hash value and/or checksum or fingerprint value.

[0100] b. A running checksum for the entire file/stream/disk being uploaded can also be calculated

[0101] c. The server can validate that the hash/checksum values for uploaded data can be independently recalculated and compared to the data calculated on the customer side to ensure that no discrepancy occurs during transmit.

[0102] d. In case of discrepancy the agent may retransmit the chunks where crc or checksum or fingerprint or hash values are in a mismatch

[0103] e. Before applying incremental changes, doyenz service responsible for copying uploaded bits may roll back to a previously known good snapshot, thus ensuring that any accidental writes or changes to the filesystem can be removed prior to apply.

[0104] f. Upon apply a new filesystem snapshot can be taken, thus ensuring that, preferably

[0105] i. The data is safely committed to disk

[0106] ii. The data cannot be tampered with (or the state before tampering is recoverable) once on disk and next apply has a reliable base to apply to or such base can be reconstructed.

[0107] Recovery verification, preferably:

[0108] g. Doyenz may employ a verification stage to verify recovery of every upload or of selected uploads (or backups or imports)

[0109] h. The verification stage is part of Doyenz pluggable architecture and backup providers (whether Doyenz or ThirdParty) can add verification steps

[0110] i. By default, generic verification step includes attaching the uploaded disk to a virtual machine, and/or verifying that it successfuly boots up and/or verifying that the os is initialized. In case of need, hardware independent adjustments are performed on the OS to ensure its ability to boot (e.g. replacement of HAL and installation of drivers).

[0111] j. Any adjustments or changes to the disk as the result of the boot can be discarded upon completion of verification using a temporary snapshot of the target filesystem (or other COW (here and elsewhere: copy on write) or similar mechanisms, or otherwise by creating a copy prior to verification)

[0112] k. In case verification fails, the backup can be chosen to not be allowed to complete, or other remediation steps can be taken to ensure validity of backups and if necessary can include notification of customers or of staff etc. . . .

[0113] l. In case a disk is not a block device, but the backup provider provides a means by which the backup files can be mounted as block device, the plug in for the particular backup provider can be used to allow mounting and performing similar verification as a block based device

[0114] m. In case a disk is not a block device but the backup provider provides tools for chain verification, verification plugin can perform chain verification as its verification step

[0115] n. In case the backup provider provides other means of backup correctness verification, the plug in will utilize those in the same general flow of apply->verify->finish or wherever the verification plugin is called, or on demand through the interface or through public doyenz api to make sure that every backup (or any particular backup) is recoverable

[0116] o. In addition, if no other verification is sufficient or possible, Doyenz rCloud can perform an actual B2C or V2C or any other type of conversion of the backup files in question to mountable disk format to ensure successful recovery and upon completion of B2C process can perform a virtual machine verification.

[0117] p. In addition, Doyenz plug in architecture allows Doyenz and 3rd party providers, including customers themselves to provide verification scripts. E.g. if customer has a line of business application and can provide a script that will ensure that line of business app is running upon system boot, doyenz verification process will execute this script during verification stage to make sure that the LOB application is performing properly upon every backup

[0118] q. Additionally, by providing multi tier plug in architecture to the verification process, Doyenz allows for business to provide tiered pricing options for different levels of verification, starting from basic—e.g. CRC/Hash upload verification and all the way to LOB specific verification scripts.

[0119] r. In addition, LOB specific verifications can be produced by Doyenz for popular applications, e.g. Exchange servers, SQL servers, CRM systems etc, to verify commonly used software is functional in the cloud version of the machine

[0120] s. In addition, those generic verification scripts for popular or otherwise chosen applications can be made customizable by customers, e.g. for exchange server, customer may provide a particular contact to be found, or a rule that a recent e-mail must exist etc. . . .

[0121] Fingerprint Map Reduction for Dedup

[0122] One of the ways to provide for uploads of large amounts of data is to represent each block or chunk of data being transferred with a unique hash or fingerprint or checksum value where such value is algorithmically calculated from the source data or otherwise identifies with some certainty the source value and compare those fingerprint/hash/crc etc values with a known list of previously transmitted or otherwise already existing values on the server side. However, to provide a hash value that one can be confident enough is truly unique; the hash values need to be significantly large.

[0123] It is usually accepted (though not required for the purpose of current invention) that such values should be in the order of 128 to 512 bits, or 16 to 64 bytes.

[0124] In addition, the likelihood of a block (or any other piece of data) being found to already exist, thus making deduplication efficient is in inverse proportion to the size of the blocks being hashed/compared. That is the larger the block, the more likely that every block in the transmit has experienced some level of change and will therefore have to be transmitted. On the other side, reducing the size of the block can lead to an unfavorable relation between the size of

the hashed values compared to block sizes. For example, if one were to choose blocks of 512 bytes for best deduplication and 512 bytes hash size for best confidence and lack of collisions, the size of the hash is equal to size of original data, and therefore there is no advantage in using it at all.

[0125] Therefore, we propose a method of optimistic hash size reduction for the purpose of deduplication of data uploads.

[0126] In this scheme, the size of hash algorithm chosen can be (though not required to be) optimistically small, e.g. a standard CRC of 32 bit. This provides the benefit of fast calculation of hash and small sizes of hash values, also providing for fast exchange of CRC maps between the server and the client.

[0127] While this can lead to an increased rate of collisions, if the CRC or the hash differ, we can be guaranteed that the blocks are indeed different.

[0128] Given that they differ with mathematical certainty, we can transfer those blocks to the server w/o incurring the cost of storing and calculating larger hash values.

[0129] The rest of the blocks have the potential to exist on the server, but can also be a collision that was otherwise undetected because of relatively small size of the hash.

[0130] Next step of the process can now collect ranges of data comprising of multiple blocks that are suspect to be the same and perform validation of their equivalence either by utilizing tree hash algorithm (see description of tree based hashing dedup) or by calculating a single large size hash for every range. Those ranges of blocks that prove to be equal even after a significantly large hash comparison need not be transmited, while blocks that have proven to contain at least some collision using large block comparison need to be further examined.

[0131] Depending on the size of the remaining ranges, one can iterate through the process by using either next level in the tree using tree based dedup or by increasing the hash size one more step and repeating the entire process for each suspect range.

[0132] This provides for minimal data to be calculated and exchanged between the client and the server for the most efficient transfer of incremental changes in large files.

[0133] Tree Based Hashing for Optimal Change Transfer

[0134] When using hash (aka fingerprint or checksum) based fingerprint files to deduplicate transfer of large files, the fingerprint files themselves can be of significant size. E.g., using a 256 bit hash algorithm, on a deduplication block of e.g. 4 kbyte an example 2TB disk would produce a hash fingerprint of 16 GB. Exchanging that much information for the purpose of figuring out which blocks have changed can potentially be larger than the entire change to be transferred.

[0135] One solution to such problem is to hold a local cache of the fingerprint file. As long as this file is kept up to date and its validity can be verified (e.g. by exchanging a single hash for the entire fingerprint file) the local copy can be used as a true reference and blocks can be hashed and compared individually to the local fingerprint file.

[0136] If however local cache space is limited, the entire hash structure would need to be exchanged if each block is represented by a single hash. Assuming a limited hash size that can fit in memory, an alternative approach to identify changed blocks is a tree of hashes. A tree of hashes is a tree where each terminal node is a hash value of a particular block (e.g. 4 k size block), and each parent node is either a hash of the data of all its children or a hash of hashes of all its children. Hash of hashes differs from hash of all children by the fact that the source data used to calculate the hash of the larger block is the hash of the smaller blocks it is comprised of, whereas in the other case, the entire larger block source data is used to calculate the hash.

[0137] Taking for example available in memory (or on disk) buffer space of a little over 1 MB (and for example 4 kb blocks), one can read 256 blocks of data and fit it entirely into buffer. As they are read (or after they are read using a separate scan), a tree of hash values can bebuilt such that the lowest level of the tree contains hash values for each (e.g. 4 k) block, next level up containing hash values for e.g. each 8 k of blocks etc.

[0138] The overhead size of such hash tree would be (assuming binary tree, 256 bit hash 4 k block size) would be a total of 16 kb, where the root node of the tree would be a hash of the entire 1 MB.

[0139] This tree would correspond to a branch of a hash tree of the entire disk (or source data) that resides on the server. (e.g., in diagram below, the green subtree is for example a branch that corresponds to the first buffer, purple branch corresponds to next buffer read, where as all the nodes together comprise the hash tree of the entire transmission (or file/upload))

[0140] The branch location in the global tree is determined by buffer size (e.g. lmb) and offset in the disk (e.g. the purple branch is offset for example by lmb from the green branch in the diagram above), thus each client can use different buffer size depending on available memory and disk space and still utilize the same generic branch exchange algorithm.

[0141] The branch (or a tree of the buffer) will then be streamed to the server in BFS order. As the server starts reading the stream, first bytes represent the hash of the entire buffer. In case they are equal to the hash of the appropriate root of a branch in full tree representation, the server can immediately stop transmit with a response to the client stating that the branches are equal and next buffer can be filled. Such response can be done either synchronously (that is the client waits for a response after each hash or several hashes being transmitted, or after each bfs level, or any other number of hashes, or as an asynchronously read response stream, that is the server responds as the client uploads the hashes, w/o waiting for the entire transmission to end, and potentially as soon as the server has replies available after comparing with a local representation of the hash tree)

[0142] In case hashes at the root of the branch differ, the streaming continues, and the next two hash elements in the stream each represent a hash of half the buffer size (assuming binary tree) (the streaming does not necessarily need to wait for response, but can continue independently). Once again, the server continues to respond (either in line, or synchronously). E.g. if the first half differ and the other is equal, the server will respond instructing the client to continue traversal only on the first half of the branch. Server responses can be as short a single bit per each hash value. Continuing to go down, a bitmap of all blocks that actually differ will be negotiated, and the upload of actual data can begin (or be done in parallel as the blocks are identified).

[0143] Worst case scenario overhead for such algorithm, assuming the disk has completely changed is 2N where N is the size of a flat fingerprint file. However, for buffers that have not changed, the overhead is as low as a size of a single hash each. Assuming 5 percent change on each backup, the information that needs to be exchanged on a 2TB disk size to fully

identify changed blocks, w/o requiring significant buffer space on the client side would amount to (assuming 256 bit hash, 4 k blocks) is a mere 1.6 GB, whereas the changed data size is 102 GB.

[0144] Plugin Based Cloud Architecture for Providers of Specific Decoupled Functions such as Restore, Hir, Automation, Etc.

[0145] In rCloud, some of the goals include the support of multiple representations of customer machines in the cloud, backing them up (or otherwise uploading/transmitting) into the cloud, verify such backups, run such machines in the lab, fail over to the cloud in case of disaster recovery and fail back to the customer environment when the event is over. In the real world of IT, customers have a diverse multitude of machine types and local backup providers that may be utilized in the course of their IT operation. Those include but are not limited to:

[0146] Physical machines with OS directly on the physical hardware

[0147] Virtual machines running in a variety of hypervisors

[0148] Local backups by multitude of third party backup providers with different backup strategies

[0149] Machines hosted in hosting environments

[0150] Virtual machines running in a third party cloud

[0151] Creating a regularly updated cloud based image of such machine sources is a conversion to the cloud. (X2C).

[0152] Doyenz therefore performs standardized operations on nonstandardized multiverse of sources.

[0153] By standardizing the operations, and then applying a plugin api to each or some of the operations, we can support the multiverse of sources by either minimal engineering investment in each new source of machine coming into the cloud, or allow third party providers to adjust their own solutions to be compatible with Doyenz.

[0154] Thus doyen can decouple—Source from Transport from Storage from Hypervisor from Lab Management etc. . . . and each can be independently adapted.

[0155] This allows us to change e.g. the best available hypervisor platform regardless of the type of VM customers chose to run etc.

[0156] Taking for example the process of daily backups

[0157] The preferably generalized process comprises one or more of the following—

[0158] If required, convert or transform the source where blocks of data can be accessed or received from the source

[0159] Identify changed blocks on geometry adjusted block disk representation of the source device

[0160] Upload changed blocks to Doyenz

[0161] Apply said changes to a snapshotted (or otherwise differential, e.g. journal) version of raw disk representation in the cloud

[0162] Verify that said machine contains a good backup.

[0163] In this case the identification and access to changed blocks may differ between each source of machine coming into the cloud, while the transport mechanism to the cloud may remain the same.

[0164] In addition, in the above example, each provider can require different type of verification, e.g. to verify that a StorageCraft backup is succesfull one needs to perform chain verification, or boot a VM etc.

[0165] More so, each customer can utilize the pluggable interface to provide specific verifications of their LOB applications or of (their) server functions. Such pluggable verification can give customers the guarantee that their appliances are in good operating condition in case of need for failover. That ability can also create a market for third party verification providers, or third party providers of HAL/driver adjustments for windows (a process required to boot a machine on a hypervisor that was not originally built on same hypervisor or is originally a physical machine).

[0166] The decoupled process of HAL/driver adjustments allows us to match any source to any hypervisor, thus allowing doyenz cloud itself be provided by a third party or on different hypervisor or physical platform, e.g. if doyenz wishes to run appliances on a foreign (non doyenz) cloud, the pluggable nature of doyenz architecture allows us to replace the plugin that adjusts windows machines to the target's cloud hypervisor and utilize it instead of local hypervisors.

[0167] Decoupling of storage and treating all/most sources as block devices allows Doyenz the flexibility of failing back to any target. That is a customer machine may start their life as physical machine, P2C to doyenz, then failover in the cloud and run in e.g. ESX virtualized environment that Doyenz cloud currently utilizes, and then fail back to customer environment as a Hyper-V appliance. (C2V)

[0168] Universal Prerestore

[0169] A restore of a source machine is a process by which such machine becomes runnable in the cloud or otherwise made executable and accessible by the user.

[0170] To run a machine in the cloud, when run on a hypervisor, the hypervisor (or physical machine if run on physical machines) must be able to access a disk in a format it can understand, e.g. raw block disk format, and the OS on this machine needs to have appropriate hardware abstraction layer and drives to be bootable.

[0171] Since Doyenz decouples the source format from the storage format and from the execution environment, the restore itself is the process of applying such HAL and driver translation and then attaching the disk to a hypervisor VM (or to physical machine) that can then execute it. Due to such decoupling, the restore itself is uniformly applicable regardless of the source that provides the storage format that is readable by the hypervisor (or other execution environment).

[0172] Supporting multiple sources universally for a purpose of restore is therefore in part a process of providing a common disk representation regardless of source.

[0173] This is obtained utilizing pluggable architecture. For most providers, at the client side, changes on the source machine or backup would be translated by the plug-in to a list of changed blocks, and those changed blocks would then be uploaded to rCloud to be applied to the common representation, thus making such sources restorable.

[0174] Alternatively, for sources that do not implement such plug-ins at the client side, a doyenz side plug in can provide a translation layer that will provide a mountable block device representation of a backup source or an api that the upload process can utilize to otherwise access blocks.

[0175] Such plug-in can utilize e.g. third party backup provider mount driver to present the chain of backup files as a standard block based device, or alternatively do a full scan read of such chain and write the results into a chosen doyenz representation of a block device mountable by hypervisors/execution environments. In addition, doyenz plug in can accept both pull and push modes, and can therefore represent itself as a destination for a third party restore or conversion, be that destination a virtualization platform or a disk format, whereas doyenz can read the data that is being pushed to it and

transfer as blocks of data, with or without necessitating any changes in 3rd party software.

[0176] Individual File Restores on a Block Based Backup

[0177] Since doyenz utilizes decoupled storage, all backup sources are stored in a mountable block based device representation.

[0178] As long as the storage system has the appropriate file system drivers (NTFS for windows etc), the device can be mounted locally for individual file extraction.

[0179] A listing of files in the file system can either be pre-opbtained at the time of backup, or be retrieved on the cloud after the device was mounted.

[0180] A web based interface provides the listing of the files in a searchable or browsable format, where such listing is sourced either from a pre-obtained listing or online from the file system.

[0181] A user can chose a file or a directory he is interested in and the file is accessed from the mounted disk and provided in a downloadable format to the user.

[0182] Instant Availability of Backed Up Machines

[0183] Every machine in the cloud can be stored in a snapshotted chain of raw block devices, thus a restore can be a process of mounting such file system, adjusting it's hardware abstraction layer and then mounting it on a hypervisor/execution platform to become accessible.

[0184] Notably, none of the processes described above require time or processing that is necessarily related in any way to the size of the backup or source machine, and can therefore be done in constant or close to constant time, as opposed to a traditional full backup restore, the length of which is dependent on the size of the source machine or the backup files.

[0185] In addition, utilizing a cloneable COW file system, such mounting can be performed on a clone of a snapshot, thus allowing simultaneous restore from multiple restore points, simultaneous concurrent restores from the same restore point all the while continually providing new backups or other services (e.g. compaction) on the source snapshotted file systems w/o interfering with restores or requiring the restores to be queued in line past for other operations to complete

[0186] Instant Failover

[0187] A failover is a special kind of restore where machine is made to be available and running in the cloud and accessible by the customer.

[0188] Utilizing instant restore and availability, instant failover is made possible

[0189] Snapshot of the Applied Blocks Will Allow Point in Time Recovery Point

[0190] Usage of Snapshot/Clone/Copy on Write (COW) Based File Systems for Compaction/Retention Policy/Instant Spinoff of Multiple Instances for Block Based

[0191] Doyenz represents each individual volume on the source machine (or a volume on a source machine backed up by a local backup third party provider) as a single block device (or virtual disk format) accessible and mountable to a hypervisor.

[0192] Doyenz can utilize snapshot based file system, such that each backup is signified with a snapshot. When previous backup has a snapshot, we can overwrite blocks directly on the block device representation, w/o changing or modifying snapshots in any way since each change is using a COW and effectively creates a branch of the original during writes. Therefore, when a customer wants to restore, each and every saved restore point is individually available for mounting on the target hypervisor or the local OS (for e.g. file based restores).

[0193] To allow write modifications on the restored machine, Doyenz clones said FS snapshot instead of mounting it directly. Such clone operation performs another branch creation, so writes going to the block device representation can be seen in the target clone, but do not change the data on the original snapshot.

[0194] Thus an unlimited number of clones can be performed on an unlimited number of snapshots (restore points) all to be simultaneously restored.

[0195] Same mechanism allows for a deletion of individual restore points, thus compacting the space used by the chain without the need to do a full re-chain or rebasing of the backups. It is achieved by collapsing a snapshot that represents an older (or undesirable) restore point. Such operation on COW file system will cause the branched changes to be collapsed down to the previous snapshot. In case there is no difference, the change that no longer exists will not utilize any space. Since Doyenz can assign restore points to individual snapshots, a compaction is as simple as removing an individual file system snapshot on a COW file system.

[0196] Alternatives to Snapshot/COW Approach

[0197] Here and in every other parts where snapshot/COW file system is mentioned, other alternatives to achieve change tracking can also be used. For example, where snapshots are used to allow access to individual restore points, the same can be achieved by utilizing journaling mechanisms, or writing each difference in a separately named file etc.

[0198] While utilizing snapshot/COW file system may give an advantage of constant time execution on certain operation, it is not a necessary requirement for the invention, as long as each difference in restore point and in restored/executed machine representation can be individually accessed. Thus any mechanism allowing for branching of writes, including but not limited to version control systems, file systems, databases etc. can be utilized to achieve same goals.

[0199] Blocks Provider can be Generic

[0200] The Doyenz DR solution can be based on a defined generic programmatic interface which provides blocks to a consumer.

[0201] Different implementations of blocks providers can be implemented by different backup software vendors.

[0202] The blocks provider can provides a list of blocks which are the disk blocks that should be backed up and represent a point in time state of a disk

[0203] The list of blocks may be provided in the following forms:

- **[0204]** t. Full disk backup blocks
- **[0205]** u. Full disk used backup blocks
- **[0206]** v. Incremental changes blocks from previous backup

[0207] A block in the provided list of block may contain the following information

- **[0208]** w. Block offset on original disk
- **[0209]** x. Block length
- **[0210]** y. Block bytes (or enough context information to retrieve the bytes from a different location)
- **[0211]** The blocks provider should be able to provide disk geometry information (cylinders, heads, sectors, sector size)

[0212] Block size may be dynamic

[0213] z. For optimized performance the block size provided may be different and change based on various characteristics

[0214] Doyenz may accept non-block, e.g., stream based, data, i.e., any data format that otherwise can be utilized by the rest of the system.

[0215] Blocks can be pushed to a different cloud storage provider (e.g., S3, EBS)

[0216] The storage of the blocks file can be at any cloud provider which supports storage of raw files or other formats supported by the system.

[0217] The backup agent can push the raw blocks to a storage cloud and notify Doyenz DR platform to pull the backup

[0218] Doyenz DR platform can pull the blocks files from that cloud storage and perform the x2C process.

[0219] Blocks Provider can be Developed by 3rd Party and Hook into Doyenz DR Platform.

[0220] Block providers can hook to Doyenz backup agent by using defined interfaces the agent provides

[0221] This particularly means that the base agent distributable binary does not have to contain the blocks providers for a certain backup solution.

[0222] The 3rd party backup product may allow the Doyenz agent to discover it and dynamically transfer the needed binary code for the blocks provider.

[0223] Some code authenticity check can be made to ensure code validity and safety and to prevent malwares from affecting the backup.

[0224] Blocks Provider May Push/Pull the Blocks Based on Schedule or Continuously

[0225] The programmatic interface used by blocks provider can be support both pull/push:

[0226] aa. Pull: the provider can returns blocks to the consumer when requested. It can be implemented in such a way that every call returns the next block.

[0227] bb. Push: the provider can send all of the blocks to the consumer when they are available.

[0228] For resume use case—the provider can start providing the blocks from different block offset

[0229] Conversion of Other Formats, Including Tape Based, to Block Based Backups

[0230] The provider can provide blocks which are not explicitly originated from a disk based format (for example 3RD PARTY BACKUP3rd Party Backup file format).

[0231] The provided blocks can appear as if they originated from a disk based format, e.g.: have block offset, length.

[0232] Converting Backups to Raw Disk Block Devices (Online and Offline)

[0233] Processing the blocks from the backup in preparation to DR VM usually means converting them to a certain Virtual Disk format (e.g. vmdk, vhd, ami . . .)

[0234] A more generic approach is to write the blocks to a raw blocks file format based on the blocks offset.

[0235] Different hypervisors can then mount the blocks file as a device if it is expose to them in a format they support (e.g.: iSCSI, NBD, . . .)

[0236] File Formats for Multi Block Sources

[0237] The backup solution can use a file format to describe all of the blocks that needed to be applied to target VM in the cloud

[0238] That file may refer blocks from multiple sources (e.g.: raw block file, previous backup disk etc.)

[0239] This can reduces the need to upload blocks which were previously uploaded to the cloud if there is a way to identify them.

[0240] Hypervisor Agnostic Cloud

[0241] The DR solution can recover backups of machines on any hypervisor by using standard interfaces to manage the VMs (e.g. Rackspace Could API)

[0242] This can be achieved for example by using the disk blocks devices mentioned above

[0243] Plugin Based Architecture (Agent)

[0244] The agent can be based on plugins which provide dynamic capabilities to different type of agents.

[0245] The plugins can define support for different blocks provider and other capabilities and behaviors of the agent

[0246] Universal Agent with Block Providers

[0247] Somewhat covered by previous items

[0248] The agent can be shipped with predefined set of blocks providers

[0249] The agent can be remotely upgraded to support additional blocks provider based on identified machines that needed to be backed up.

[0250] 3rd part backup products can interface directly with the agent and push the blocks provider dynamically as needed.

[0251] Automatic Failback of Protected VMs (Reverse Backup, C2V)

[0252] Failback can be requested by user or otherwise initiated

[0253] Backend prepares a VM to be downloaded for failback

[0254] Agent can then downloads the VM and deploys to specified target

[0255] Agent may coordinates with backend to automatically provide deltas of the running DR VM to complete the failback on customer site.

[0256] Backend shuts down the DR VM when it has the right conditions have met (e.g. can determine that the time to transfer the next delta went under a certain threshold)

[0257] Agent can applies the deltas at customer at start the VM back on customer site

[0258] Files Block Based Backup

[0259] Block based backup concept should not be limited to full disk backups

[0260] It can be possible to implement block based backup for specific files/paths on a file system

[0261] Using file system driver the backup provider can trace write to certain files and save changed blocks information

[0262] Backup blocks provider for file based backups provides the blocks of the changed files

[0263] There could be additional mechanism tracks file meta data changes like ACLs, attributes etc.

[0264] Change Blocks Detection

[0265] Significance

[0266] Cloud DR solution may upload backups of incremental changes based on the customer recovery point schedule.

[0267] Since in many cases only a WAN link is available between the customer and the Cloud datacenter minimizing the uploaded size can significantly improve SLA (for example—meet a daily recovery point protected in the cloud).

[0268] In order to upload only incremental block changes a block change detection mechanism can be implemented.

[0269] Some of the approaches for detecting changed blocks are described below.

[0270] Using Backup Product Changed Blocks Tracking APIs

[0271] Some backup products provide APIs which can be used to retrieve a list of changed blocks from a certain point in time.

[0272] For example VMWare provides a set of APIs (vStorage API, CBT) for that purpose

[0273] Even when such specific APIs exists—limitations to their functionality may cause them to provide a super-set of all changed blocks (e.g.: vStorage API CBT might in some cases provide a list of all blocks on disk instead of just the blocks which were changed). Therefore in order to minimize upload size a dedup mechanism can be applied as well.

[0274] Comparing Mounted Recovery Point to Signature

[0275] In some cases the information of which blocks have changed on a disk is not directly available to Doyenz backup agent (e.g. StorageCraft ShadowProtect backup files, Acronis True Image backup files, backups which create VMWare vmdks etc). This is because the blocks information is stored in proprietary backup files with no programmatic which support accessing the changed blocks directly.

[0276] In many of those cases it may be possible to mount the recovery point file chain as raw blocks device (e.g. for StorageCraft ShadowProtect it is possible to use SBMount command, VMWare vmware-mount.exe can mount different vmdk types).

[0277] As mentioned above—if a signature file is created for a backup it can be possible to perform changed blocks detection by comparing all blocks on a mounted raw disk it is wished to be backed up.

[0278] Since this involve scanning of all disk sectors the process will be dependent on fast IO available to the scanning code.

[0279] An optimization for this could be scanning only sectors that contains used data. This could be obtained by accessing specific file system APIs and retrieve used blocks information (e.g. for NTFS it is possible to use $Bitmap file as a source for used blocks).

[0280] Tracing Writes to Virtual Disk

[0281] Some disk backup products have the capability of generation VM virtual disks (e.g. ShadowProtect's Head-Start)

[0282] This capability can be used by Doyenz agent to trace information about the blocks as they are written to the virtual disk by the backup product. Example of such information can be block offset, block length or even the blocks data.

[0283] Capturing blocks as they are written can be done in different way. Following are examples:

- **[0284]** cc. Using file system filter driver which traces the write to certain destination
- **[0285]** dd. Create a custom virtual file system and direct the Virtual Disk generation to it.

[0286] The virtual file system will proxy writes to the destination file while capturing the blocks information.

- **[0287]** ee. Hook the backup product APIs used to write to the Virtual Disk and capture block information during the write.

[0288] In case block data (the actual bytes) were not captured—a secondary phase can be used to read the blocks from the Virtual Disk by mounting it using Virtual Disk mounting tools (for example VMWare VDDK).

[0289] Changed block detection in this case can be done for example by utilizing a previous backup signature file (compare digest of block against digest at signature file offset) or any other more sophisticated de-duplication technique mentioned in other documents.

[0290] Tracing Reads from Mounted Backup Files Chain

[0291] One of the challenges is determining the changed blocks in proprietary backup files chain (like for example a chain of backups from ShadowProtect, Acronis True image)

[0292] A possible approach could be to use a backup chain mounting tool to mount the chain as a raw disk device

[0293] The next step then can be to perform a scanning of the new device by reading each block on the disk

[0294] Using a file system filter driver to trace all reads from the file it may be possible to correlate between the blocks read from the disk to a backup files in the backup chain

[0295] Once the blocks for each file have been detected they can be used as blocks for a blocks provider

[0296] The agent can then upload only the blocks that are referenced by an incremental backup file

[0297] Emulating a Hypervisor Product

[0298] Some backup products have the capability of creating a VM by connecting to a Hypervisor3RD PARTY.

[0299] In order to perform changed blocks detection it may be possible to emulate the Hypervisor by creating a process which implements the protocol the Hypervisor uses. For example ESX emulation can implement the vSphere APIs and VDDK network calls in order to intercept the calls from the backup software.

[0300] The emulator can either simulate results to the caller or to proxy the calls to a real Hypervisor and proxy back the reply from the Hypervisor.

[0301] While the backup product performs writes to Virtual Disks—the emulator can capture the block information and written data in order to generate changed block detection.

[0302] The blocks can by de-dupped to avoid capture of pre-uploaded blocks to Doyenz datacenter.

[0303] Many of the different mentioned dedup techniques can be used in this case as well.

[0304] Other Methods

[0305] Other methods of obtaining change data, including but not limited to interception, integration, introspection, or instrumentation may be used.

[0306] All or some of the data may be obtained using any of the alternative methods.

[0307] Any number of the alternative methods may be combined and used together or alternatively.

[0308] Transmission Layer Deduplication

[0309] Transmission layer deduplication is an approach where there may be a sender and a receiver of a file, whereby the sender knows something about data that is already present on the receiver, and as a result, may only need to send:

[0310] Data that represents something unknown to the receiver

[0311] Data location information such that the receiver knows where to place blocks of data (either received from the sender, or retrieved locally) in order to reconstitute the target file.

[0312] The idea is that the file (or files) may be either lazily or eagerly reconstituted at some point in time after the transmission is complete. In the case of eager reconstitution, the file may be reconstituted prior to saving and reading (although it may be reconstituted into a reduplicated storage). In the case of lazy reconstitution, only the new block and loca-

tion information data may be saved, and the file may be dynamically reconstituted from the original sources as the file is read.

[0313] Block Level Deduplication and Block Alignment

[0314] Deduplication may be performed on the basis of blocks within the file. In this approach, a fingerprint may be computed for each block, and this fingerprint may be compared to the fingerprints of every other block in the file, and to fingerprints of every file in the reference set of files. With a naive and rigid fixed size block approach, it is possible to miss exact matches because the reference block may be aligned against a different block boundary. Although choosing a smaller block size may remedy this in some cases, another approach is to use semantic knowledge of how blocks are laid out in the files to adjust block alignment as necessary. For example, if the target and reference files represent disk images, and the block size is based on file system clusters, the alignment should be adjusted to start at each of the disk image's file system's cluster pools. This may cause smaller blocks just prior to a change in alignment.

[0315] File Change Representation is Calculated Before Uploading and Verified when Applied

[0316] A file's signature itself does not need to be transferred as part of the upload. Since the sender knows something about the files on the receiver (through the signature), it can build a change representation that only:

[0317] Contains new data

[0318] References existing data on existing files

[0319] This representation can be computed and transferred on the fly. This means that the representation may not be known before the transfer begins.

[0320] The integrity of the representation can be verified by:

[0321] sprinkling checksums within the representation

[0322] appending the representation with an information block that contains:

[0323] ff. A magic number

[0324] gg. The size of the representation

[0325] hh. The checksum of the representation

[0326] or other means.

[0327] On apply, (assuming the starting file is a clone of the previous version of the same file) the representation may instruct the receiver to do a combination of one or more of the following steps

[0328] Leave a block in place

[0329] Replace a block with an existing block from a different file

[0330] Replace a block with an existing block from the same file

[0331] Replace a block with another from the representation itself

[0332] Signature Calculation

[0333] File Signatures may be calculated in many different fashions. For example, signatures can be computed for blocks in flight, or they may be computed on blocks laying static on a disk. Also, they may be represented in many different fashions. For example, they may be represented as a flat file, a database table, or in optimized hybrid data structures.

[0334] Canonical Compacted Signature Computation

[0335] A compacted signature includes a fingerprint and an offset for each non-zero block in the file being fingerprinted. In this case, the block size can be omitted because it is implicit.

[0336] One possible approach to computing a compacted signature is to start from the beginning of the file, and, using whatever semantic knowledge that is available, align with logical blocks in the file. For each logical block, compute the fingerprint. If the fingerprint matches the fingerprint of a zero block, do nothing. If it matches the fingerprint of a non-zero block, write out the start of block offset, and the given fingerprint.

[0337] Dynamic Fingerprinting

[0338] Fingerprints can be computed for individual blocks, or for runs of blocks. A fingerprint for a run of blocks is the fingerprint of the fingerprints of the blocks. This can be used to identify common runs between two files that are larger than the designated block.

[0339] An example of this approach:

[0340] When a match found, store the fingerprint, and track the offset and size

[0341] If the next block constitutes a match, check to see if it matches the next block in the previous version. If so increment the size and incorporate the next block's fingerprint into the larger fingerprint

[0342] Continue until a next block in the current file no longer matches a next block in the previous file.

[0343] Concurrent Signature Calculation on Sender and Receiver Sides

[0344] Both the sender and the receiver can have a representation of the final target file (such as a bootable disk image) on the completion of a transfer. In the case of the receiver, the representation can be the file itself. In the case of the sender, the representation can be the signature of the previous file, together with the changes made to the signature with the uploaded data. With this data, an identical signature of the final file can be computed on both sides, without having to transfer any additional data. On the sender's side, the signature can be computed by starting with the original signature, and modifying it with the fingerprints of the uploaded blocks. In the case of the receiver, the signature can be computed the same way, but it can also be periodically computed by the canonical algorithm of walking the file. In any case, it is valuable to have a compact method for determining that the signatures on both sides are identical. This can be done by computing strong hash (such as MD5 or SHA) on segments of both signatures, and comparing them.

[0345] Generational Signatures for Reliable Sender Side Signature Recovery

[0346] During an upload, a sender may deal with two signatures for each file:

[0347] The signature of the previous version of the file

[0348] The signature of the new version of the file

[0349] The sender may use the signature of the previous version to identify matches that do not need to be uploaded, and generate the signature of the current version to assist in the next upload. On completion of an upload, the receiver may need to verify the integrity of the uploaded data. Once it is verified, the sender can delete the signature of the previous version and replace it with the signature of the current version. If anything goes wrong with verification, the sender may need to use the signature of the previous version to re-upload data.

[0350] The sender may verify a file's signature before using it (by comparing strong hashes as described above). If the signature is incorrect, it can be supplied by the receiver, either in part, or in its entirety. In some cases, the on the receiver side may be reorganized (for example, by changing the finger print

approach, or fingerprint granularity), which would invalidate all existing signatures. In any such cases, a correct signature can be re-computed on the receiver via the canonical approach.

[0351] Generational Signatures for Reliable Agent Side Signature Recovery

[0352] The agent may store a local copy of fingerprint file which it scans to determine which blocks require to be uploaded. However, when uploading blocks, the client may need to updated said file. In case of transmission error or a full upload failure, the client may then need to recover itself back to a state that is comparable to that of the server. This will be achieved by one of two approaches:

[0353] 1. The updated hashes that may be transferred to the server may be kept in a local (client side) journal and only applied to the main file once a validation of successful upload has been received from the server

[0354] 2. A new full fingerprint file may be created for each or some uploads. Old file may be deleted upon receiving a confirmation from the server that the upload is successful and current full hash on the server matches that on the client. (Generational)

[0355] Efficient Signature Lookup

[0356] In most cases, uploads may be for small changes to very large files. Since the files may be very large, their signatures may be too large to be read into physical memory in their entirety. In order to balance memory usage, a single strategy my work, but a hybrid approach may be also used for fingerprint lookup. For example, an approach might involve a combination of:

[0357] Caching the signature of a zero block

[0358] Caching the signature of commonly referenced blocks

[0359] Optimistic signature prefetching

[0360] Tree based random lookup

[0361] Optimistic Signature Prefetching

[0362] In most cases, the next version of the file being uploaded will share much of the same layout as the previous version. This means that in the common case, the signature of the current may be very similar to the signature of the previous version. To leverage this, the representation builder may fetch signatures for comparison (from the signature of the previous version of the file), from the portion representing the fingerprints of blocks slightly before the current checked offset, through blocks that fall a small delta beyond this. The representation builder can maintain a moving window, and fetch chunks of fingerprints as needed front he previous version In most cases, a fingerprint should match either a zero fingerprint, or a fingerprint in this prefetch cache. When there is no match, the new blocks fingerprint can be, or may need to be, checked against some or all fingerprints for the previous version.

[0363] Tree Based Random Lookup

[0364] In cases where a random fingerprint lookup is required, the representation builder can use a tree based approach. An example of this:

[0365] The signature file is sorted

[0366] Duplicate fingerprints are eliminated

[0367] An in memory datastructure is built that contains the first n bytes of a signature, and the offset in the file where fingerprints with this prefix begin.

Lookup then amounts to:

[0368] Do a hash lookup on the first n bytes of the target fingerprint against the above data structure (if there is no match, then the signature doesn't match any in the previous version)

[0369] Load the segment of the file that represents fingerprints with this prefix into memory

[0370] Do a lookup against the loaded segment.

[0371] Secure Multihost Deduped Storage/Transport

[0372] Blocks may be encrypted as they are written to storage. An index may be maintained to map the fingerprint of an unencrypted logical block to its encrypted block on a file system. Blocks can be distributed among storage facilities at various levels of granularity

[0373] Block by block

[0374] Logical files remain in place on a single storage host

[0375] Logical groups of files remain in place on a single storage host.

[0376] Unlimited Scalability

[0377] Storage in such a fashion can be scaled without limit. With a block level granularity, each new block can be written to a storage host with the most available space. With less granularity (i.e., files) data sets can be migrated to different storage hosts.

[0378] Pre-Balancing Larger Grained Distribution

[0379] In the case of larger grained distribution (e.g. file based) the load balancer may not know how large the unit will end up in advance. Series of uploads can grow a distribution unit well beyond its initial size. This means that a pre-balancing storage allocator for this level of granularity may make predictions about how large each unit will grow before allocating storage to it.

[0380] migrations

[0381] In some cases, larger grained distribution units may grow to be too large for their allocated host. In this case, they may be migrated to a different host, and metadata referring to them may be updated.

[0382] Service/Application Level/Grain Restore on Block Based Backup

[0383] To restore a service based on a block based backup the following steps may be used

[0384] Apply the backup to a disk image

[0385] Mount the disk image and collect the files and meta-data representing data for the given service

[0386] Perform any necessary transformations to the files to make them compatible with a target service (e.g., different versions of the same service, or different services performing similar functionality)

[0387] Instantiate the new service with the previously collected files and meta-data

[0388] Block Based Backup Using Command Line Tools

[0389] Blocks for a backup may be obtained using command line tools such as dd, which can be used to read segments of raw disk images as files. One approach to this would be to have the backup sender either resident on the system, or remotely logged in to the system that has the target files (for example, the supervisor of a virtualization host, such as ESX). The command line tool would then be run to read blocks to the sender. This could be optimized through a multi-phase approach such that the command line tool is actually a script that invokes a checksum tool on each block, and makes decisions on whether to transfer blocks based on whether the sender might need them. For example, the script could have

some minimal awareness of the signature used by the client (e.g., fingerprints for zero blocks, and a few common blocks).

[0390] An advantage of this approach is that it can be used in environments where the system that has direct access to the files to be transferred does not have the resources to run a full sender.

[0391] An alternative includes naive implementation of a signature file. I.e.: flat file of digest per block offset (including empty blocks). The file size is the (disk size/block size)* digest size.

[0392] Blocked Based Architecture

[0393] The goal is to build a generic architecture which can enable cloud recovery in a generic way independent of the backup provider.

[0394] A backup provider provides blocks to backup per backed up disk (ideally only changed blocks)

[0395] Blocks source dedups the blocks and upload them to the cloud

[0396] Upload service stores the blocks on LBS in a generic file format

[0397] Store Service applies the blocks to a vmdk when backup is complete

[0398] VMDK can be booted in an ESX hypervisor

[0399] Future strategy could even abstract the persistent file format and store everything as raw disk bytes and then it will become hypervisor neutral.

[0400] Goal

[0401] Define file formats to be used for block based back-ups, transfer and apply. The files will be effectively used to ensure minimum number of blocks will be uploaded by using signatures and other dedup techniques.

[0402] Note: current focus is not deduping since this may be required only for 3RD PARTY Windows backup altough the proposed design addresses this but does not give full details for implementation.3RD PARTY.

[0403] Approach for Block Based File Format

[0404] This format may include one or more of the following:

[0405] Have a reference file

[0406] Have multiple sources of blocks

[0407] Describe only the blocks that are different (or in different positions) than they are in the reference file

[0408] Describe, for each block in the target file that is different, where to find the block in one of the block sources.

[0409] Include internal validation (i.e., if a file becomes corrupted, there should be a check that finds this without requiring any external data)

[0410] Ensure integrity of the source files

[0411] Incrementally transferred while reading from sources (no need to buffer it before uploading)

[0412] Support version to allow file format changes and extensions

[0413] Should be compact (significantly smaller than uploaded blocks)

[0414] Support upload resuming in case of interruption

[0415] Files Usage Scenario

[0416] We need to be able to handle the following example cases:

[0417] current—the file being backed up from the client and is written to the primary storage (usually vmdk)

```
case 1: // block is same as previous at the same offset
 current |------------!a!-----------------|
 previous |------------!a!-----------------|
 case 2: // block was seen at a different offset in previous
 current |----!a!------------------------|
 previous |----!b!-----!a!----------------|
 case 3: // block wasn't seen at all in previous
 current |----!c!------------------------|
 previous |----!b!-----!a!----------------|
```

[0418] previous—the previous version of the file backed up and snapshotted on the primary storage

[0419] Example pseudo code for usage on client agent side

```
prep:
// is the current signature valid?
if (no local signature or signature.hash != backend.signature.hash)
get fresh signature from backend;
for (block : blocksFromProvider)
{
handle(block);
}
handle(block)
{
h = md5(block.data);
if (signature.getHashAt(block.offset) == h)
{
// nothing to do - block is the same as previous one
(update stats and progress only)
}
else
{
// check if we have seen this block earlier
prev = blocksIndex.get(h);
if (prev != null)
{
assert (prev.offset != block.offset); // if false then blockIndex is out of sync
// we have seen this block in a different offset
write block meta to BU_token.blkinfo;
signature.update (block.offset, h);
}
else // this is a new unseen block
{
write block meta to BU_token.blkinfo;
signature.update (block.offset, h);
blocksIndex.update (h, block.offset);
write raw block bytes to upload stream file (BU_token.blkraw);
}
}
}
```

Upload BU_token.blkinfo

Design

Terms Definition

[0420] Reference file—a file which represents the currently backed up disk device in the cloud (e.g. "/NE_token/diskl. vmdk")

[0421] Blocks Source file—a file which contains blocks used as source of block information in the blocks file (e.g. the previous vmdk, "/NE_token/diskl.vmdk@BU_token")

[0422] High level

[0423] The solution may use several files:

[0424] Raw Blocks File

[0425] ii. Contains consecutive raw blocks that needed to be applied to the current backup.

[0426] jj. The file can be generated and uploaded directly without being persisted to the local customer storage.

[0427] kk. For manual import the file can be generated into the import local drive

[0428] Block Changes Info File

[0429] ll. meta information about each block uploaded in the Raw blocks file

[0430] mm. Will be used by the backend to apply uploaded blocks to the correct target location.

[0431] Blocks Signature File

[0432] nn. A file which contains a checksum (md5) for each 4K block offset on the disk

[0433] oo. Used to check existence of a block before uploading it to reduce upload size in the common case

[0434] Blocks Hash Index File (Aka "Transport Dedup", "Rsync with Moving Blocks", "d-Sync", "Known Blocks")

[0435] pp. In order to determine if block bytes needed to be uploaded a fast index of block hashes is required.

[0436] qq. The index may be big and not fit in customer memory and therefore needs to backed by a disk file.

[0437] rr. The index will be cached locally at the customer and can be recreated from the signature file if needed.

[0438] Example Raw Blocks File Format

[0439] File name suffix

[0440] blkraw

[0441] Binary format

[0442] The file is a binary file

[0443] Byte ordering—Network Byte Ordering (Big-Endian)

[0444] File structure

[0445] Simple raw blocks laid out consecutively in the file.

[0446] |- - block0- -|- -block1- -|- -block2- -| . . . |- -blockN- -|

[0447] 4 KB 4 KB 4 KB 4 KB

[0448] Example Block Changes Info Format

[0449] File name suffix

[0450] blkinfo

[0451] Binary format

[0452] The file is a binary file

[0453] Byte ordering—Network Byte Ordering (Big-Endian)

[0454] File structure

[0455] General Layout

| |--header--|--src file/s info--|----------changed blocks info----------|

[0456] Header

Length:16B	
|--magic--|--version--|
int64	int64
magic: 0xd04e2b10cdeed009	
version: 0x0001	

[0457] Source Files Information

Length:4B + N*1KB				
|--files count--|--file1--------|--file2--------|....|--fileN--------|			
int32	1KB	1KB	1KB	

[0458] Source File Info Block

Length:1KB			
|--file md5--|--file ID--|--file name----------|		
16B	int32	1004B	

[0459] Block Information

Length:36B			
|--src file ID--|--offset src--|--offset ref--|--block md5--|			
int32	int64	int64	16B
src file ID: the ID of the file defined after the header			
offset src: offset in bytes on the source file (usually the raw blocks file)			
offset ref: offset in bytes on the reference (target) file.			

[0460] Sizing

[0461] Assume:

cluster size of backed up disk: 4 KB

Hash: MD5 (128 bit/16B)

Block info size: 36B

[0462] Uploaded size per 1 GB

1 GB/4 KB->262144 blocks->

blockInfoSize * 262144=36B * 262144=9437184B=9 MB per GB

[0463] 100 GB used space would max to 900 MB (max because dedup would reduce it)

[0464] Example Signature File Format

[0465] File name suffix

[0466] blksig

[0467] Binary format

[0468] The file is a binary file

[0469] Byte ordering—Network Byte Ordering (Big-Endian)

[0470] Format options:

[0471] Flat Signature File

[0472] ss. Just md5 at corresponding offset that can be directly accessed by calculating offset in the file

[0473] tt. Unused zero blocks will also contain the signature

[0474] uu. Pros: very simple to implement and maintain (create, read, write)

[0475] vv. Cons: file size is big (4 MB per 1 GB of volume size) since it must contain all empty blocks.

[0476] Sparse Signature File

[0477] ww. Like the flat file but empty blocks hashes are not stored

[0478] xx. Pros: the file takes small amount of disk space—only the used blocks hashes. (4 MB per 1 GB of used size)

[0479] yy. Cons: implementation complexity—downloading the file from backend may require sparse download.

[0480] Compact Signature File

[0481] zz. File is compressed by containing offset:md5 pairs where zero blocks are skipped

[0482] aaa. Pros: the file is very small and compresses well on consecutive equal data and zeros.

[0483] bbb. Cons: no way to write into the file of new block since it will affect the compaction, therefore during backup a new delta signature file must be created and post backup will have to collapse the original with the delta to be the new signature. This process will have to accurately repeated on the backend side.

[0484] Example Index File Format

[0485] The requirement is to be able to do fast lookup of block offset given an md5 hash.

[0486] Possible Data Structures

[0487] B+Tree or a just use a database which effectively creates a B/B+tree on a table index.

[0488] Disk based hash table—flat file with hash collission buckets at constant offsets which need to be resized when a bucket gets full. The file should be mmap-ed for better performance.

[0489] Issues

[0490] B-tree drawback is that is suffer from fragmentation for the type of data we intend to use.

[0491] A mitigation strategy for this is creating pages with small fill factor which should reduce fragmentation till pages start to get full.

[0492] The hash table suffers from the need to rehashing when buckets get full.

[0493] So essentially both solutions suffer from similar problem and the choice should most likely be based on ease of implementation.

[0494] Design

[0495] Create an empty index

[0496] Insert/lookup index during backup

[0497] If need rebuild parts of the index while waiting for chunk upload to complete or rebuild all if must.

[0498] On the post backup signature processing—while rebuilding the new signature from repopulate the index with big fill factor so it would be ready for next backup.

[0499] Notes

[0500] If index get corrupted/missing—it can be rebuilt from the signature file like in step 4.

[0501] An optimization would be seed an index at the backend with known blocks for target OS/apps and send to client before backup start. This might have potential to reduce initial # upload size by 10-20 GB per server.

[0502] We can consider thinking if there is a similar data structure or enhancement to the current 2 options which will allow partial rebuilding of the index instead of full rebuild every time it is needed.

[0503] Alternative Approach

[0504] Create a file with sorted blocks hashes (md5) from the signature file

[0505] Build a Trie on top of the sorted hashes file

[0506] Maintain an in-memory block index (hash table or such) for new blocks

[0507] During backup lookup block in in-mem storage and then in the Trie.

[0508] Post backup processing will have to rebuild the sorter blocks hashes files by doing a merge from original file and the in-mem structure.

[0509] Design and Implementation Notes

[0510] ccc. the block info may be combined with raw bytes upload for simplicity (it was debated if that really is simpler or not)

[0511] ddd. Alternatives to MD5 as the fingerprinting algorithm can be used. SHA-X may be better for performance reasons (although the hashing is the least of the problem from time consumption perspective, JO is much bigger).

[0512] eee. support for different blocks sizes from the same block provider is an option

[0513] fff. For recoverability—generational signature files may be used. This may be needed in case backup gets aborted before completion—without it the sig file may become out of sync.

[0514] ggg. The support for multiple files may be an optional optimization initially.

[0515] Default single block source and since default target (e.g.: previous vmdk and single raw blocks source) may be used as an option.

[0516] hhh. "capabilities APIs" may be used where the blocks provider will have to match to certain backup capabilities (sending different block sizes, non-block aligned offsets, etc)

[0517] iii. The terms reference file, source file may alternatively be replaced by

[0518] i. reference file->destination file

[0519] ii. source file->server blocks file

[0520] BlocksTool

[0521] example utility

[0522] Blocks tool is a tool that used to test block based operations that are performed by the block based framework.

[0523] As new functionality is created and added to block based backup the new code could be tested using this tool.

[0524] Usage

```
$ java -jar Blocks Tool.jar
Usage: java -jar BlocksTool.jar <action> <options>
--backup -cbt <cbt_xml_file> -srcvmdk <vmdk_file>
[-sig signature_file] [-path
files_path]
--apply -srcraw <source_blkraw_file> -srcinfo
<source_blkinfo_file> -target
<target_vmdk>
```

[0525] backup input file:

[0526] cbt_xml_file: CBT info file in the format created by 3RD PARTYagent

[0527] vmdk_file: flat ESX vmdk file used as source for point in time backup

[0528] backup output files:

[0529] blkraw: raw blocks to upload

[0530] blkinfo: blocks information (refers to the blkraw file

[0531] blksig: blocks signature file of backed up disk.

[0532] Example:

```
blockstool.sh --backup chgtrkinfo-b-
w2k8std_r2_x64_1.xml w2k8std_r2_x64-
flat.vmdk
```

[0533] Backup

[0534] Creates block based backup files from source flat ESX vmdk (not the one created by 3rd party!) and a CBT information in XML format that 3RD PARTYagent generates. Additional signature file is created unless passed a specific signature file from previous backup.

[0535] Apply

[0536] Performs blocks based copy from the block based backup files of all blocks into a target destination flat ESX vmdk file.

[0537] Example Usage

```
$ java -jar Blocks Tool.jar --backup -cbt
F:\\tmp\\blocks\\full\\chgtrkinfo-b-
w2k8std_r2_x64-000001-28-11-2011-09-59.xml -srcvmdk F:\\tmp\\b
locks\\full\\w2k8std_r2_x64-flat.vmdk -path f:\\tmp\\blocks
Performing Backup:
cbtXmlFile = F:\tmp\blocks\full\chgtrkinfo-b-w2k8std_r2_x64-000001-
28-11-
2011-09-59.xmlsourceVmdkFile = F:\tmp\blocks\full\w2k8std_r2_x64-fl
at.vmdk
sigFile = f:\tmp\blocks\7c537730-3615-476d-aa96-03b6dcc1f3cb.blksig
rawBlocksFile = f:\tmp\blocks\7c537730-3615-476d-aa96-
03b6dcc1f3cb.blkraw
blocksInfoFile = f:\tmp\blocks\7c537730-3615-476d-aa96-
03b6dcc1f3cb.blkinfo
..
$ java -jar BlocksTool.jar --apply -srcraw F:\\tmp\\blocks\\
11ff07ad-87b6-4db6-
872f-b33ff01c48bb.blkraw -srcinfo F:\\tmp\\blocks\\11ff07ad-87b6-
4db6-872f-
b33ff01c48bb.blkinfo -target F:\\tmp\\blocks\\target_restored.vmdk
```

[0538] Example Generic Block Based Agent Class Design

```
Example implementation:
class BlockInfo
{
long offset,
long length;
byte[ ] data;

}

interface BlocksReader
{
BlockInfo readBlock (long offset, long length);
}

// reads blocks from a vmdk using vddk

class VddkBlocksReader implements BlocksReader

// reads blocks from ESX cbt snapshot point
class VadpBlocksReader implements BlocksReader

// reads blocks from raw mounted windows disk block device
class RawDeviceBlocksReader implements BlocksReader

interface BlocksProvider implements Iterable<BlockInfo>
{
```

-continued

```
Iterator<BlockInfo>iterator( );

}

// opens vmdk from IMG*x backup path and uses change blocks
xml from 3rd party3RD PARTY
class 3rdPartyVmdkBlocksProvider implements BlocksProvider
{
3rdPartyVmdkBlocksProvider (
String vmdk,

String changedBlocksXmlFile,

VddkBlocksReader reader)
...
}

// opens local vmdk generated by 3rd Party convert and reads blocks
class BeWinVmdkBlocksProvider implements BlocksProvider
{
BeWinVmdkBlocksProvider (

String vmdk,
byte[ ] writtenBlocksBitmap, // captured using vddk hooking
VddkBlocksReader reader)
...
}

// uses VADP APIs to get the changed blocks from ESX vmdk
class VADPBlocksProvider implements BlocksProvider
{
IVADPBlocksProvider (
ESXConnection con,
String vmdk,
BackupContext ctx // the snapshot sequence id etc.
)
}

// mount v2i files chain and reads blocks from mount
class 3RDPARTYv2iBlocksProvider implements BlocksProvider
{
3RDPARTYBlockProvider (
String v2iFile,
byte[ ] writtenBlocksBitmap,

RawDeviceBlocksReader reader)

...

}

// mount tib files chain and reads blocks from mount
class AcronisBlocksProvider implements BlocksProvider
{
AcronisBlocksProvider (

String tibFile,
byte[ ] writtenBlocksBitmap,
RawDeviceBlocksReader reader)
..
}

// sbmount sp files chain and reads blocks from mount

class SPBlocksProvider implements BlocksProvider
{
SPBlocksProvider (
String spFile
byte[ ] writtenBlocksBitmap,
RawDeviceBlocksReader reader)
...
}

// mounts VSS snapshot and read blocks from mount
class VSSBlocksProvider implements BlocksProvider
{
```

-continued

```
VSSBlocksProvider (
Guid shadowId,
byte[ ] writtenBlocksBitmap, // captured somehow, VSSProvider?
RawDeviceBlocksReader reader)

...
}

// mounts VHD and reads blocks from mount / use hv snapshots?
class HyperVBlocksProvider implements BlocksProvider
{
HyperVBlocksProvider(

?
)
...
}
```

[0539] Example Usage3RD PARTY:

```
BlocksProvider p = new 3rdPartyVmdkBlocksProvider (
"e:\backups\IMG00002\disk1.vmdk",
"cbt_file.xml",
new VddkBlocksReader(".\vddkBlocksTool.exe", cmdExecutor),
);

Iterator<BlockInfo> it = p.iterator( );
while (it.hasNext( ))
{
BlockInfo b = it.next( );

blocksHandler.handle (b);

}
```

[0540] Manual Onboarding

[0541] Intake Device

[0542] As one of the steps of transferring machine sources from the customer and to the cloud, Doyenz have developed a method and built an apparatus that can be used to transfer customer (or any other) source machines on physical media.

[0543] In one example embodiment of the intake apparatus, the physical media is standard hard drives.

[0544] The Copy Agent

[0545] In this device, the doyenz agent can utilize it's plugin architecture to perform all standard steps of identifying machine configuration, getting source blocks or source files etc, but where a transfer plugin differs from a standard plugin. This "manual intake" aka "drive intake" transfer plug in substitutes uploading of the data to the cloud with copying the data to a destination disk. The plug in can be a meta-plug in that has two functionalities combined—on one hand the copying of the data to a physical media, and on another hand a plug in used usually on the cloud side of the Doyenz cloud that can ensure that the data written to disk can be formatted and stored in the same way a doyenz upload service in the cloud would have stored it in the transient live backup storage (a transient storage that can be used to store uploads before they complete and ready for application to the main storage)

[0546] The agent further comprises

[0547] an integration service that upon request from the user can generate a shipping label using either user's or Doyenz shipping account with a standard shipping service, update the disk with the shipping number and unique id that would allow Doyenz to identify the disk with the shipping.

[0548] an integration service that integrates with Doyenz (or the business that operates doyenz based cloud) CRM and sales system thus tying in and referencing the support/crm/sales ticket with the process of manual onboarding and putting enough identification information on the shipped disk to make such integration identifiable by the intake apparatus

[0549] The act of copying the data to the disk, shipping it and then copying to the cloud is generally faster than a direct upload (depending on bandwidth and other factors.), however, it introduces a delay for the time that the disk is in the shipping and processing. The agent may be able to utilize such delay by starting an upload of next backups even before the original on disk backup was applied in Doyenz. This can be achieved by maintaining ordered list of backups and corresponding files and sources and being able to reorder the application of such uploads on the cloud side.

[0550] The Drive Intake Apparatus

[0551] On the cloud side, the drive intake apparatus may be comprised of a computer system with a hot-swappable drive bays attached to disc controllers. On said device, a special intake service is running. The service comprises of the following mechanisms:

[0552] 2. A detection mechanism can beused to detect drives as they are inserted into the bays

[0553] 3. A mechanism can beused to identify drives and the backups on them, and thus know whether the drive was already processed or not

[0554] 4. A mechanism that can transition or trigger the rest of the system to think that the backup or upload are fully uploaded and are ready to be applied to the main storage

[0555] 5. A mechanism that can forensically or simply wipe the disk and make it available for reuse upon completion of the "upload".

[0556] 6. A monitoring console that displays all existing drive bays and displays whether they contain valid uploads, whether those uploads are in the process of being applied to the main storage and whether the intake apparatus is done with a particular drive. A user of the console has indication whether the drive is ready to be taken back into circulation (or sent back to customer if originated from the customer) and which bays are available for use.

[0557] 7. A database structure (or other configuration structure) that presents each bay as a standard doyen live backup system and therefore allows the rest of the system be decoupled and not require specific knowledge whether the source came from upload or was sent in a mail by a customer

[0558] Backup Software Integration

[0559] This entire section of the document is one possible implementation of the general system. The section refers to specific 3rd party software as examples only. Other combinations of software and alternative implementations exist.

Solution Proposals

[0560] Customer side Incremental VMDK based: Note: we have since learned that they pulled support for vmdk generation without a esx host

[0561] jjj. Snapshot Approach:

[0562] i. Artificially set snapshot before writing incremental to VMDK (by altering text in VMDK file) so that writes go to a delta file instead of the flat file

[0563] ii. Send the deltas to doyenz,

[0564] iii. Fake a vmsd

[0565] iv. Perform apply as with esx/vsphere backups

[0566] v. Collapse snapshot at client side via one of many possible fragile approaches:

[0567] 1. Get change blocks from DC

[0568] 2. Mount vmdk twice—once with the delta and once without. Merge changes from the delta mounted one to the non-delta mounted one. Needs validation to ensure that the same flat file can be mounted from two different vmdks without causing problems.

[0569] kkk. File tracing approach

[0570] i. Trace writes to the VMDK to identify the change blocks.

[0571] Dedup server based & Doyenz side incremental VMDK or traditional restore

[0572] lll. Synchronize a dedup server at customer site and at doyenz, and try to generate incremental vmdks from that server

[0573] mmm. Run 3rd party+Dedup server at Doyenz, none at customer site, and have customer site agents send directly to doyenz (using client side dedup)

[0574] nnn. Use client side dedup, client side 3rd Party (for local backups) and set Doyenz up as an OST dedup server, try to generate incremental VMDKs from that.

[0575] ooo. Syncronize 3rd Party/Dedup at customer site with a Doyenz build 3rd Party dedup solution receptical (to make it multi-tenanted and reduce the memory requirements—it doesn't need to dedupe amongst clients) and feed this data to a 3rd party server to do VMDK based restores.

[0576] 3RD PARTY 3rd Party Approach Investigation and Progress

[0577] Basic Technical Requirements

[0578] Online seeding

[0579] Backup Upload

[0580] Manual seeding

[0581] Storage/Storage management

[0582] Trial restores

[0583] Failover

[0584] Failback.

[0585] Complications in Backing Up from 3rd Party

[0586] Backups are all written in tape format, to actual tape, or to 3RD PARTY BACKUP if the data is written to disk

[0587] The tapes represent files, not disk images

[0588] Incremental 3RD PARTY BACKUPs are big because they contain the entire contents of any files that were changed.

[0589] Lack of 3rd Party deletion tracking requires frequent rebasing

[0590] Customer upload bandwidth is not expected to be significantly better than the current approach

[0591] Solutions Diagram

[0592] Transport Options

[0593] Direct Upload of 3RD PARTY BACKUPs

[0594] We can build a custom agent that uploads 3RD PARTY BACKUP files. Implementation may involve detecting the 3rd Party Backup files that correspond to a specific backup, This could be handled through the powershell api. This may also require re-cataloging on or side.

[0595] Customer-side Implications Customer must have sufficient bandwidth to upload ~200 GB/wk/server (assuming each server is approximately 120 GB).

[0596] Datacenter Implications Doyenz must provide sufficient bandwidth to upload all customer data on a regular basis.

[0597] Data Encryption Data can be stored encrypted

[0598] Restore Implications Does not provide instant restores. Requires 3rd Party in the Doyenz datacenter to perform restores

[0599] Development cost* *Small in comparison to others

[0600] Supportability* *Uncertain. Biggest support risk involves the restore using 3rd Party.

[0601] Storage implications Similar to our current storage for shadow protect—without the snapshots per backup.

[0602] Storage management Requires rebasing and deleting a prior series of backup sets.

[0603] Machine management Machines would have to be co-managed by Doyenz and by 3rd Party. Doyenz would need to keep track of each one for backup purposes, and 3rd party would need to track them for restore purposes.

[0604] Pros simplest solution, should be easy to create agent plugins to handle this.

[0605] Cons Large amount of data upload, requires a lot of bandwidth in order to meet our SLAs. Slow restores that have lots of moving parts

[0606] 3rd Party to 3RD PARTY STORAGE APPLIANCEs

[0607] Approach outline: Customer does not have 3RD PARTY STORAGE SOLUTION on site. Customer either schedules backups to go directly to a 3RD PARTY STORAGE APPLIANCE running in Doyenz's cloud, or schedules a set-copy following standard backups to transfer them to a 3RD PARTY STORAGE APPLIANCE running in Doyenz's cloud. The Doyenz side 3RD PARTY STORAGE APPLIANCE is started at the beginning of the backup or set-copy job, and closes down on the completion of the job. This requires re-cataloging on our side.

[0608] Customer-side Implications Customer must either give up local copies, or must add a set copy to their existing schedule.

[0609] Datacenter Implications Doyenz must provide a VM running 3RD PARTY STORAGE APPLIANCE, with ~4 G of memory for each customer for the duration of upload. SSH tunneling will be required or a dedicated public IP per customer will be required

[0610] Data Encryption Setcopy will store unencrypted data locally.

[0611] Restore Implications Requires 3rd Party in the Doyenz datacetenter to perform restores

[0612] Storage implications Servers backed up by a single instance of 3rd Party are stored together in the VMDK corresponding to their instance of 3RD PARTY STORAGE APPLIANCE.

[0613] Storage Management Each 3RD PARTY STORAGE APPLIANCE instance is stored in ZFS in a similar fashion to our current machine storage. A snapshot is taking following each 3rd Party dedup solution, and snapshots are backed up via zfs sends to an archive.

[0614] Machine Management Machines are stored together for a customer, and are not separable without a 3RD PARTY STORAGE APPLIANCE instance.

[0615] Supportability and Operations cost Unknown. It may require 3rd Party help to sort out corrupted repositories.

So far, there are lots of ways setting up a 3RD PARTY STORAGE APPLIANCE and getting 3rd Party Dedup Solution to work to it can fail.

[0616] Pro Uses a "proven" deduplication solution

[0617] Cons

[0618] ppp. Requires 3RD PARTY STORAGE APPLIANCE to backup and recover data

[0619] qqq. 3RD PARTY STORAGE APPLIANCE3RD PARTY STORAGE APPLIANCELots of moving parts, fragility

[0620] rrr. 3RD PARTY STORAGE APPLIANCE does not communicate internal problems

[0621] Risks

[0622] sss. 3RD PARTY STORAGE APPLIANCE's are touchy about configuration, and when misconfigured, they don't give clear indications about what needs to change.

[0623] ttt. We don't have a robust, mechanical, way of spinning up 3RD PARTY STORAGE APPLIANCEs that leads to simple instructions for automation

[0624] uuu. Lots of moving parts that are out of our hands

[0625] vvv. We don't know the traffic compression rate of this approach.

[0626] www. We don't know how robust the storage on 3RD PARTY STORAGE APPLIANCEs will be at this point

[0627] Solution Cost Development, operations and support costs are high

[0628] 3Rd Party Storage Solution 3rd Party Dedup Solution

[0629] Approach outline: Customer installs 3RD PARTY STORAGE SOLUTION on their site, schedules an 3rd Party Dedup Solution job with each that synchronizes their repository with a 3RD PARTY STORAGE APPLIANCE running in Doyenz's cloud. The Doyenz side 3RD PARTY STORAGE APPLIANCE is started at the beginning of the 3rd Party Dedup Solution job, and closes down on the completion of the job. This requires re-cataloging on our side.

[0630] Customer-side Implications Customer must have 3RD PARTY STORAGE SOLUTION installed.

[0631] Datacenter Implications Doyenz must provide a VM running 3RD PARTY STORAGE APPLIANCE, with 2 to 4 G of memory for each customer for the duration of upload. 3rd Party storage solution to 3RD PARTY STORAGE APPLIANCE communication will require a VPN connection

[0632] Data Encryption Data is store and transmitted encrypted

[0633] Restore Implications Requires 3rd Party in the Doyenz datacetenter to perform restores

[0634] Supportability *and Operations *Unknown. It may require 3rd Party help to sort out corrupted repositories. So far, there are lots of ways setting up a 3RD PARTY STORAGE APPLIANCE and getting 3rd Party Dedup Solution to work to it can fail.

[0635] Storage implications Servers backed up by a single instance of 3rd Party are stored together in the VMDK corresponding to their instance of 3RD PARTY STORAGE APPLIANCE.

[0636] Storage Management Each 3RD PARTY STORAGE APPLIANCE instance is stored in ZFS in a similar fashion to our current machine storage. A snapshot is taking following each 3rd Party dedup solution, and snapshots are backed up via zfs sends to an archive.

[0637] Machine Management Machines are stored together for a customer, and are not separable without a 3RD PARTY STORAGE APPLIANCE instance.

[0638] Pros Uses a "proven" deduplication solution

[0639] Cons

[0640] xxx. Requires 3RD PARTY STORAGE APPLIANCE to backup and recover data

[0641] yyy. 3RD PARTY STORAGE APPLIANCE3RD PARTY STORAGE APPLIANCELots of moving parts, fragility

[0642] zzz. 3RD PARTY STORAGE APPLIANCE does not communicate internal problems

[0643] Risks

[0644] aaaa. 3RD PARTY STORAGE APPLIANCE's are touchy about configuration, and when misconfigured, they don't give clear indications about what needs to change.

[0645] bbbb. We don't have a robust, mechanical, way of spinning up 3RD PARTY STORAGE APPLIANCEs that leads to simple instructions for automation

[0646] cccc. Lots of moving parts that are out of our hands

[0647] dddd. We don't know the traffic compression rate of this approach.

[0648] eeee. We don't know how robust the storage on 3RD PARTY STORAGE APPLIANCEs will be at this point

[0649] Solution Cost Development Cost Development, operations and support costs are high

[0650] VSS Snapshots of Local 3RD PARTY STORAGE SOLUTION

[0651] Approach outline: Customer installs 3RD PARTY STORAGE SOLUTION and a Doyenz agent on their site. Customer schedules backups to run against the 3RD PARTY STORAGE SOLUTION, with a post command to notify the agent of completion. Following each backup, the Doyenz agent performs a VSS snapshot, and sends the file changes since the last backup to Doyenz. This requires re-cataloging on our side.

[0652] Customer side implications May require a custom VSS provider to capture changes in data.

[0653] OpenDedup Synchronization

[0654] Approach outline: Customer installs a Doyenz agent and sets 3rd Party up to do incremental VM generation (either to ESX or Hyper-V). The Doyenz agent sets up a file system on top of OpenDedup to receive the generated VMs, and uploads the deduped VM via OpenDedup's synchronization mechanism.

[0655] Storage implications Storage can becompletely managed by OpenDedup

[0656] Storage Management Storage management is mostly out of our hands.

[0657] Machine Management Potentially, manage machines as a root directory with each backup being a sub directory.

[0658] Customer-side Implications Customer should preferablyt be running a hypervisor that mounts an OpenDedup volume.

[0659] Datacenter Implications Doyenz should preferably establish and maintain one or more OpenDedup services.

[0660] Restore Implications If we are backing up vmdks, we get instant restore. OpenDedup provides an NFS service, which we just mount from the ESX host.

[0661] Supportability*. Although OpenDedup can beopen source

[0662] Pros

[0663] ffff. Gives us control of the dedup solution

[0664] gggg. Provides for the potential of instant restores.

[0665] Cons

[0666] hhhh. Immature and somewhat complex dedup platform.

[0667] Lightweight Dedup Transmission (Much Like Rsync with Block Motion)

[0668] Approach outline: Customer installs a Doyenz agent. The Doyenz datacenter and the customer agent share a dedup fingerprint for some number of previous uploads. Agent uses this to map blocks of next upload, uploads a new fingerprint and any require changes. Doyenz writes new blocks and rearranges existing blocks in storage to match the dedup fingerprint. The effect is that this dedups transmission, but not necessarily storage.

[0669] Use for VMDKs

[0670] The previous VMDK should be adequate for providing the fingerprint for the next upload.

[0671] Experimental results show that this works fairly well with 4 k blocks.

[0672] Better results may be obtained by utilizing VMDK structures for exact block alignment.

[0673] Use for 3RD PARTY BACKUPs

[0674] This approach requires a number of prior 3RD PARTY BACKUPs for fingerprint matching, and somewhat more complex data structures for keeping track of which file contains which block.

[0675] It also requires parsing of the 3RD PARTY BACKUPs to achieve any reasonable block alignment.

[0676] Need the 3RD PARTY BACKUPs to be stored unencrypted.

[0677] Need to go back to every and look at every incremental until a rebase.

[0678] Need a file system equivalent to track the authoritive source of specific blocks

[0679] Backup Capture Alternatives

[0680] Capture 3RD PARTY BACKUPs

[0681] Approach outline

[0682] iiii. Customer points a Doyenz agent at a storage facility for 3RD PARTY BACKUPs

[0683] jjjj. Agent performs some sort of chain analysis and uploads 3RD PARTY BACKUPs as necessesary.

[0684] Transmission implications Not really feasible without some sort of dedup.

[0685] ESX Host

[0686] Approach outline:

[0687] kkkk. Customer has an ESX host.

[0688] llll. 3rd Party is configured to perform incremental P2V restores to this host at each backup

[0689] mmmm. Doyenz captures the changed blocks and either uploads them as they are, or does a transmission level dedup/redup

[0690] Transmission Implications Not particularly feasible without block level dedup

[0691] Customer Implications Requires an ESX host

[0692] Restore Implications HIR is already completed. Can be handled in a similar fashion to ESX backups.

[0693] Hyper-V Host

[0694] Approach outline:

[0695] nnnn. Customer has an Hyper-V host.

[0696] oooo. 3rd Party is configured to perform incremental P2V restores to this host at each backup

[0697] pppp. Doyenz captures the changed blocks and either uploads them as they are, or does a transmission level dedup/redup

[0698] Transmission Implications Not particularly feasible without block level dedup

[0699] Customer Implications Requires Hyper-V (comes with SBS 2008 R2)

[0700] Restore implications Can be handled in a similar fashion to ESX backups. Requires HIR at restore time

[0701] ESX Stub

[0702] Approach outline:

[0703] qqqq. Doyenz agent will run a local web server which mocks vSphere API calls.

[0704] rrrr. Customer starts 3rd Party incremental convert to ESX VM which the ESX stub intercepts and return proper responses to 3rd Party.

[0705] i. Intercepting vSphere API calls can be done using a web server

[0706] ii. Intercepting vStorage API calls can be done by hooking VDDK library or implementing a TCP based mock server.

[0707] ssss. Write requests to the vmdk will be de-dupped and written locally.

[0708] tttt. Doyenz agent will upload the de-dupped VM and apply to a VM stored in the cloud.

[0709] Customer-side Implications: has to allow the local web server to run and bind to ESX ports and have enough memory and storage for efficient dedup.

[0710] Storage implications: re-dupped VM will take significant storage unless dedupped again in a dedup enabled file system.

[0711] Restore implications: restore is immediate and similar to current ESX/vSphere restore

[0712] Pros:

[0713] uuuu. Low customer requirements

[0714] vvvv. Instant restore

[0715] Cons:

[0716] wwww. Slightly invasive if we are replacing all ESX calls made to go through a stub

[0717] xxxx. Need to deal with fragility and complexity of the vSphere APIs

[0718] yyyy. Need to deal with cases where customer has web server which listens on the same port

[0719] zzzz. High cost in development and handling edge cases

[0720] Variation of this idea which could be used as a more expensive but incremental path towards this solution is to implement a reverse proxy to a real running ESX instance at Doyenz DC and de-dup only the writes transport calls.

[0721] Restore Alternatives

[0722] Run 3rd Party in the Doyenz Datacenter

[0723] Approach outline: 3rd Party starts, updates its catalog from the repository, and performs the following steps:

[0724] B2V restore of the system full, without applications

[0725] Simultaneous restore of applications, system incrementals, and application incrementals.

[0726] Customer Implications Restores might be very slow.

[0727] Datacenter Implications Either need to take a large additional hit for cataloging at restore time, or the data center needs to re-catalog frequently. If we re-catalog frequently, we need to manage a large number of 3rd Party instances (on the order of 1 for every 25 to 100 customers uploading VMs).

[0728] Receive VMs from Customer

[0729] Approach outline: Data uploaded corresponds to hard drive blocks and possibly VM meditate files. These are applied to a VMDK on the Doyenz side following receipt. Restore is a matter of starting up the given VM on an ESX host in the datacenter.

[0730] Customer Implications The customer may need to do some additional configuration to set up the VM generation on their side. Restores seem nearly instantaneous.

[0731] Datacenter Implications Depending on how they are generated, we may need to run HIR on VMs at restore time.

[0732] Storage alternatives

[0733] Storage inside of a dedup repository

[0734] Storage as VMs in ZFS snapshots

[0735] Storage as raw 3RD PARTY BACKUPs

[0736] Failback alternatives

[0737] Send VM back to customer

[0738] Update a dedup repository and synchronize this back to the customer

[0739] Perform a full 3RD PARTY BACKUP backup and send 3RD PARTY BACKUPs back to customer

[0740] Full Solution Proposals

[0741] 3RD PARTY STORAGE SOLUTION to 3RD PARTY STORAGE APPLIANCE

[0742] 3rd Party to 3RD PARTY STORAGE APPLIANCE Approach

[0743] Basic Approach

[0744] Customer does not have 3RD PARTY STORAGE SOLUTION on site. Customer either schedules backups to go directly to a 3RD PARTY STORAGE APPLIANCE running in Doyenz's cloud, or schedules a set-copy following standard backups to transfer them to a 3RD PARTY STORAGE APPLIANCE running in Doyenz's cloud. The Doyenz side 3RD PARTY STORAGE APPLIANCE is started at the beginning of the backup or set-copy job, and closes down on the completion of the job.

[0745] Backup Path

[0746] From the customer perspective:

[0747] Customer installs the Doyenz agent.

[0748] Customer adds a Doyenz based 3RD PARTY STORAGE APPLIANCE as an OST target. This is done through either a customer specific public IP, or through tunneling from a local interface to Doyenz.

[0749] Customer either makes this the target of the backup for a Doyenz managed machine, or, if the customer wants a local copy of the backup data, the customer makes this the target of a set-copy following the backup.

[0750] If the backup to Doyenz, or set-copy to Doyenz fails, 3rd Party will try again on the next scheduled backup.

[0751] The customer will have a web interface, provided by Doyenz, to which he or she can connect, and view backups that have been stored. The customer can use this interface to perform test restores and fail-overs.

[0752] Technical implications:

[0753] Doyenz will need to set up a 3RD PARTY STORAGE APPLIANCE for each customer

[0754] aaaaa. Need to determine how many of these can run simultaneously on an ESX host

[0755] bbbbb. Stored as VMs on a store service

[0756] Customer will need to install Doyenz agent, which may configure tunneling in order to connect to a cloud based 3RD PARTY STORAGE APPLIANCE

[0757] Doyenz will need to make 3RD PARTY STORAGE APPLIANCE available for initial connection.

[0758] Restore Path

[0759] From the customer perspective:

[0760] Customer connects to Doyenz application website

[0761] Customer selects machine to restore

[0762] Customer clicks restore and after some amount of time, machine is restored.

[0763] Customer has VNC connection with restored machine.

[0764] Technical implications:

[0765] Doyenz will need to spin up the appropriate 3RD PARTY STORAGE APPLIANCE and a 3rd Party instance to perform the restore.

[0766] Doyenz will have to make the restore in several steps (in addition to the standard routing issues, etc.)

[0767] ccccc. B2V of the most recent full backup, system only

[0768] ddddd. Log into restored VM

[0769] eeeee. Perform an application, system incremental, and application incremental backup all at once.

[0770] ESX Stub Approach

[0771] Basic Approach

[0772] Customer server will reside a doyenz agent which will handle ESX VMDK generation, detect the change blocks, dedup to reduce size of transmission and upload the change blocks to the Doyenz data center. The change blocks will be applied to a VMDK which then gets stored for instant restore

[0773] Backup Path

[0774] From the Customer Perspective:

[0775] Customer installs the Doyenz agent.

[0776] Customers sets up the backup schedule—full and incremental backups

[0777] Customer enables simultaneous convert o esx vm on that schedule

[0778] Customer sets pre and post command to trigger our agent Customer needs to change malware detection policies to exclude Doyenz agent and/or 3RD PARTY—Needs investigation if this is needed

[0779] Customer may need doyenz agent with every beremote.exe which is likely to mean that it will need to reside on every machine. Pending investigation

[0780] The customer can use Doyenz web user interface to acces the cloud backups and/or perform test restores and fail-overs.

[0781] Technical Implications:

[0782] Doyenz agent will run a local web server which mocks vSphere API calls.

[0783] Customer starts 3rd Party incremental convert to ESX VM which the ESX stub intercepts and return proper responses to 3rd Party.

[0784] fffff. Intercepting vSphere API calls can be done using a web server

[0785] ggggg. Intercepting vStorage API calls can be done by hooking VDDK library.

[0786] Write requests to the vmdk will be de-dupped and written locally.

[0787] We will require buffer the writes to make sure we are only writing the final changes. Will require extra disk space on the client proportional to the change data size

[0788] May need extra memory requirements—need to investigate

[0789] Doyenz agent will upload the de-dupped

[0790] VM and apply to a VM stored in the cloud.

[0791] Restore Path

[0792] From the Customer Perspective:

[0793] Customer connects to Doyenz application website

[0794] Customer selects machine, backup and a restore point to restore

[0795] Customer clicks restore and after some amount of time, machine is restored.

[0796] Customer has VNC connection with restored machine.

[0797] Technical Implications:

[0798] We need a redup service that runs writes reduped blocks to a mounted VMDK

[0799] Step to Conform the VMX

[0800] Higher storage requirements than our existing ESX implementation. Guess is 10%. The arises as the blocks might be in different places and ZFS does not deal with that

[0801] Archiving needs to be adapted to handle consolidation

[0802] Failback—Option 1—VMDK, Option 2—Run 3rd Party and send them a 3RD PARTY BACKUP backup

[0803] Issues Encountered/Concerns

[0804] May be perceived invasive if we are replacing all ESX calls in runtime to go through a stub

[0805] Need to deal with fragility and complexity of the vSphere APIs

[0806] Need to deal with cases where customer has web server which listens on the same port

[0807] High cost in development and handling edge cases

[0808] If a incremental backup fails, 3rd Party will require a rebase. We need to understand how likely we are to cause an incremental to fail. This is likely even in the 3rd Party to 3RD PARTY STORAGE APPLIANCE case.

[0809] Hyper-V Approach

[0810] Basic approach

[0811] Customer server will reside a doyen agent that will use a hyper-V VHD generation to detect the change blocks, dedup to reduce size of transmission and upload the change blocks to the Doyenz data center. The change blocks will be applied to a VMDK which then gets stored and restored as a HIR instant restore

[0812] Backup Path

[0813] From the Customer Perspective:

[0814] Customer installs the Doyenz agent.

[0815] Customers sets up the backup schedule—full and incremental backups

[0816] Customer enables simultaneous convert to hyper-v vm on that schedule

[0817] Customer sets pre and post command to trigger our agent

[0818] Customer needs to change malware detection policies to exclude Doyenz agent and/or 3RD PARTY—Needs investigation if this is needed

[0819] The customer can use Doyenz web user interface to access the cloud backups and/or perform test restores and fail-overs.

[0820] Technical Implications:

[0821] Customer starts 3rd Party incremental convert to Hyper-V VM which the doyenz agent will intercept writes to the VHD.

[0822] Write requests to the vmdk will be de-dupped and written locally.

[0823] We will require buffer the writes to make sure we are only writing the final changes. Will require extra disk space on the client proportional to the change data size

[0824] May need extra memory requirements—need to investigate

[0825] Doyenz agent will upload the de-dupped

[0826] Blocks will be applied to a VM stored in the cloud.

[0827] Restore Path

[0828] From the Customer Perspective:

[0829] Customer connects to Doyenz application website

[0830] Customer selects machine, backup and a restore point to restore

[0831] Customer clicks restore and after some amount of time, machine is restored.

[0832] Customer has VNC connection with restored machine.

[0833] Technical Implications:

[0834] We need a redup service that runs writes reduped blocks to a mounted VMDK

[0835] Step to perform HIR

[0836] Step to create and conform the VM configurations

[0837] Higher storage requirements than our existing ESX implementation. Guess is 10%. The arises as the blocks might be in different places and ZFS does not deal with that

[0838] Archiving needs to be adapted to handle consolidation

[0839] Failback—Option 1—VMDK, Option 2—Run 3rd Party and send them a 3RD PARTY BACKUP backup

[0840] Issues Encountered/Concerns

[0841] Need to get the feature to work

[0842] Potentially bottleneck in file system interception—need to do it efficiently

[0843] High cost in development and handling edge cases

[0844] If a incremental backup fails, 3rd Party will require a rebase. We need to understand how likely we are to cause an incremental to fail. This is likely even in the 3rd Party to 3RD PARTY STORAGE APPLIANCE case.

[0845] vSphere Spoofing (for Example Using Public APIs)

[0846] Preparation steps.

[0847] 1. It's require to hack Download Service to analyse http post/get command.

[0848] a. Download and apply the batch file under attachment

[0849] b. buildall.bat DownloadService

[0850] c. deployDownloadService.bat

[0851] The Download Service can act as a proxy to record all the traffic between 3rd Party and ESX.

[0852] 2. copy a VM with 3RD PARTY installed, move that vm to any esx host, power it up, run 3RD PARTY, change the ESX address to your DownloadService. eg"10.20.11.12:30111"

[0853] Founding so far.

[0854] doGet command is hacked. A standard response of doGet is in ESXResponseTemplate

[0855] doPost:

[0856] 2. RetrieveServiceContent is hacked, it returns the same response on every call.

[0857] 3. Logout is hacked.

[0858] 4. These command are preferably called in this order: CreateContainerView, CreateFilter, WaitForUpdateEx, DestroyPropertyFilter

[0859] 5. Above sequence is called multiple times on each backup.

[0860] 6. CreateContainerView is called slightly different. on (DataCenter, DataStore, VirtualMachine)

[0861] 7. CreateContainerView preferably returns a session id.

[0862] Advise on further research.

[0863] 8. Instead of analyzing the doGet/doPost alone, write a simple java class to call the vSphere api. Then compare the log file between 3RD PARTY and that temporary java class.

[0864] VSphere Agent

[0865] Goals

[0866] The goal is to integrate the VSphere agent with the ACU code base to leverage:

[0867] Server side configuration management

[0868] UploadService based uploads

[0869] DFT upload mechanic

[0870] Common code maintenance.

[0871] Components

[0872] The common backup worker currently used by the SP agent

[0873] A VSphere plugin, comprising:

[0874] hhhhh. A VSphere specific machine abstraction

[0875] iiiii. VSphere specific file transfer mechanic

[0876] jjjjj. Buffering to facilitate httpfiletransfer keepalive

[0877] A virtual machine to host the agent. Options under consideration:

[0878] kkkkk Windows—to leverage the existing C# updater

[0879] lllll. Linux—to leverage leverage free licensing and the lower disk space requirement

[0880] Configuration pages to handle the new VSphere configuration options

[0881] Design Considerations

[0882] The http file access is fragile, and the cost of losing it with ESX 4.1 and greater is high. We need to continuously explore other options, and design VSphere interaction with this fragility in mind.

[0883] VMDKs are usually very sparse, and we should consider this in the upload and in the LBS storage. This may involve detecting runs of zeros and marking them

[0884] Concurrence limitation can be important.

[0885] Example Solution Research

[0886] 3RD PARTY Backups ideas

[0887] Upload 3RD PARTY backup files similar to SP backup files

[0888] mmmmm. Restore backups on demand using 3RD PARTY convert to ESX vmdk

[0889] nnnnn (or) Restore backups on demand using 3RD PARTY WinPE restore disk

[0890] Client side changed block detection

[0891] ooooo. vmdk approach—Configure 3RD PARTY to perform "convert to open vmdk" at the end of daily backup

[0892] i. When backup convert completes—identify changed blocks from previous day vmdk

[0893] ii. Main problem with this approach is how to perform diff of changed blocks. Couple of options to address that:

[0894] 1. Option 1: Perform a binary diff on the 2 files—expensive from 10 bandwidth and storage (research)

[0895] 2. Option 2: Identify changed blocks using 3rd party tools which can open 3RD PARTY backup files

[0896] 3. Option 3: Mount backup files and identify changed blocks using VSS snapshots—initial investigation turned out that this may be non-trivial since they use custom device snapshots which are not easily accesible

[0897] 4. Option 4: Mount backup files and identify changed files (and then blocks) by comparing NTFS MFTs

[0898] 5. Option 5: Detect mapping between backup files to blocks by mounting backup chain and detecting system io calls from it while reading the disk

[0899] 6. Option 6: User 3RD PARTY APIs to determine changed blocks (not sure if it even supports this)

[0900] ppppp. Detect changed blocks directly from 3RD PARTY backup files

[0901] i. Using 3rd party APIs/documentation about backup file structure

[0902] ii. Trace reads from mounted backup files (research)

[0903] 1. Mount backup chain at last restore point

[0904] 2. Scan the mounted disk device block after block

[0905] 3. Intercepting block reads using a filesystem filter driver

[0906] 4. Map block read to chain files that were not uploaded

[0907] iii. Mount latest backup chain and previous backup chain file and run binary diff on block level—

[0908] 1. pro: very reliable

[0909] 2. con: may be expensive from JO bandwidth point of view.

[0910] iv. Mount only the latest backup chain and scan disk against previous md5 log of previous chain

[0911] qqqqq. Upload changed blocks only

[0912] rrrrr. Apply only changed blocks to zfs dataset mounted ESX vmdk on backend and take zfs snapshot—this takes care of consolidation (research)

[0913] sssss. When doing restore—perform necessary HIR operations

[0914] i. Using SP restore HIR

[0915] ii. By running HIR scripts on the mounted vmdk

[0916] ttttt. Boot vmdk in an Hypervisor:

[0917] i. ESX—will require conversion to ESK vmdk for openvmdk approach mentioned above which is an expensive operation in terms of JO bandwidth

[0918] ii. VirtualBox/VMWare Server/XEN—no current platform support for this—expensive from dev time

[0919] Concerns

[0920] Is a disk scanning on a customer like physical machine is fast enough?

[0921] Scanning mounted chain method may not be reliable. Is there a reliable way to detect the changed blocks consistently?

[0922] How many concurrent vddks mounts to vmdk can we maintain on a single box?

[0923] Thoughts on Block Hash Lookup Index

[0924] I'd first like to say that I don't think this is a must for 3RD PARTY since the signature file could be sufficient (although sub optimal) initial phase. The lookup which is used for the "d-sync" could be added later without changing the backend given the current design. It will certainly be a must for 3rd Party Windows agent.

[0925] So there are couple of approaches I was thinking about but probably none is simple in terms of development effort.

[0926] The requirement is to be able to do fast lookup of block offset given an md5 hash.

[0927] Data structures to support that:

[0928] 1. B+Tree or a just use a database which effectively creates a B/B+tree on a table index.

[0929] 2. Disk based hash table—flat file with hash collision buckets at constant offsets which should be resized when a bucket gets full. The file should be mmap-ed for better performance.

[0930] B-tree drawback is that is suffer from fragmentation for the type of data we intend to use. A mitigation strategy for this is creating pages with small fill factor which should reduce fragmentation till pages start to get full. The hash table suffers from the need for rehashing when buckets get full. So essentially both solutions suffer from similar problem and the choice should most likely be based on ease of implementation.

[0931] The idea is as follows (assuming index structure was selected):

[0932] 3. Create an empty index

[0933] 4. Insert/lookup index during backup

[0934] 5. If need rebuild parts of the index while waiting for chunk upload to complete or rebuild all if must.

[0935] 6. On the post backup signature processing—while rebuilding the new signature from repopulate the index with big fill factor so it would be ready for next backup.

[0936] If index get corrupted/missing—it can be rebuilt from the signature file like in step 4.

[0937] An optimization would be seed an index at the backend with known blocks for target OS/apps and send to client before backup start. This might have potential to reduce initial upload size by 10-20 GB per server.

[0938] We can consider thinking if there is a similar data structure or enhancement to the current 2 options which will allow partial rebuilding of the index instead of full rebuild every time it is needed.

[0939] While a preferred embodiment of the invention has been illustrated and described, as noted above, many changes can be made without departing from the spirit and scope of the invention. Instead, the invention should be determined entirely by reference to the claims that follow.

The embodiments of the invention in which an exclusive property or privilege is claimed are defined as follows:

1. A system comprising elements described above herein.

2. A method comprising steps described above herein.

* * * * *