

UNITED STATES PATENT AND TRADEMARK OFFICE

BEFORE THE PATENT TRIAL AND APPEAL BOARD

AMERICAN AIRLINES, INC. AND SOUTHWEST AIRLINES CO.,
Petitioners

v.

INTELLECTUAL VENTURES II LLC,
Patent Owner.

IPR2025-00931

U.S. Patent No. 8,332,844 B1
Filing Date: February 21, 2007
Issue Date: December 11, 2012

**DECLARATION OF DR. RICHARD NEWMAN IN SUPPORT OF
PATENT OWNER'S PRELIMINARY RESPONSE**

Table of Contents

I.	Introduction And Qualifications	1
II.	Summary Of Materials Reviewed And Considered	4
III.	Legal Standards	6
IV.	Level Of Skill in the Art and Perspective Applied in This Declaration	8
V.	Claim Construction	10
VI.	The '844 Patent	11
VII.	Background Art.....	17
VIII.	The Claims 1-27 Of The '844 Patent Would Not Have Been Obvious.	22
A.	Overview of Sapuntzakis	22
B.	Overview of Holzmann.....	27
C.	Overview of Federwisch.....	29
D.	Overview of Birse.....	34
E.	Ground 1: Claims 1-13 Would Not Have Been Obvious In View of Sapuntzakis in combination with Holzmann.....	39
1.	[1.2] a plurality of second storage units configured to store leaf images of respective compute nodes, said leaf images including only additional data blocks not previously contained in said root image and changes made by respective compute nodes to the blocks of said root image, wherein said leaf images of respective compute nodes do not include blocks of said root image that are unchanged by respective compute nodes	39
2.	[1.3] a cache configured to cache blocks of said root image previously accessed by at least one of said compute nodes	41
3.	Holzmann does not cure the shortcomings of Sapuntzakis.....	41
4.	There is a lack of motivation to combine Holzmann and Sapuntzakis.....	42

5.	A POSITA would be discouraged from forming the proposed combination	46
6.	Relocating caching to Holzmann’s central appliance changes Sapuntzakis’s principle of operation	48
7.	No Reasonable Expectation of Success in Combining the References	51
F.	Ground 2: Claims 14-27 Would Not Have Been Obvious In View of Sapuntzakis in combination with Federwisch	55
1.	[14.4] wherein further at least one of said compute nodes is configured to index said root image and provide the indexing results to another of said compute nodes.....	55
G.	Ground 3: Claims 1-13 Would Not Have Been Obvious In View of Birse in combination with Holzmann.....	56
1.	[1.2] a plurality of second storage units configured to store leaf images of respective compute nodes, said leaf images including only additional data blocks not previously contained in said root image and changes made by respective compute nodes to the blocks of said root image, wherein said leaf images of respective compute nodes do not include blocks of said root image that are unchanged by respective compute nodes	56
2.	[1.3] a cache configured to cache blocks of said root image previously accessed by at least one of said compute nodes.;.....	57
3.	Holzmann does not cure the shortcomings of Birse....	58
4.	A POSITA Would Not be Motivated to Combine the References	59
H.	Ground 4: Claims 14-27 Would Not Have Been Obvious In View of Birse in combination with Federwisch	60
1.	[14.4] wherein further at least one of said compute nodes is configured to index said root image and provide the indexing results to another of said compute nodes.....	60
IX.	Conclusion.....	61

I. Introduction And Qualifications

1. I, Richard Newman, submit this declaration on behalf of Intellectual Ventures II LLC, (“Patent Owner”) to provide my opinions concerning the validity of claims 1-27 (the “Challenged Claims”) of U.S. Patent No. 8,332,844 B1 (the “’844 patent,” EX1001), in support of Patent Owner’s Preliminary Response to the Petitioner for *Inter Partes* review of the ’844 patent, which I understand was filed by American Airlines and Southwest Airlines, (collectively the “Petitioner”) and accompanied by a Declaration of Dr. Erez Zadok (the “Zadok Declaration”), among other exhibits.

2. I am more than eighteen years of age, and I am a citizen of the United States, currently residing in the State of Florida.

3. In formulating my opinions, I have relied upon my knowledge, training, and experience in relevant art. My qualifications, along with a more detailed summary of my background and publications, are stated more fully in my *curriculum vitae*, which has been provided as Exhibit EX2015. Here, I provide a concise summary of my qualifications.

4. I am co-founder and Chief Systems Officer of Aerial Technology AG, a technology startup incorporated in Switzerland.

5. I am also a Senior Technical Fellow at IC2 Corporation, in Gainesville, Florida.

6. From 1986 until 2021, I was on the graduate faculty of the Department of Computer Information Science and Engineering at the University of Florida, chairing the Provost Office level Standing Committee on Networking and Telecommunications from 1994 until 1996, and was awarded the University-wide Superior Accomplishment Award for faculty for 1995-1996. From 1991 until 1997, I served as Associate Director of the Space Communications Technology Center.

7. Among other positions, I have been an instructor for IBM's continuing education program and for Raytheon, as well as Johns Hopkins CTY program, and was a technical consultant for Intellon Corporation of Ocala, Florida, which was acquired by Atheros and then Qualcomm, Inc. while I worked with them. I participated in the development of the HomePlug Powerline Alliance's HP1.1, HPAV, and GreenPHY standards, as well as IEEE P.1901 and P.1905 standards.

8. I have coauthored over 100 journal and refereed conference articles primarily in the areas of distributed systems, networking, and security.

9. I am a named inventor on seventeen patents and published patent applications corresponding to the areas of my professional work. Accordingly, I have a general understanding of the U.S. patent prosecution process and of the novelty and non-obviousness requirements for patentability.

10. I believe that my extensive industry experience and educational background qualify me as an expert in the relevant field of operating systems, networking, and distributed computing systems, including distributed file systems as described in the '844 patent and related art.

11. I taught Operating Systems at the graduate and undergraduate levels at the University of Florida and for IBM in Boca Raton, Florida, Research Triangle Park, North Carolina, and Toronto, Canada. I introduced the Distributed Systems course at the University of Florida in the late 1980's, which eventually became the graduate Operating Systems course, and taught it for many years as well.

12. I am knowledgeable of the relevant skill set that would have been possessed by a hypothetical person of ordinary skill in the art ("POSITA") at the time of the claimed invention of the '844 patent, which I have treated as December 30, 2004 for purposes of this proceeding. I have the necessary expertise to provide my informed opinions in this IPR, and I am an expert in the art of the '844 Patent, due to the following:

- I received my undergraduate degree in mathematics, then my master's and doctoral degrees in computer science;
- I have extensive experience in operating systems, networks, and distributed computing. I have been conducting state-of-the-art research in use of parallel, distributed and cloud computing for over four decades;

- I have extensive experience in software design and development in both academia and industry; and
- I have chaired over 20 doctoral student committees in computer science and served on the supervisory committees of scores more in computer science and related fields during the last 30+ years, in addition to supervising over 100 masters students. Many of these students are currently employed by companies such as Facebook, Amazon, Intel, and Microsoft.

II. Summary Of Materials Reviewed And Considered

13. The opinions contained in this Declaration are based on the documents I reviewed and my knowledge and professional judgment. In forming the opinions expressed in this Declaration, while drawing on my experience, I reviewed the following documents:

	Description
Paper 2	Petition For Inter Partes Review
EX1001	U.S. Patent No. 8,332,844 B1, entitled “Root Image Caching and Indexing for Block-Level Distributed Application Management,” issued December 11, 2012 to Kulkarni, et al. (“the ’844 patent”)
EX1003	Declaration of Dr. Erez Zadok (“Zadok Decl.”)

	Description
EX1005	Sapuntzakis, et al. “Optimizing the Migration of Virtual Computers,” Association for Computing Machinery, Special Interest Group on Operating Systems, Operating Systems Review, Proceedings of the 5th Symposium on Operating Systems Design and Implementation (OSDI), Volume 36, Issue S1, published December 31, 2002 (“Sapuntzakis”)
EX1006	U.S. Patent No. 7,089,300B1, entitled “Method and Apparatus for Administering the Operating System of a Net-Booted Environment,” filed on October 18, 1999, and issued August 8, 2006 to Birse, et al. (“Birse”)
EX1007	U.S. Patent Application Publication No. 2005/0283597 A1, entitled “System and Method for Booting Multiple Servers from a Single Operating System Image,” filed on June 22, 2004, and published on December 22, 2005 (matured into U.S. Patent No. 8,601,100, issued December 3, 2013 to Holzmann) (“Holzmann”)
EX1008	U.S. Patent Application Publication No. 2003/0182313 A1, entitled “System and Method for Determining Changes in Two Snapshots and for Transmitting Changes to Destination Snapshot,” filed on March 19, 2002, and published on September 25, 2003 (matured into U.S. Patent No. 6,993,539, issued January 31, 2006 to Federwisch, et al.) (“Federwisch”)

My opinions are additionally guided by my appreciation of how a person of ordinary skill in the art (POSITA) would have understood the claims of the '844 patent at the time of the invention, which I have been asked to assume is December 30, 2004. For purpose of this declaration, I will use the definition of POSITA as provided by Dr. Zadok (Ex1003) and described in Section IV.

III. Legal Standards

14. I am not an attorney. However, I have been informed by counsel of certain legal standards and have applied those standards, described below, in formulating my opinions.

15. I understand that the claims of a patent define the limits of a patentee's exclusive rights. I understand that to determine the scope of the claimed invention, courts may construe claim terms, the meaning of which the parties may dispute. I understand that claim terms should generally be given their ordinary and customary meaning as understood by a person of ordinary skill in the art at the time of filing of the patent application.

16. I understand that claims must be construed in light of and consistent with the intrinsic evidence. In this context, I understand that intrinsic evidence includes the claims themselves, the written disclosure in the specification, and the patent's prosecution history. I understand that the specification is always highly relevant to the claim construction analysis and often is the single best guide to the meaning of a disputed term. I understand that extrinsic evidence may also be considered when construing claims and may include, for example, technical dictionaries, technical publications and books, treatises, and expert testimony, though the intrinsic record will be more persuasive as to the meaning of a particular term.

17. I understand that one or more prior art references can render a patent claim obvious to a person of ordinary skill in the art if the differences between the subject matter set forth in the patent claim and the prior art are such that the subject matter of the claim would have been obvious at the time the claimed invention was made. In analyzing obviousness, I understand that it is important to consider the scope of the claims, the level of skill of a person of ordinary skill in the relevant art, the scope and content of the prior art, the differences between the prior art and the claims, and “secondary considerations” of non-obviousness, which I also understand can include, for example, evidence of commercial success of the invention, evidence of a long-felt need that was solved by an invention, evidence that others copied an invention, or evidence that an invention achieved an unexpected result.

18. I understand that the claimed subject matter can be obvious when a person of ordinary skill in the art would have been motivated to combine pre-existing elements, and that combination yields no more than what a person of ordinary skill in the art would have expected. I also understand that, in assessing whether a claim is obvious, one must consider whether the claimed improvement is more than the predictable use of prior art elements according to their established functions. I understand that there need not be a precise teaching in the prior art directed to the specific subject matter of a claim because one can take account of

the inferences and creative steps that a person of ordinary skill in the art would employ. However, I further understand that obviousness cannot be based on the hindsight combination of components selectively culled from the prior art.

19. I understand that for a reference to qualify as a relevant prior art reference in an obvious analysis, the reference must be “analogous”—meaning it is from the same field of endeavor as the patented technology, and if not, then the reference must be reasonable pertinent to the particular problem facing the inventors of the patent at issue. I also understand that a reference is reasonably pertinent if a person of ordinary skill in the art would have reasonably consulted those references and applied their teachings in seeking a solution to the problem facing the inventors.

IV. Level Of Skill in the Art and Perspective Applied in This Declaration

20. I understand that certain issues relating to validity must be judged from the perspective of a person of ordinary skill in the art (“POSITA”). I further understand that factors that may be considered in determining the level of skill of a person of ordinary skill in the art include, but are not limited to, (1) the type of problems encountered in the art; (2) prior art solutions to those problems; (3) rapidity with which innovations are made; (4) sophistication of the technology; and (5) the educational level of active workers in the field.

21. I have been informed and understand that a person of ordinary skill in the art is a hypothetical person who is presumed to be aware of all pertinent prior art, thinks along conventional wisdom in the art, and is a person of ordinary creativity. Importantly, this hypothetical person is of ordinary skill in the art to which the claimed subject matter pertains.

22. With respect to the '844 patent, Dr. Zadok's declaration states, "In my opinion, a POSITA would have a bachelor's degree in computer science, computer engineering, electrical engineering, or an equivalent degree. A POSITA would also have had approximately two years of experience working in the fields of networked data storage, file systems, and operating systems. Additional experience might substitute for less education, and *vice versa*." EX1003, ¶35

23. For purposes of this Declaration, I agree with Dr. Zadok, and have applied the above-defined perspective of a person of ordinary skill in the art, as of December 30, 2004, in forming and expressing my opinions. Unless otherwise stated, my testimony below refers to the knowledge of one of ordinary skill in the art at that relevant time.

24. At the time of the earliest priority date for the '844 Patent (December 30, 2004), I met the qualifications to be a POSITA, and I believe that I also have a sufficient level of knowledge, experience, and education to provide expert opinions in the field of the '844 Patent.

25. While my own level of skill exceeded that of the ordinary level of skill as of December 30, 2004, I am well-acquainted with the actual performance and capabilities of a person of ordinary skill in the art as defined above. For example, during the relevant timeframe, I was supervising and evaluating the performance of such individuals, such as my graduate students who were performing research in this area. I was also working alongside software professionals who were working for Intellon Corp. Furthermore, my participation in professional organizations and in conferences and other events informs me of the state of the industry and the level of experience of individuals who had an ordinary level of skill.

V. Claim Construction

26. In my analysis, I have applied the plain and ordinary meanings of the claim terms in the '844 Patent, as a person of ordinary skill in the art would have understood them in the context of that patent in December 30, 2004. And it is my opinion that the prior art set forth in the Petition fails to disclose or render obvious the subject matter of the challenged claims under any reasonable construction and interpretation of the challenged claims' terms, including their plain meanings.

27. I reserve the right to further explain how skilled artisans would understand terms in the '844 Patent's claims, including but not limited to the terms

discussed above, in the event the Board institutes an *inter partes* review of the '844 Patent.

VI. The '844 Patent

28. The '844 Patent introduces an innovative, “technology for among other things root image caching and indexing for block-level distributed application management.” EX1001, Abstract. This “involves storing blocks of a root image on a first storage unit and storing blocks of leaf images on respective second storage units [which] ... include additional data blocks not previously contained in the root image and changes made by respective compute nodes to the blocks of the root image.” *Id.* Additionally, “the technology includes caching blocks of the root image that have been accessed by at least one compute node.” *Id.* Finally, “the technology also includes receiving indexing results pertaining to the root image from one compute node and providing the results for other compute nodes.” *Id.*

29. Importantly, the technology is independent of the operating system. For example, it can operate at the level of the device driver, which provides a layer of abstraction that hides the location of the blocks (i.e., the drive on which they are located and whether they are original or modified) from the file system. In the disclosed system, the compute node may modify some parts of the operating system image without affecting the other compute nodes that access the same root

image, since the blocks it modifies are stored separately from the read-only root image. At the same time, compute nodes whose image derives from that of another compute node can share read access with the modified blocks in the parent's image, and even produce its own leaf image as it modifies blocks. This system allows for multiple versions of the same set of data to be stored efficiently, as only the portions of the data which have changed or been added take up additional space. However, this requires that accesses to a particular version be merged on a block-by-block basis, taking the most recently modified or added block up the path to the root, where blocks that have not been changed are to be found. Previous branching file systems operated at the file level, with new versions of a file replacing the previous version as they are created, so that small modifications to a large file still produced a large file. By storing changed blocks only, a time-space trade-off is made by which the only computation that is required is to determine which storage block holds the corresponding block of the desired version (essentially dereferencing an index, which is very fast) at the cost of storing the changed blocks (when only a small part of the block may be changed).

30. Accessing the branching file system by blocks rather than by files allows the system to be truly file system agnostic, that is, whereas part of the responsibility of the file system is to map the bytes of a file to its blocks and those blocks to their actual storage locations, the resolution performed by the Union

Block Device (UBD) is to map the storage location blocks (thought by the file system to be the actual storage locations of file blocks) to either a block of the root image stored on one device or a block of a leaf image stored on a second device. This mapping is completely transparent to the file system; it simply requests a block and receives the appropriate block from the UBS, wherever it may have been stored.

31. Caching of blocks as disclosed by the '844 patent occurs at the storage location, and may be performed for blocks of both the root image and for blocks in the leaf image. Moreover, only the parts of the operating system image that are needed by the compute node are transferred, so that the delay experienced in starting the compute node (booting it up) is minimized.

32. Time required to boot ("bring-up time") a compute node is significantly reduced by introducing the Union Block Device (UBD), which in the disclosed embodiment appears to the compute node as a device driver interfacing to a local disk. Previously, the entire boot image would be transferred to the compute node and then the compute node would access blocks of the boot image locally. Whether the file system is on the compute node or on the server, the compute node issues requests for the blocks corresponding to the pages it requires to operate. These blocks are stored in the compute node's memory (e.g., RAM) where their contents can be accessed quickly as needed by the instructions

executed by the process or operating system. In the disclosed system, the UBD performs transparent resolution for block access by resolving block access requests to the appropriate storage device with either the unchanged block of the root image or the new or modified block of the leaf image, without creating an entire separate image for each modified root image or application image, and without transferring the entire image to the compute node. Critically, in the disclosed embodiment, the compute node's file system treats the UBD as a device driver for an attached device, so that it considers any blocks accessible through the UBD to be equivalent to blocks on a local block storage device, hence they can be used by the paging system. This in turn means that only the pages that are needed for the working set during the "bring-up" stage need to be transferred through the UBD to the compute node's memory.

33. Caching on the storage devices allows rapid access to blocks that are in the block cache without having to wait for a transfer from the physical storage device, improving access time by an order of magnitude. Note that this is distinct from the block cache that is typically present on each compute node, where the blocks that have been recently accessed by that node (only) are stored for rapid access without making a call to retrieve the block through the driver (or UBD in this case).

34. In the disclosed embodiment, the UBD acts like a device driver in that it takes requests for storage device blocks and accesses the blocks as stored in the “device,” except now that device is actually two or more storage devices, and the storage device block number referenced by one compute node in its request to the UBD may translate to a different storage block or even different actual device than the one to which the same storage device block number translates for a different compute node.

35. In summary, the ‘844 Patent discloses an operating system-independent and efficient method and system for managing clusters of compute nodes that minimizes the amount of time required to bring a system up on a compute node and minimizes the storage space required to keep multiple versions of the operating system as modified by the compute nodes. It also reduces the amount of redundant work done by sharing the result of indexing the read-only root image once by one compute node and sharing the result with other compute nodes.

36. As a result, these capabilities represent significant improvements over prior systems for operating system and application management in distributed systems.

37. The independent claims are recited below for reference purposes.

Only claims 1 and 14 are given as claims 7 and 19 are the method versions of claims 1 and 14, respectively.

Reference	Claim Limitation
Claim 1	
[1.0]	1. A system for providing data to a plurality of compute nodes, the system comprising:
[1.1]	a first storage unit configured to store blocks of a root image of said compute nodes;
[1.2]	a plurality of second storage units configured to store leaf images of respective compute nodes, said leaf images including only additional data blocks not previously contained in said root image and changes made by respective compute nodes to the blocks of said root image, wherein said leaf images of respective compute nodes do not include blocks of said root image that are unchanged by respective compute nodes; and
[1.3]	a cache configured to cache blocks of said root image previously accessed by at least one of said compute nodes.
Claim 14	
[14.0]	14. A system for indexing file systems for a plurality of compute nodes, the system comprising:
[14.1]	a first storage unit configured to store blocks of a root image of said compute nodes;
[14.2]	a plurality of second storage units configured to store leaf images of respective compute nodes, said leaf images comprising only additional data blocks not previously contained in said root image and changes made by respective compute nodes to the blocks of said root image, wherein said leaf images of respective compute

	nodes do not include blocks of said root image that are unchanged by respective compute nodes; and
[14.3]	a plurality of union block devices corresponding to said compute nodes, said union block devices configured to interface between said compute nodes and said first and second storage units to distribute said file systems to said compute nodes, wherein said union block devices are configured to create said file systems by merging the blocks of said root image stored on the first storage unit with the blocks of respective leaf images stored on respective second storage units, and
[14.4]	wherein further at least one of said compute nodes is configured to index said root image and provide the indexing results to another of said compute nodes.

VII. Background Art

38. **Paging systems:** Most modern operating systems are configured with a paging system that supports virtual memory, allowing a process running on a computer to behave as though it has a much larger random access memory space than the computer actually has. In order to simplify management of virtual memory space, the process memory is divided into fixed size pages, with a byte in memory accessed according to page number and offset within that page. Processes generally only access a relatively small part of memory space at a time. Since processes generally execute a limited set of instructions over a limited amount of data compared to the total range of instructions and data, processes will have a “working set” of pages that contain these instructions and data. As long as that pages of the working set are in actual memory, access is fast and the process runs efficiently.

When a page is needed that is not in memory, then the paging mechanism must detect this and schedule that page to be read in from storage (e.g., disk). Demand paged virtual memory is supported by the page table and a dereferencing operation that examines the entry in the page table corresponding to the desired page of virtual memory and determines whether it is in actual memory or not. If it is, then the entry holds the base address of where the page is stored in actual memory, and to this base address the offset is added (or appended) to form the physical address of the desired byte. If the page is not in physical memory, then the process is suspended until the page can be brought into memory, which requires the paging system to determine the storage device and the location of the page on that storage device, then schedule the block or blocks corresponding to that page to be read from the storage device into a free location in physical memory. This may entail evicting a page from physical memory, writing it back out to its location on a storage device if it has been modified, and updating the page table accordingly. When the page has been read in from the storage device, then the paging system updates the page table to reflect the physical location and that the page is in memory, then resumes the process. The paging system knows the files in storage that are associated with the process image, and uses the file system's mechanisms to determine the disk block(s) that must be accessed.

39. **Storage Device Drivers:** Actual storage devices are accessed using drivers, which are low-level programs that translate requests from programs via the operating system into reads and writes of the registers that the device uses as its hardware interface. Drivers are specific to the characteristics of the device and the exact ways in which the device operates and communicates with the rest of the system, for example, how many platters, tracks, and sectors a disk has, where registers are located to write the desired location, where registers are located to write commands, what the format and semantics of those commands may be, where status registers are to read and what the format and semantics of those registers may be, and so on. These minute details are hidden from the file system and other parts of the operating system that access the device so that those parts of the operating system do not have to be changed when a device is changed, but rather only the device driver associated with the new device. As the device driver provides an abstraction to the operating system, it is also able to mask whether the device is a real piece of hardware attached to the system or something else, such as a remote server accessed by communicating over a network connection.

40. Device drivers operate at a low level of abstraction, leaving the more complex abstractions to higher layers. In the storage stack, device drivers sit between the actual hardware and the higher layers, such as the file system, and as such only handle reading and writing of the units of data known to the device (on

the lower side) and the units presented to the rest of the system (on the higher side). The physical device has its own units of data corresponding to its electrical, mechanical, and programming properties. The driver may combine two or more of these low-level data units into a single unit of a size desired by higher layers so that all devices, regardless of their individual quirks, present a uniform access size to the layers above.

41. **Disk Management:** The part of the operating system that manages disk storage and access may itself group multiple of these units into larger units for efficient access or management. For example, the blocks on the storage unit that hold data for later access and those that are available to be allocated and overwritten must be known to the system so that it does not overwrite files that have not been deleted and so that it can find locations to store new files or modifications to existing files efficiently. For a fixed size storage device, the larger these allocation units are, the smaller the data structures that are used to keep track of the used and free storage units. Accordingly, there is a risk of potentially at the allocating more space than is actually needed to store some data.

42. **File Systems:** File systems present the user and processes with named, persistent storage, with attributes such as access permissions, ownership, creation and modification times, etc. Most files systems present a hierarchical organization of directories/folders. A file or directory may be referenced by its

path, that is, the sequence of directory names from the root of the file system down to the file or directory itself, and the file system provides a way to resolve that path name by recursively looking up each directory in turn from the root on down until the final path element is encountered. It does this according to the particular way in which the file system is implemented, but generally, at each stage there is a current directory to search for the next path element, and then if it is found, the location where it is stored is determined. If it is another directory, then it is loaded into memory and searched; if it is the desired file, then its storage location can then be used to access data stored in it. In order to simplify disk space management, file storage is generally indexed so that adjacent blocks of a file are not necessarily stored in adjacent disk blocks. In this way, any free block on a disk can be used in principle to store any part of a file, although in practice, constraints and preferences are used to make disk access more efficient by storing related blocks “near” each other if possible. When a byte in a file is to be accessed, the file block in which that byte is located is first found (usually by truncating some of the bits off of its byte address within the file), and then the corresponding file block is accessed. If it is already in memory (in the system’s block cache) then the byte is accessed from RAM, otherwise the block (and perhaps some other blocks near it) are scheduled to be read from the storage device using the device driver interface. The file system translates path and file names to files, and byte offsets within a file into a block

and block offset within the file, and file blocks to storage device and block. The device driver then translates these blocks into the specific addressing needed by the hardware and arranges for transfer of one or more device level storage units corresponding to that block to or from main memory.

VIII. The Claims 1-27 Of The '844 Patent Would Not Have Been Obvious.

43. Petitioner challenges claims 1-27 (the “Challenged Claims”) of the '844 Patent based on several grounds.

44. In my opinion, none of the claims of the '844 Patent would have been obvious over Petitioner’s prior art, for reasons I explain below. I will first review the Petitioner’s prior art. Then I will address each of the grounds.

A. Overview of Sapuntzakis

45. Sapuntzakis is titled “Optimizing the Migration of Virtual Computers,” ACM, Special Interest Group on Operating Systems, Operating Systems Review, Proceedings of the 5th Symposium on Operating Systems Design and Implementation (OSDI), Volume 36, Issue S1, published December 31, 2002 (“Sapuntzakis”). It focuses on optimizing the efficiency of moving “the state of a running computer across a network, including the state in its disks, memory, CPU registers, and I/O devices.” This comprehensive state is called a “capsule” and “includes the entire operating system as well as applications and running processes.” EX1005, 2. The goal is to move the current state of the system being

used by a user to a new machine where the user is or will soon be, so that they can continue as though they brought the same physical machine with them, to the extent that this is possible, i.e., giving them a consistent environment. EX1005, 3.

46. Aside from some question as to whether Sapuntzakis references Type I (i.e., running on raw hardware) or Type II (i.e., running on a host operating system) virtual machine monitors EX1005 2, Sapuntzakis suspends the software and virtual machine state, captures the state, and serializes it for transmission over the networks of the time (i.e., slow by today's standards). It is telling that the disclosed system uses virtual hard disks that are backed by files in the host OS's file system. EX1005 2.

47. Capsules may derive from previous capsules, and so only changes may be transmitted. EX1005 3.1. The incremental cost of saving a capsule disk is reduced by capturing only the differences using copy-on-write virtual disks. EX1005, 6.

48. Sapuntzakis reduces the start-up time by paging disk data on demand. However, "moving the first capsule to a machine over the network can be costly, but may still be faster and less laborious than downloading and installing software from scratch." This suggests that the first instance of a capsule is copied in its entirety to the target machine. "Disks in capsules form a hierarchy starting from a *"root disk*, which is a complete capsule disk." EX1005 4.2. The disk in a capsule

and its children in the tree represent complete disks, and not just the portions that hold an operating system image. Once a capsule disk has children, it becomes immutable. EX1005 4.2.

49. Computers and storage work as caches of capsules 1. Here, the term cache is used at the file level with gaps in the cached files filled in through transfer of the missing or changed blocks.

50. Active capsules, however, unlike the system branching store file system disclosed by '844 are transitory. "The capsule ... is destroyed when [the student] logs out." EX1005 4.2. Only leaf capsules can be active, and these correspond to a user, not to a compute node. Once the user logs out, the capsule is destroyed, whereas the leaf nodes in the '844 file system are associated with the compute nodes and persist.

51. Disks are approached at the file system level. Capsule hierarchies allow efficient usage of disk space by sharing common data among different capsule disks. This also translates to efficient usage of the buffer cache of the host OS, when multiple capsules sharing COW disks simultaneously operate on the same host." EX1005 4.2. This allows for sharing of cached disk blocks only between processes on the same host, the same as a block cache would in any OS that has one. It does not disclose the block cache claimed in the '844 Patent that supports improved efficiency of access across multiple compute nodes.

52. Note that “COW disks for a capsule disk could be distributed across many hosts since they were created on different hosts.” *Id.* Contrast this with the system disclosed in the ’844 Patent in which the leaf disk for a compute node is not local but held at the file server.

53. “Each suspend and resume of an active capsule creates a new active capsule, and adds another COW layer to its disks.” *Id.*, EX1005 4.2. This contrasts with the system claim in the ’844 patent, in which the compute node has its own leaf disk on a server. In Sapuntzakis, the latest COW disk is always local. *Id.*, EX1005 4.3. This is necessary since the latest COW disk holds at least the parts of the system that have been written by the processes of the running capsule, which would otherwise incur an intolerable delay for access across a slow, high-delay communication line. The compute nodes in ’844 Patent may even be diskless thin clients, and whether thick or thin, the leaf image is stored at a storage server.

54. “First, the memory in most [operating] systems caches disk blocks.” *Id.*, EX1005 4.4. Cache for HCP is akin to the OS level block cache, not a cache on the disk itself. In the system disclosed in the ’844 Patent, the cache is at the storage device server.

55. “Most users run code that is distributed in binary form, with most of this binary code copied unmodified into memory when the application runs, and the same binary code (e.g., Microsoft Office or the Netscape web browser) is

distributed to millions of people. As a result, we expect to find common blocks, even between different users' capsules." *Id.*, EX1005 4.4.

56. Hash cache is persistent and does pre-fetch. "Each entry has a pointer to a file and a file offset." *Id.*, EX1005 4.4.1 "Also the cache can read ahead in the file, priming an in-memory cache with data blocks." *Id.* This shows that the cache has knowledge of files and their structures (i.e., can reference subsequent blocks in a file), unlike a block device driver.

57. Alignment. "When copying a memory image, we divide the file into page-sized blocks from the beginning of the image file." "For disk images, it is not effective to naively chop up the disk into page-sized chunks from the start of the disk; file data on the disk is not consistently page aligned." *Id.*, 4.4.2. "On a related note, ext2, FAT, and NT file systems all default to block sizes less than 4 KB when creating smaller partitions; as a result, files may not start on page boundaries. Luckily, the operator can specify a 4 KB or larger block size when creating the file system." The size of the blocks and their alignment depend on interaction with the file system and operating system.

58. FS interaction. Figure 4 shows a typical HCP session, demonstrating that the client calls the server first requesting a file handle (so that it can access the particular file desired), then it sends the hash of the (page-sized) block to the server, which causes the client to request the block if the hash does not match the hash of

the block the client has in its cache. *Id.*, 4.4.3. “Blocks found via the hash cache are copied into the output file...” *Id.* This indicates that the server is not simply serving blocks of a remote disk, but has knowledge of the existence of files to provide a file handle, and their structure, or at the very least their size, in order to provide the hashes of the page-sized blocks of the file.

B. Overview of Holzmann

59. Holzmann is titled “U.S. Patent Application Publication No. 2005/0283597 A1, entitled “System and Method for Booting Multiple Servers from a Single Operating System Image,” filed on June 22, 2004, and published on December 22, 2005 (“Holzmann).” Holzmann discloses a system and method for booting multiple servers or other network resources from a single operating system image. The operating system image is stored on a solid state disk. When a server is booted, cache space is allocated in the volatile memory portion of the solid state disk. This cache is used to store data necessary for booting and operation of the operating system.” EX1007, Abstract.

60. Holzmann describes prior art where “a separate copy of the operating system data 400 is loaded for each server...” *Id.*, [0031]. In Holzmann, “a single copy of the appropriate operating system data 400 is loaded into the memory portion 205.” *Id.*, [0032]. Separate cache space is allocated for each server: “By

allocating separate cache space for each server or other network resource, the inventive system prevents corruption of data....” *Id.*, [0033].

61. In an alternative configuration, “The memory in the non-associative cache 501 is divided into several lines 507. Each line 507 corresponds to a particular memory segment of the operating system data stored in memory portion 205. Each line 507 is further divided to store different kinds of information.” *Id.*, [0034]. Holzmann goes on to describe typical cache line metadata used to manage the cache. The key point is that these lines are not disk blocks, but typical memory cache lines, with a validity bit, lock bit, and tag to confirm the actual memory line to which the cache line corresponds. *Id.*, [0034], [0035]. The lock bit is used essentially as a “dirty bit” to indicate that the data in the line have been changed, and a server number data field is used to distinguish which server changed the data in the line, so that the same server may continue to make changes, while a different server is returned an error condition. *Id.*, [0036]. This does not support a copy-on-write branching file system as disclosed in the ’844 patent, and this cache is not the same as a block cache, and so does not correspond to the caching disclosed in the ’844 Patent.

62. Further, Holzmann’s shared cache essentially maintains one version (the latest) of the operating system. If one server writes a line, the locked bit for that line is set and the server number field is set to reflect the server that modified

the line. Any subsequent read of that line will obtain the modified data, which will not be specific to the server reading it, nor will it be the original system image data, so the cache is neither compute node specific nor does it preserve the original system data as read-only.

C. Overview of Federwisch

63. Federwisch refers to U.S. Patent Application Publication No. 2003/0182313 A1, entitled “System and Method for Determining Changes in Two Snapshots and for Transmitting Changes to Destination Snapshot,” filed on March 19, 2002, and published on September 25, 2003 (“Federwisch”).

64. Federwisch presents a “system and method for remote asynchronous replication or mirroring of changes in a source file system snapshot in a destination replica file system using a scan (via a scanner) of the blocks that makes up two versions of a snapshot of the source file system, which identifies changed blocks in the respective snapshot files based upon differences in volume block numbers” Federwisch, Abstract.

65. “Trees of blocks associated with the files are traversed, bypassing unchanged pointers between versions and walking down to identify changes in the hierarchy of the tree.” *Id.* “This technique allows regular files, directories, inodes and any other hierarchical structure to be efficiently scanned to determine differences thereof.” *Id.* “Any deleted or modified inodes already on the destination

are moved to a temporary or ‘purgatory’ directory and, if reused, are relinked to the rebuilt replicated snapshot directory.” *Id.*

66. The hierarchical structure described is that of the file system organization, with particular reference to the Unix fast file system, which uses inodes to store file meta-information and storage locations, with storage information managed in a hierarchical fashion. Here, the first several blocks of a file have their location stored in the inode itself, but for larger files, the inode contains pointers (block numbers on the storage device) for primary indirect blocks (which are much larger than the inodes and contain only pointers to many file blocks), secondary indirect blocks (which contain many pointers to primary indirect blocks), and even tertiary indirect blocks (which contain many pointers to secondary indirect blocks).

67. Federwisch discloses methods for mirroring to “improve reliability and facilitate disaster recovery....” *Id.*, [0007]. “One common form of update involves the use of a ‘snapshot’ process in which the active file system at the storage site, consisting of inodes and blocks, is captured and the ‘snapshot’ is transmitted as a whole, over a network ... to the storage site. Generally, a snapshot is an image (typically read-only) of a file system at a point in time, which is stored on the same primary storage device as the active file system and is accessible by users of the active file system.” *Id.*, [0008].

68. “According to an illustrative embodiment, the source scans, with the scanner, along the index of logical file blocks for each snapshot looking for changed volume block numbers between two source snapshots.” *Id.*, [0015]. “[A] difference indicates changes in the underlying inodes of the respective blocks.” *Id.*, [0015]. This again indicates that the scanner, and replication, is intimately enmeshed with the file system and its representation of file storage.

69. The “file system-independent” method taught by Federwisch refers only to the capability of storing the changed files on a destination file system that differs from the source file system. “This format consists of a set of standalone headers with unique identifiers. Some header[s] refer to follow-on data and other carry relevant data within their stream. For example, the information relating to any source snapshot deleted files are carried within “deleted files” headers. All directory information is transmitted first, followed by file data. ... At the destination, the format is unpacked and inodes contained therein are [sic] transmitted over the network are mapped to a new directory structure. Received data file blocks are written according to their offset in the corresponding destination file. An inode map stores entries which map the source inodes (files) to the destination’s inodes (files).” *Id.*, [0016]. Again, this clearly demonstrates that the system envisioned by Federwisch is intimately connected with file systems that use inode structures, and does not act as a block device driver. Moreover,

“After the directory tree is built, the transfer of file data begins. Changes to file data from the source are written to the corresponding replica files (as identified by the inode map). When the data stream transfer is complete, the purgatory directory is removed and any unlinked files (including various deleted files) are permanently deleted.” *Id.*, [0016]. This shows that Federwisch is not really file system agnostic at all, and data are removed at the level of files. In the system disclosed in the ’844 Patent, there are only disk block reads and writes, with no regard to details of file system structure or even files themselves.

70. “The clients transmit requests for file service to the source and destination filers 310, 312, respectively, ...” *Id.*, [0046]. The clients do not access the shared storage via block access requests, but by file access requests. This is further emphasized by “Data is transferred between client and the respective filer 310, 312 using data packets 374 defined as an encapsulation of the Common Internet File System (CIFS) protocol or another appropriate protocol such as NFS.” *Id.*, [0046]. “Notably, the application interacts with the file system layer to gain knowledge of files so that it is able to use a file-based data structure (inodes files, in particular) to replicate source snapshots at the destination.” *Id.*, [0055].

71. The client accesses the snapshot held locally, which receives updates from the source.

72. The snapshot is a “persistent consistency point image (PCPI) ... a point-in-time representation of the storage system, and more particularly, of the active file system, stored on a storage device ... and having a name or other unique identifiers that distinguishes it from other PCPIs taken at other points in time.” *Id.*, [0057]. “The terms ‘PCPI’ and ‘snapshot’ will be used interchangeably though out [sic] this patent” *Id.*, [0057]. “Snapshots are generally created on some regular schedule.... Under one storage scheme, a number of recent snapshots are stored in succession (for example, a few day’s worth of snapshots each taken at four-hour intervals), and a number of older snapshots are retained at increasing time spacings (for example, a number of daily snapshots for the previous week(s) and weekly snapshot for the previous few months).” *Id.*, [0058]. In other words, the snapshot and its management as disclosed by Federwisch is not an active, ongoing representation of the up-to-date state of a possibly changed file system, but is an archival capture of the file system’s state at some previous time. This does not match well with “on-the-fly” updates to the disk contents disclosed in the '844 Patent, rather this aspect of Federwisch is useful for increased reliability and recovery from disaster as disclosed in the abstract.

73. In fact, rather than the system building on the original, read-only root image as in the system disclosed in the '844 Patent, Federwisch modifies the active file system while the snapshot remains read-only. “Fig. 7 shows an exemplary

inode file system structure 700 after a file data block has been modified. In this illustrative example, file data which is stored at disk block 520C is modified. The exemplary WAFL file system writes the modified contents to disk block 520C', which is a new location on disk. Because of this new location, the inode file data which is stored at disk block (515) is rewritten so that it points to block 520C'. This modification causes WAFL to allocate a new disk block (715) for the updated version of the data at 515." *Id.*, [0061].

74. Lastly, all the nodes in Federwisch have the same view of the file system state – the snapshots are created to allow higher availability and for recovery in case of failure. In the system disclosed in the '844 Patent, each compute node has its own view of the system image, based on its own updates (and perhaps the updates of its ancestors in the branching file system). The system is not designed for recovery, but rather for quick distribution of the necessary parts of the operating system and for changes to be made by compute nodes to their own version of the modified root image.

D. Overview of Birse

75. Birse refers to U.S. Patent No. 7,089,300B1, entitled "Method and Apparatus for Administering the Operating System of a Net-Booted Environment," filed on October 18, 1999, and issued August 8, 2006 to Birse, et al. ("Birse"). In this patent a "method and apparatus are provided for supplying a reliable and

maintainable operating system in a net-booted environment.” The focus is on easy maintenance of the operating system for multiple network computers (NC) clients by one of the NC clients so that updates can be made easily.

76. Birse is concerned with user workstations in a networked environment. (see EX1006 4:13 – 35). There is a shadow file made for each user, which is transferred to the NC client when the user goes to log in, and the persistent parts are transferred back to persistent storage when the user logs out.

77. Birse divides the NC server (having persistent storage from which the NC clients are provided the operating system software) into “one or more system volumes 174, one or more application volumes 176, a user registry 178, non-persistent client-specific data 184, and persistent user data 186.” EX1006, 5:10–13.

78. “For example, at the conclusion of a user session on an NC client, the server software management process 180 may provide data that is to be preserved between user sessions to the user environment management process 182 and reinitialize the client-specific storage area associated with the NC client for use by the next user.” *Id.*, 6:17-22.

79. “[T]he NC server 170 may be configured to automatically replace the shadow volume at every boot of a NC client 150.” *Id.*, 6:26-29. That is, unless the user logs in as an administrator planning to change the OS, whatever changes are

made on an NC client are deleted once the user session is terminated. This is consistent with the management goals of Birse, to keep predictable and consistent system and application software available across all NC clients.

80. However, with an eye to the user experience, persistent user data is kept so that users can enjoy the environment they create as they work over multiple sessions. “[T]he user environment management process 182 tracks and maintains the persistent user data 186 to insure that changes the user has made during the current session will be persistent at the next [login].” *Id.*, 6:35-38. This shows that Birse focuses on users rather than client machines.

81. As an NC client may be a thin client (having no local disk), the NC client is configured to boot from the NC server to obtain the operating system image and application program images. “NC clients have no operating system internally. Therefore, they depend on reliably retrieving a complete operating system from the network to boot and continue operations.” *Id.*, 9:28-31. “At step 375, boot information is returned to the NC client 310. The boot information may include ... the name of the file to be loaded I not the NC client’s memory and executed, and the names and locations of shadow volumes, if any.” *Id.*, 10:19-24. In the second phase of bootstrapping, “... the NC client 310 sends a file transfer request to the NC server 320 specifying the bootfile identified in the NC server’s Bootp reply. At step 385, the NC server 320 responds by initiating the transfer of

the boot image. Upon receipt at the NC client 310, the boot image is stored in a local RAM of the NC client 310.” *Id.*, 10:27-32. Only after the entire file has been transferred and stored in the NC client’s RAM does the NC client begin executing the operating system program. “After the NC client 310 has received the boot image from the NC server 320, the RAM boot phase begins at step 390. During the RAM boot phase, the NC client 310 begins to execute the boot image out of the local RAM.” *Id.*, 10:33-36. Birse then goes on to give some examples of operating system boot images. “Finally, at step 395, the NC client 310 mounts the remote system and appliance volumes 174 and 176.” *Id.*, 10:45-46. This clearly shows that the OS is not paged in on demand as execution proceeds, as that would require that the (remote) system volume be already mounted. In addition, “At this point, a login may be presented on the NC client 310. Based on the user information, the user environment management process 182 may move all the components that are specific to this user, including preferences, to the user’s system environment.” *Id.*, 10:49-54.

82. Data that are changed during a session are stored in “shadow volumes.” “[O]ne or more shadow volumes, may be created for the NC client 150 and stored with the non-persistent client-specific data 184 for the purpose of storing user preferences and /or changes to the operating system.” *Id.*, 7:42-46. As noted above, the persistent user data is transferred from the shadow volume to the

persistent user data storage on the NC server when the user logs out to be available for subsequent logins. “[T]he user’s desktop preferences and other customized settings are all preserved from session to session and supplied to the NC client as it boots from the network, the user may login to any NC client on the network and have the same user experience.” *Id.*, 9:6-10. Thus the NC client is again cited as a user workstation hosting one user at a time.

83. “The clients folder 680 is an area that may be used to store non-persistent client-specific information, such as modifications to the shared operating system 660.” *Id.*, 12:52-54. The clients folder then has a subfolder for each client. “[C]lients folder 680 includes a folder for each NC client. For example, an NC #1 folder 690 is an area for storing client-specific informations, such as shadow image 695, corresponding to client #1.” *Id.*, 12:57-59. Thus shadow files are not shared and their indexes are not shareable.

84. There is no hierarchy or tree of COW shadows as the shadow copy is deleted when a user session ends, once the persistent data is written to the appropriate storage area as noted above. In order to save space, banding is used to store the shadow system volume. “[T]he sparse shadow system volume 1120 is written in band increments rather than as an entire file. A band is a predetermined number of blocks. For example, according to one embodiment bands are 128K (256 512-byte blocks). ... Using bands, the disk space used by the shadow volume

1120 will consist only of the actual data written (in band increments).” *Id.*, 15:25-32. Furthermore, since bands comprise multiple blocks, bands stored in the sparse shadow volume will contain unchanged blocks from the original boot image. This does not match the negative limitation recited in the claims of the '844 Patent where the leaf copies contain only new blocks and modified blocks of the root image.

85. In summary, Birse does not teach a branching copy-on-write file system, and neither does Birse teach paging the operating system image on demand.

E. Ground 1: Claims 1-13 Would Not Have Been Obvious In View of Sapuntzakis in combination with Holzmann.

86. The opinion of Dr. Zadok (EX1003) and the Petition do not address the key innovations in the '844 Patent. In addition to the distinctions between the '844 Patent and Exhibits 1005 and 1007 which I have described in the previous section, the references do not disclose the following.

- 1. [1.2] a plurality of second storage units configured to store leaf images of respective compute nodes, said leaf images including only additional data blocks not previously contained in said root image and changes made by respective compute nodes to the blocks of said root image, wherein said leaf images of respective compute nodes do not include blocks of said root image that are unchanged by respective compute nodes**

87. Sapuntzakis is aimed at virtual machine migration, that is, suspending a running virtual machine, collecting all pertinent data into an object, transferring

that object (a “capsule”) over any of a variety of networks to a destination host, then unpacking the capsule and activating the virtual machine at the destination. The virtual machine runs on a Type II hypervisor which can, and does, make use of the host operating system over which it runs.

88. Sapuntzakis discloses “capsules” that are objects encapsulating the entire machine state of a virtual machine, including in part its disks. These capsules are oriented toward user sessions, and not toward compute nodes. Indeed, the capsules are designed to migrate from one computer to another in order to save the state associated with a user until he or she logs out. In such a system one could expect at least one capsule per active user, plus stored capsules from users that have suspended their activities without logging out, and inactive capsules from which other capsules are derived when a new capsule is created. In contrast, in the system disclosed in the '844 Patent, there is one leaf image per compute node (in use or not), regardless of the number of users in the system.

89. Sapuntzakis contemplates sending the entire capsule to the destination workstation (EX1005 3.3), so that the files there could include the entirety of the root operating system image. While paging the capsule disks is discussed, it is done as a method to speed up start-up time, and does not preclude storage of the entirety of the root operating system as a shadow COW disk on the destination work station.

2. [1.3] a cache configured to cache blocks of said root image previously accessed by at least one of said compute nodes

90. Sapuntzakis does not disclose a disk server cache for the shared root image. The hash cache described by Sapuntzakis deals with a block cache at the user's workstation that holds blocks in the shadow COW disks, which must be checked not only for their presence (i.e., a copy of the block has been made to the local disk), but also for currency (i.e., has it been modified since the local copy was made). The system stores a hash of the block contents in a hash table indexed by the first several bits of the hash value, which points to a bucket with entries, each holding a full hash value along with a pointer to a file and an offset for blocks in the cache. The cache holds recently accessed blocks, and can pre-fetch blocks from the file as well. EX1005 4.4.1.

91. This cache exists at the user's workstation, and not at the networked disk server as disclosed by the '844 Patent. Moreover, it contains blocks that have not been accessed yet, contrary to the teachings of the '844 Patent.

3. Holzmann does not cure the shortcomings of Sapuntzakis

92. In the context of the '844 Patent, "cache" refers to a volatile memory cache located at the storage server to hold recently accessed blocks stored on its disks. The caching mentioned by Sapuntzakis is similar to the standard block cache that most operating systems implement, and it is located at the compute node itself.

93. Holzmann does not cure the shortcomings of Sapuntzakis. The cache disclosed by Holzmann, while it resides at the storage server, only maintains a single cache for the system image, which may be changed by servers without creating a leaf image. Furthermore, it is more of a traditional cache in that it employs cache lines, which are not identified with disk blocks. These have the typical cache line metadata of data tag, validity bit, locked bit, and server number data (this field is only common in distributed systems). The server number data is used to prevent incompatible changes by two different servers. In fact, if a second server attempts to change the same cache line, then it is returned an error condition rather than creating a new leaf image block.

4. There is a lack of motivation to combine Holzmann and Sapuntzakis

94. Sapuntzakis is expressly session-oriented and ephemeral. The paper's primary usage scenario explains that each user derives an individual capsule at login and that "the capsule migrates with the student as he commutes, and is destroyed when he logs out." EX1005 at 5 (Fig. 1). Disk state is implemented as a hierarchy of copy-on-write (COW) disks exposed to VMware via a shim library that intercepts "plain disk" I/O and redirects requests to a local disk server, which then resolves reads down a chain of immutable remote COW disks. EX1005, §4.3 at 6–7 (Fig. 2) (shim library between GSX Server and C library; redirection to local disk server; COW chain). To improve responsiveness, each host maintains

a local “shadow COW” that caches blocks fetched from the remote immutable COWs; cached blocks in the shadow COW can be reused for multiple capsules and across suspend/resume of a single capsule, avoiding network traffic for those blocks. EX1005, §4.3 at 7. For wide-area migration, Sapuntzakis employs a hash-based copy protocol (HCP) tailored to constrained links, noting throughput “well under 10 Mbps” and that HCP is “not intended for high-bandwidth LAN environments.” EX1005, §4.4 at 7.

95. Holzmann addresses a different problem: remote-booting multiple servers from a single operating system image stored on a solid-state disk, with centralized, volatile memory used to accelerate subsequent boots and OS operations. In Holzmann, a single copy of operating system data is loaded into device memory, and the control module segments that memory into per-server cache spaces (e.g., “Cache Space for First Server 401,” “Cache Space for Second Server 402”). EX1007, [0032]–[0034]; Fig. 4. Holzmann further discloses a non-associative cache organization in which each cache line holds a data tag, a validity bit, a locked bit, server-number data, and the data itself; validity and lock bits, together with the server number, coordinate concurrent use and restrict writes. EX1007, [0034]; Fig. 5; see also [0035], [0037] (cache population and write-admission behavior).

96. Given these teachings, a POSITA would recognize at least three technical mismatches that would have discouraged grafting Holzmann's cache into Sapuntzakis's capsules. First, there is a lifecycle mismatch. Holzmann's benefit depends on persistent, cross-boot reuse of OS blocks from a centralized device memory with per-server coordination, whereas Sapuntzakis deliberately scopes each capsule to a user session that ends at logout and is reconstructed thereafter. EX1005, at 5 (Fig. 1); EX1007, [0032]–[0034]. The premise of multi-boot reuse is orthogonal to Sapuntzakis's session-scoped lifecycle.

97. Second, there is a placement and consistency mismatch. Sapuntzakis places I/O translation and caching at the host, below VMware, with immutable remote COWs and per-host shadow COW caches tied to the capsule's storage path, which keeps coherence simple and localized. EX1005, §4.3, at 6–7. Holzmann moves the caching locus into a central storage appliance's volatile memory and coordinates access across servers using validity/lock bits and server-number metadata at a cache-line granularity. EX1007, [0034]; Fig. 5; ¶37. Integrating Holzmann's central, mutable cache into Sapuntzakis's immutable-upstream COW chain would relocate the trust and concurrency boundaries and require new protocols to reconcile line-locked, multi-writer cache semantics with Sapuntzakis's per-host shadow-cache model. Those are architectural changes, not routine substitutions.

98. Third, there is a problem-framing mismatch. Sapuntzakis optimizes for migration over constrained networks with per-host demand paging and a WAN-oriented HCP that is “not intended for high-bandwidth LAN environments,” while Holzmann assumes a centrally attached SSD serving many boots from a single OS image in a storage-area-network context. EX1005, §4.4 at 7; EX1007, [0030]–[0034]; Fig. 4. Dropping a centralized, multi-boot OS cache into Sapuntzakis’s stack would not address the stated bottlenecks and would conflict with the design choices that favor per-host shadow caching of immutable COWs.

99. Accordingly, the record reflects fundamental lifecycle, placement/consistency, and problem-framing mismatches that would have discouraged a person of ordinary skill and would require non-routine redesign rather than a predictable substitution.

100. For these reasons, grounded in the disclosures themselves, it is my opinion that a POSITA would not have had a motivation to graft Holzmann’s centralized, multi-boot OS cache into Sapuntzakis’s ephemeral, session-scoped capsules, and doing so would require significant re-architecture to lifecycle, placement, and consistency mechanics rather than a predictable use of known techniques.

5. A POSITA would be discouraged from forming the proposed combination

101. In addition, Holzmann explicitly criticizes per-server images and advances a single-image, multi-boot cache. Holzmann's background identifies the then-prevailing practice of keeping "a separate operating system image ... for each server" in a SAN remote-boot environment and calls it "wasteful and expensive," then states the need for booting "multiple servers ... using a single operating system image." EX1007, [0009]–[0010]. Holzmann responds by loading one copy of Operating System Data 400 into device memory and carving out per-server cache spaces (401, 402, 403) for persistent cross-boot reuse, with figures and text explaining this segmentation and reuse model. EX1007, [0032]–[0034]; Fig. 4. Holzmann then details cache-line metadata for shared use across servers—data tag 502, validity bit 503, locked bit 504, and server number data 505—to coordinate reads and writes at the centralized cache. EX1007, [0034]–[0037]; Fig. 5.

102. Sapuntzakis adopts the opposite lifecycle and reuse strategy. Sapuntzakis's core usage model is session-scoped and ephemeral: a user derives an individualized capsule at login and "the capsule migrates with the student as he commutes, and is destroyed when he logs out." EX100,5 §4.2 (university example). Rather than a centralized, cross-node boot cache, Sapuntzakis relies on host-local mechanisms tuned to migration and start-up latency: ballooning to compress memory state for faster start, EX1005, §4.1, demand paging of COW

disks with a shim and local disk server, and per-host shadow COW disks (one per compute server) that cache fetched blocks from immutable remote COWs. EX1005, §4.3; Fig. 3. For wide-area moves, Sapuntzakis uses hash-based copy (HCP) that it expressly says is “well under 10 Mbps” and “not intended for high-bandwidth LAN environments,” confirming the design’s WAN-migration focus rather than SAN-boot caching. EX1005, §4.4.

103. The teachings are divergent to a POSITA. First, the lifecycle expectations conflict. Holzmann’s benefit arises from persistent, cross-boot reuse of OS blocks from a centralized device cache serving many servers from one image, EX1007, [0032]–[0034]; Fig. 4, whereas Sapuntzakis intentionally resets state at the session boundary and reconstructs capsule derivatives thereafter. EX1005, §4.2. A system premised on multi-boot persistence is the opposite of a system designed to be destroyed at logout.

104. Second, the placement and consistency models are at odds. Holzmann centralizes the cache inside the storage appliance and coordinates cross-server access with validity and lock bits and server identifiers at cache-line granularity. EX1007, [0034]–[0037]; Fig. 5. Sapuntzakis localizes caching to per-host shadow COWs under a local disk server, with immutable upstream media that simplify coherence and avoid shared-device arbitration. EX1005, §4.3; Fig. 3. Integrating Holzmann’s central, mutable cache into Sapuntzakis’s immutable-upstream chain

and per-host shadow caches would invert the trust and concurrency boundaries and require new protocols, which is not a routine substitution.

105. Third, the bottlenecks and performance goals diverge. Holzmann assumes a SAN-attached SSD serving many server boots from a single OS image and optimizes for shared, high-speed boot reuse. EX1007, [0030]–[0034]; Fig. 4. Sapuntzakis targets migration over constrained links and per-host demand paging, with HCP expressly tuned to low bandwidth and “not intended for high-bandwidth LANs.” EX1005, §4.4. Adding a centralized, multi-boot OS cache does not address Sapuntzakis’s stated bottlenecks and undermines its session-scoped design.

106. In view of these disclosures, a POSITA would understand that Holzmann discourages per-server or per-capsule duplication by labeling it “wasteful,” while Sapuntzakis embraces individualized, session-scoped capsules with host-local caching and WAN-migration optimizations. EX1007, [0009]–[0010]; EX1005, §§4.1–4.4.

6. Relocating caching to Holzmann’s central appliance changes Sapuntzakis’s principle of operation

107. Relocating caching to Holzmann’s central appliance changes Sapuntzakis’s principle of operation for the following reasons.

108. (1) Read path inversion (from host-local COW/“shadow” checks to central OS-cache arbitration). In Sapuntzakis, a read traverses the host’s COW chain; on a miss to an immutable ancestor, the host fetches and deposits the block

into its local shadow COW for future reuse. EX1005, §4.3, Fig. 3. By contrast, Holzmann’s design has the appliance decide whether a requested OS block is in the central cache and return it from appliance memory, with segmentation keyed by the requesting server number. EX1007, ¶¶21–22, 24–25; Fig. 4. Moving Sapuntzakis’s cache to the appliance would eliminate the host-scoped shadow-COW principle and replace it with central cache-line lookups and arbitration—a different operational locus and decision process.

109. (2) Write-control model reversal (from “latest COW is always local” to central lock/validity gating). Sapuntzakis deliberately guarantees that writes never leave the host: “the latest COW disk is always local, [so] all writes are local.” EX1005, §4.3. Holzmann, however, controls write admission at the appliance using locked bits and server-number checks, denying writes that conflict with another server’s locked line. EX1007, ¶36; Fig. 5. Substituting Holzmann’s mechanism would require cross-machine locking and coherency semantics in the central device—precisely what Sapuntzakis avoids by keeping the writable layer local. That change is not a mere parameter tweak; it replaces the system’s write path and concurrency boundaries.

110. (3) Reuse mechanism and lifecycle shift (from per-host, session-scoped reuse to cross-boot, appliance-scoped reuse). Sapuntzakis’s reuse is host-local: cached blocks in the shadow COW “can be re-used for multiple capsules”

on that host, and the capsule itself is destroyed at logout. EX1005, §4.3; §4.2. Holzmann's reuse is appliance-global and cross-boot, tied to a single shared OS image with per-server cache partitions. EX1007, [0020]–[0022]; Fig. 4. Migrating to that model changes not just where data is cached, but what is being reused (OS boot blocks vs. COW-chain blocks), when it is reused (cross boots vs. within a session), and who arbitrates reuse (appliance vs. host)—all different operational principles.

111. (4) Trust and coherence boundaries (immutable-upstream vs. mutable central cache). Sapuntzakis's upstream COW ancestors are immutable, simplifying coherence; the host's shadow cache cannot conflict with upstream writers because there are none. EX1005, §4.3. Holzmann's cache is mutable and must police concurrent server writes with lock/validity metadata. EX1007, [0034]–[0037]; Fig. 5. Moving to Holzmann's locus would demand new correctness guarantees and error paths (e.g., returning errors on locked lines), altering the behavior and guarantees that Sapuntzakis's capsule runtime relies on.

112. In short, the proposed modification changes how reads are satisfied, how writes are controlled, and how reuse is achieved. It replaces Sapuntzakis's host-centric “immutable-upstream COW + per-host shadow” model with Holzmann's appliance-centric, lock-mediated OS cache. That is a change in principle of operation, not a predictable variation. And because Sapuntzakis

already addresses reuse and bandwidth with ballooning, demand paging, and HCP (EX1005, §§4.1, 4.3, 4.4; Figs. 3, 6, 8), there is no evidence-based “gap” that would have directed a POSITA to re-architect the system into Holzmann’s central, cross-server boot cache.

7. No Reasonable Expectation of Success in Combining the References

113. Holzmann’s cache is appliance-resident and keyed to a single shared operating-system image. The device loads one copy of operating system data into volatile memory and divides that memory into per-server cache spaces (e.g., “Operating System Data 400,” “Cache Space for First Server 401,” “Cache Space for Second Server 402”), with metadata at each cache line including a data tag, a validity bit, a locked bit, and server-number data to coordinate reuse and exclude writes that would corrupt the shared image. See, EX1007, Abstract; ¶¶30–37; Figs. 4–5. Sapuntzakis, by contrast, structures its runtime I/O path around a layered copy-on-write (COW) hierarchy accessed through a host-local disk server and a shadow COW per compute server, where “the latest COW disk is always local,” reads from immutable remote ancestors are demand-paged, and fetched blocks are cached in the local shadow COW “for multiple capsules” on that host. See EX1005, §4.3 at 6–7; Fig. 3. The capsule lifecycle is session-scoped and ephemeral, with the university example explaining that the capsule is destroyed at logout. See EX1005, §4.2.

114. A POSITA would see that “success” here is not merely grafting any cache onto any stack; it is maintaining Sapuntzakis’s stated invariants while improving or at least not degrading behavior. First, there is no taught or suggested coherence protocol that reconciles Holzmann’s per-server cache-line semantics (validity/lock/server-ID) with Sapuntzakis’s immutable-ancestor COW chain and host-scoped shadow caches. Holzmann assumes a mutable, centrally arbitrated cache for a single shared OS image, where the appliance decides whether a write is admitted based on the locked bit and the requesting server number. See, EX1007, ¶¶34–37; Fig. 5. Sapuntzakis assumes the opposite: ancestor disks are immutable, all writes terminate locally in the latest COW on the host, and remote data are pulled read-only and then cached in a per-host shadow COW. See, EX1005, §4.3 at 6–7; Fig. 3. Moving the caching locus from each host’s shadow COW into Holzmann’s central device would invert these assumptions and force new cross-machine locking, invalidation, and trust boundaries for every read and write path—mechanisms that neither reference teaches. Absent such a protocol, a POSITA could not reasonably expect correctness to be preserved.

115. Second, the expected performance is uncertain and likely unfavorable given the disclosed goals. Sapuntzakis’s performance hinges on locality: keeping the latest COW “always local,” caching remote ancestor blocks into a host-local shadow COW “for multiple capsules,” and reducing network traffic through

demand paging, ballooning, and hash-based copy (HCP) optimized for constrained links. See EX1005, §4.1; §4.3 at 6–7; §4.4. Relocating the hot blocks to a SAN-resident OS cache would reintroduce network round-trips into the steady-state read path and would bypass the host-local reuse that Sapuntzakis leverages across capsules on the same machine. Given the paper’s explicit emphasis that HCP is “not intended for high-bandwidth LAN environments,” a POSITA would not assume that centralizing the cache on a storage appliance will maintain the low-latency behavior of the existing host-local shadow-cache design. See EX1005 §4.4.

116. Third, the identity and addressing models are mismatched in ways that defeat a straightforward transplant. Holzmann’s cache metadata and “server number” field are designed to protect a single shared OS image across many servers as they are “booted,” and its memory segmentation (Operating System Data 400 / Cache 401 / Cache 402) presumes that the cache is keyed to the OS image, not to a layered COW lineage that differs per capsule. See, EX1007, ¶¶32–34; Fig. 4. Sapuntzakis’s data path, in contrast, is keyed to the capsule’s COW stack (with immutable ancestors and a host-local latest COW), and its reuse story explicitly targets “multiple capsules” on the same host through the shadow COW. See EX1005, §4.3 at 6–7; Fig. 3. Translating Holzmann’s line-oriented, server-segmented metadata into the block-addressed, per-capsule COW semantics in Sapuntzakis is not taught or suggested, and a POSITA would have no principled

way—on this record—to predict that such a translation would preserve isolation between capsules or prevent stale data exposure.

117. Finally, the capsule lifecycle itself undermines any expectation that Holzmann’s multi-boot cache will deliver benefits without side effects. Sapuntzakis purposely scopes state to a session and destroys the capsule at logout (EX1005, §4.2). That choice reduces long-lived cross-session coupling and complements the per-host shadow-cache model. By contrast, Holzmann’s benefits accrue when the same shared OS image is reused “as additional servers ... are booted,” persisting cached OS blocks across boots under central control (EX1007, [0030]–[0034]; Fig. 4). A POSITA would not expect that repurposing Holzmann’s central, cross-boot appliance cache into an ephemeral, per-session capsule regime—without any taught coherence or lifecycle integration—would preserve Sapuntzakis’s intended responsiveness and isolation.

118. In sum, because neither reference teaches the missing coherence, placement, and lifecycle mechanisms needed to make such a transplant work, and because Sapuntzakis already solves reuse and bandwidth at the host via shadow COW caching and WAN-oriented ballooning/HCP (EX1005, §§4.1, 4.3, 4.4; Fig. 3), a POSITA would lack a sound, evidence-based expectation that importing Holzmann’s centralized, multi-server boot cache would succeed.

F. Ground 2: Claims 14-27 Would Not Have Been Obvious In View of Sapuntzakis in combination with Federwisch

119. The shortcomings of Sapuntzakis have been detailed in the previous section, and its combination with Federwisch neither makes sense nor does it remedy its shortcomings. The references do not disclose the following.

- 1. [14.4] wherein further at least one of said compute nodes is configured to index said root image and provide the indexing results to another of said compute nodes.**

120. Sapuntzakis discusses no index at all, and Federwisch's index is used as an input that is peculiar to a given node, not as an output that can be shared with other nodes.

121. Federwisch is concerned with efficiently storing and transmitting snapshots of a system rather than serving active system blocks dynamically. Although Federwisch strives to maintain a single view of the system for all computers in the system, a file-system independent format is used for transmission of snapshots over the network. That is, Federwisch's snapshot describes changes to the file system in terms of deleted files, newly created files, and modified files, with these changes processed by the receiving file system in an appropriate order. Federwisch, [0016]. Since the source file system and destination file system may be completely different, whatever indexing scheme is used by one is unlikely to be of use to the other for ongoing activities. Whatever indexing Federwisch includes in the network messages can only be used universally to update changes to the file

system, and not replace the work that the destination must do to perform whatever indexing is necessary for its file system.

122. The index used by the scanner 820 is not shared with other nodes, but is used locally to ascertain which files and blocks have changed in an efficient way. Technically, the gulf is decisive and demonstrates that the claimed invention would not have been obvious to a POSITA.

G. Ground 3: Claims 1-13 Would Not Have Been Obvious In View of Birse in combination with Holzmann.

123. The opinion of the Dr. Zadok (EX1003) and the Petition do not address the key innovations in the '844 Patent. In addition to the distinctions between the '844 Patent and Exhibits 1006 and 1007 which I have described in the previous sections, the references do not disclose the following.

- 1. [1.2] a plurality of second storage units configured to store leaf images of respective compute nodes, said leaf images including only additional data blocks not previously contained in said root image and changes made by respective compute nodes to the blocks of said root image, wherein said leaf images of respective compute nodes do not include blocks of said root image that are unchanged by respective compute nodes**

124. Birse provides a three-volume runtime system including a read-only core system volume 922, a read-write user system volume 924, and a client-specific shadow system volume 930. The shadow system volume contains changes made by the particular user, but may contain an entire copy of the shared operating

system image 660. [EX1006, 12:58-64]. Thus this latter version of the shadow system volume does not meet the requirement that the leaf image does not include unmodified blocks.

125. The sparse shadow system volume that “is written in band increments rather than as an entire file” [*Id.*, 15:25-26] also fails to meet this limitation. The example band size given comprises 256 blocks. [*Id.*, 15:28]. If even one block in the band is changed, all the blocks of the band are written to the sparse shadow volume, including the 255 unchanged blocks. This version of the shadow system volume also fails to meet the limitation.

2. [1.3] a cache configured to cache blocks of said root image previously accessed by at least one of said compute nodes.;

126. Birse does not disclose a disk server cache for the shared root image. The closest thing to caching provided by Birse is the sparse shadow system volume, but this does not apply to the read-only core system volume, nor is it a block-level cache as disclosed in the '844 Patent.

127. Moreover, the hash cache of Birse is located on the compute node, not at the disk server, as in '844 Patent. Since this block cache is located at the compute node, it is not shared with other compute nodes, so blocks are not loaded into this cache according to the access patterns of other nodes. This version of the shadow system volume also fails to meet the limitation.

3. Holzmann does not cure the shortcomings of Birse

128. Holzmann teaches a different thing in a different place for a different purpose. Holzmann describes a solid state disk appliance that provides boot services to multiple servers from a single operating system image. Holzmann does not teach copy-on-write much less a plurality of leaf images, one per server.

129. Rather, Holzmann teaches server-side caching. However, the cache is a shared cache supporting access by multiple servers and not providing a separate cache for each server holding the parts of the operating system image that it has changed. The cache disclosed by Holzmann is organized around cache lines (which suggests a memory cache in which the data in the cache lines are much smaller than a memory page or a storage block), with mostly typical metadata fields per cache line, including a data tag, validity bit, locked bit, and server number data. EX1007 [0034]. “Each line 507 corresponds to a particular memory segment of the operating system data stored in memory portion 205.” *Id.* This again strongly indicates that the cache lines are intended to provide faster access to the data in them than regular RAM memory access would allow, which is consistent with the memory cache in the standard storage hierarchy (i.e., it sits between RAM/ROM and the registers on the processor) and is not consistent with caching disk blocks in RAM for faster access than would be afforded by disk access.

130. Moreover, when a server modifies a previously unchanged cache line, the locked bit is set and the server number data field is updated to remember which server made the change. Henceforth, writes to that cache line are only allowed by that same server, whereas other servers attempting to modify that cache line are returned an error condition. *Id.*, [0036]. Thus, there is no notion of multiple leaf images, each with its own cache, nor a block cache as described by the '844 patent. There is one current image of the operating system image that is shared by all servers, and is severely limited in how it may be modified at all.

4. A POSITA Would Not be Motivated to Combine the References

131. To meet the claim from Birse and Holzmann, a person of ordinary skill would have to: (i) introduce the '844 Patent's root/leaf composition model in which unchanged root blocks never appear in the leaf and writes go only to the leaf; (ii) relocate and re scope Holzmann's appliance cache so that it caches only blocks of the shared root image, not generic OS data, and so that it is consumed through a block level merge with each node's leaf beneath the file system; and (iii) strip out Holzmann's per line write semantics and server ownership locking — which exist precisely because Holzmann's cache is used as a read and write appliance cache for a single OS image — to conform to the root read/leaf write division of responsibility in the '844 Patent.

132. Accordingly, a POSITA starting from Holzmann's model would have no record based reason to re introduce per node, persistent image artifacts—precisely what claim 1 requires for the leaf images—because Holzmann's stated objective is to avoid them. Conversely, a skilled artisan starting from Birse's per client shadow scheme would not be steered toward Holzmann's single image boot appliance with a centrally partitioned OS cache.

H. Ground 4: Claims 14-27 Would Not Have Been Obvious In View of Birse in combination with Federwisch

133. The shortcomings of Birse have been detailed in the previous section, and its combination with Federwisch neither makes sense nor does it remedy its shortcomings. The references do not disclose the following.

- 1. [14.4] wherein further at least one of said compute nodes is configured to index said root image and provide the indexing results to another of said compute nodes.**

134. Birse fails to disclose indexing of any sort at all. It appears that Federwisch was combined with Birse in order to satisfy the claim language of indexing. However, to the extent that Federwisch discloses indexing, it is either as a means to describe changes in the incremental snapshot, or as a structure given to the scanner to find blocks that have changed in the write-anywhere system. The index is not computed or derived by any part of the reference, rather it is an existing file system component that the reference uses to its advantage.

IX. Conclusion

135. For the foregoing reasons, it is my opinion that claim 1 of the '844 Patent would not have been obvious over the prior art in the Petition.

136. Since Claims 2-6 are depend from Claim 1, they would not have been obvious over the prior art in the Petition for the same reasons as Claim 1.

137. Since Claim 7 is the method version of Claim 1, it is my opinion that it would not have been obvious over the prior art in the Petition . Similarly, as claims dependent on claim 7, claims 8-12 would not have been obvious either.

138. For the reasons elaborated above, it is my opinion that claim 14 of the '844 Patent would not have been obvious over the prior art in the Petition. Likewise, Claim 19 is the method version of claim 14 and would not have been obvious over the prior art in the Petition either.

139. Claims 15-18 and 20-22 depend on claims Claim 14 and 19, so they would not have been obvious over the prior art in the Petition for the same reasons as Claims 14 and 19.

140. Claim 23 is a stored program version of Claim 1, so it and its dependent claims 24-27 would not have been obvious over the prior art in the Petition for the same reasons as Claim .

141. All statements made herein of my own knowledge are true, and all statements made on information and belief are believed to be true. Further, I am

aware that these statements are made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under 18 U.S.C. § 1001. I declare under penalty of perjury that the foregoing is true and correct.

Executed on August 29, 2025, in Gainesville, Florida.

A handwritten signature in black ink, appearing to read "Dr. Richard Newman", written over a horizontal line.

Dr. Richard Newman