UNITED STATES PATENT AND TRADEMARK OFFICE

BEFORE THE PATENT TRIAL AND APPEAL BOARD

DOCKER, INC.,
Petitioner

v.

INTELLECTUAL VENTURES II LLC,
Patent Owner.

IPR2025-00840

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. 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 (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 Docker, Inc., ("Docker", 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 EX2013. 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 co-authored 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 '844 patent, which I have treated as December 30, 2004 for purposes of this Declaration. 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 1	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.")
EX1004	U.S. Patent No. 6,618,736 (EX1004) U.S. Patent No. 6,618,736 ("Menage"), titled "Template-Based Creation and Archival of File Systems," issued on September 9, 2003 from an application filed on March 9, 2001

	Description
EX1005	U.S. Patent No. 7,395,324 (EX1005) U.S. Patent No. 7,395,324 ("Murphy"), titled "Method and Apparatus for Maintaining a Computer System," issued on July 1, 2008 from an application filed on June 28, 2000.
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 No. 7,398,382 (EX1007) U.S. Patent Application Publication 7,398,382 ("Rothman"), titled "Method and Apparatus to Enhance Platform Boot Efficiency," issued on July 8, 2008 from an application filed on December 29, 2004

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 purposes of this Declaration. For purpose of this Declaration, I will apply the definition of POSITA as proposed 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 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. I understand that the parties dispute whether the '844 Patent is limited to a February 21, 2007 filing date, or is entitled to an earlier priority date. My opinions in this Declaration would not change if the Board were to adopt and apply either the December, 2004 or February, 2007 date.

25. At the time of the earliest priority date for the '844 Patent (December, 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.

26. While my own level of skill exceeded that of the ordinary level of skill as of December, 2004 and February, 2007, 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

27. 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 the ‘844 Patent in December, 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 terms of the challenged claims, including their plain meanings.

28. I reserve the right to further explain how skilled artisans would understand terms in the claims of the ‘844 Patent, 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

29. 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*.

30. Importantly, technology disclosed in the '844 Patent is independent of the operating system. For example, embodiments of the disclosed system 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) 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).

31. 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 UBS 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.

32. Caching of blocks 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.

33. Time required to boot ("bring-up time") a compute node is significantly reduced by introducing the Union Block Device (UBD), which in the disclosed embodiments 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 UBS to the compute node's memory.

34. 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 UBS in this case).

35. 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.

36. 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.

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

38. 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

39. **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.

40. **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.

41. 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.

42. **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, at the expense of perhaps allocating more space than is actually needed to store some data.

43. **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 it's 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.

44. **Operating system block caches**: The operating system often has a component that is dedicated to managing access to block storage devices such as disks. This component may handle requests for access to storage blocks by caching in-memory copies of recently accessed device blocks so that repeated accesses to the same block are efficient and minimize traffic over the communication lines to

the storage device, which may be shared with other devices. When a device storage block is requested by the system on behalf of a process, the block manager determines whether a copy of that block is in memory. If it is, that block is accessed and some meta-data is updated to reflect the access (e.g., the block is marked as "dirty" if it is written, and its last access time is updated in whatever way the cache management scheme dictates). If the block in not in memory, then some other block may have to be evicted (written out to device storage if it is "dirty") and its space allocated to hold a copy of the desired block, then the desired block is scheduled for retrieval from the device. All processes that have access to the system memory where the block cache is held may have access to any block in the cache, assuming that permissions and locks allow it, so the block cache is shared across all processes running on the same compute node, but is not shared across all compute nodes, as they do not share main memory.

45. **File caches**: In distributed systems, files may be accessed and transferred across networks. It is often desirable to store copies of recently accessed files in a location close to where future accesses may originate. For example, in access to the world wide web, a web proxy located on a campus network may store copies of web pages where any browser on the campus may be served a stored web page from the proxy without incurring the delay and congestion incurred by retrieving that page from the original server across the Internet. This form of

caching stores entire files and may check for coherence by checking the last modification timestamp of the local copy against the timestamp on the source copy, only downloading a fresh copy if the file has changed since it was last downloaded. Similar processes may be used for distributed file systems in which multiple copies of a file may exist within the system, and files may migrate from one location to another according to where they are being used.

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

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

47. 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 Menage

48. Menage refers to U.S. Patent No. 6,618,736 (EX1004) U.S. Patent No. 6,618,736 (“Menage”), titled “Template-Based Creation and Archival of File Systems,” issued on September 9, 2003 from an application filed on March 9, 2001.

49. Menage discloses techniques for solving the problems of creating efficient, separate file systems for virtual private servers that may be co-located on a single physical machine, and for backing up these systems with snapshots. Two sets of storage units are disclosed, with the first set shared and read-only by the

servers. A copy-on-write (COW) file system copies data onto one of the second set of private storage units when a server performs a write. A usage map is employed to indicate on which storage unit a data item is located. A set of private storage units and corresponding usage map form a "template," and these templates can form a "stack" in which writes occur only to the "top" template, whereas reads are made from the topmost template in the stack. That is, the templates in a stack are a sequence of COW disks. The file system is then a chain, rather than the branching file system disclosed in the '844 Patent.

50. To implement the redirection necessary for COW disks, Menage intercepts relevant system calls with a wrapper, which consults the usage map for the templates associated with the calling process to decide where to perform a read or write. Writes are always performed to the active (top) template's private storage unit, and the usage map is updated accordingly.

51. Caching the shared storage units is mentioned in passing at 10:4-6 of Menage, but no details are given. Similarly, caching the usage map is mentioned at 8:54-57 of Menage, but again, no details are given. In both cases, the cache is "in the computer memory" or in "a memory integrated with the storage device" "to provide high-speed access."

52. Menage also discusses efficient ways to take disk state snapshots for archival purposes, but these are not relevant to the '844 Patent. A POSITA

interested in solving the problems addressed by the ‘844 Patent, e.g., finding less cumbersome techniques of updating the boot image(s) for a cluster of computer nodes, would not have looked to archival systems for guidance.

B. Overview of Murphy

53. Murphy refers to U.S. Patent No. 7,395,324 (EX1005) U.S. Patent No. 7,395,324 (“Murphy”), titled “Method and Apparatus for Maintaining a Computer System,” issued on July 1, 2008 from an application filed on June 28, 2000. It focuses on systems for installing and maintaining files, file systems, BIOS data, operating system data and other data on computers in a networked environment.

54. To that end, Murphy discloses a software application program “to automatically configure and manage workstations.” EX1005, 5:39-40. When a client workstation boots over the network, the server provides a preloader application that “prepares the client for further configuration, …” *Id.*, 11:3-4. The client then interacts with the server to share its parameters and to perform updates as needed prior to booting the operating system.

55. As part of the process, a “File management program 412 may be used to manage file integrity on the local hard drive or another storage devices (sic) associated with client 202.” *Id.*, 14:50-52. The file management program creates an “index” of the files, which “may to [sic] contain a listing of the relevant files,

the size of the files, the file location and/or other information about the files." *Id*., 14:56-58. This is little more than a directory listing, and not the index contemplated by the '844 Patent. Furthermore, this mechanism provides for repair and updates to the client's own, private storage devices, not a shared storage device, and even if this index did satisfy the claim element, it would make no sense to share it with any other client, as it only relates to files on the client's own storage devices.

C. Overview of Birse

56. Birse refers to U.S. Patent No. 7,089,300 B1, 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.

57. 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.

58. 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.

59. "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.

60. "[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.

61. 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.

62. 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.

63. 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.

64. "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.

65. 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. 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.

D. Overview of Rothman

66. Rothman refers to U.S. Patent No. 7,398,382 (EX1007) U.S. Patent Application Publication 7,398,382 ("Rothman"), titled "Method and Apparatus to Enhance Platform Boot Efficiency," issued on July 8, 2008 from an application filed on December 29, 2004.

67. Rothman's concern is reducing boot time and network load when booting clients over a network. The method employed to accomplish this is to cache files at intermediate network nodes "proximate to the client." EX1007, 4:10. These network nodes may be switches, routers, bridges, or gateway servers, and are not themselves disk-level storage servers.

68. Caching is accomplished by the intermediate network node performing deep packet inspection when needed to discover boot activity and file transfers associated with it. OS boot images are cached at the intermediate network node and subsequent network boot request traffic transiting that node is served the cached file, if it matches. Alternatively, the boot server can direct the intermediate node to store files as it determines and server them to the client(s).

69. Version control and integrity are handled using checksums using an image checksum table (or "boot image verification table"). This enables the cached files to be updated when necessary. A caching device can ask the boot server to

verify the checksum of a file it plans to serve it to a client, and update it if the checksum does not match, before serving the client.

70. This form of caching is at the file level, is at an intermediate device rather than a network storage server, and operates at the file level, not at the block level. There is no notion of a COW file system, much less one that is branching and served live as multiple clients access it, nor is there any form of indexing mentioned. Only whole, unmodified files are cached, and they are served based on the client or the group to which a client belongs.

E. Ground 1: Claims 1 and 7 Would Not Have Been Obvious over Menage.

71. The reference does not disclose the following.

1. [1.3] a cache configured to cache blocks of said root image previously accessed by at least one of said compute nodes

72. Menage fails to disclose caching at the block level of the root image previously accessed by one of the compute nodes. Since Menage's COW is arranged as a stack of templates per virtual server, the only place where an access by one of the virtual servers could cause a block to be inserted into a block cache to the benefit of other virtual servers when they access the same block, is at the root level. Menage only mentions caching of the shared storage units in passing, without any indication of cache implementation. The possible caching of the usage map does not satisfy the requirements of the '844 Patent at all, since it is local to

the virtual server and hence not shared (or even shareable in a meaningful way), and it caches the usage map metadata used to resolve data accesses, not the data blocks being accessed.

73. Technically, a usage map cache and a root-image data-block cache serve different purposes, operate over different objects, and exhibit different behaviors:

74. Object of caching: Menage's cacheable object is the usage map (metadata); the '844 cache is the content itself—"blocks of the root image."

75. Population trigger: In the '844 system, population is demand-driven by reads—blocks are cached because a node accessed them (step 322) and the cache is updated "based on the read request" (step 345). Menage's remarks do not teach demand-driven population of shared-image data blocks keyed to prior access.

76. Serve-on-hit semantics: The '844 flow requires serving requested data from the cache when present (step 328). Menage's metadata cache can speed up decisions (which unit to read) but does not itself serve the data content

77. Menage fails to disclose a block level cache of root image data whose contents are populated on access by other virtual machines. Neither are any of the details of what is stored in the cache, what metadata is used for cache management, cache replacement policies, or any other parameters necessary to implement a cache given. Hence the claim element is not met.

F. Ground 2: Claims 14 and 19 Would Not Have Been Obvious In View of Menage in Combination with Murphy

78. The Petition's references do not disclose the following elements of Claim 14.

1. **[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;**

2. **[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**

79. Murphy does not disclose the branching, COW disk system required by the '844 Patent, with a root image that is accessed from a storage server on reads and leaf images stored on other storage devices. Murphy has files stored on the boot server that provide a client station with the files it needs, or needs to repair or update, which are thenceforth accessed locally on the client. This point is crucial in that no parts of files are actively shared across the client computers – each client computer obtains its own copies of the files it needs and proceeds from there.

Therefore, the final claim element is irrelevant to Murphy. There are no shared files to index across multiple clients.

3. [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.

80. Menage makes no mention of any indexing that is shared or even shareable between clients. The closest thing to an index mentioned in Menage is the usage map, which relates to a particular virtual private server and cannot be meaningfully shared with any other virtual server, let alone another compute node.

81. Murphy, as described above, does not share file system access amongst the clients. Each workstation has its own disk, which is repaired and updated as needed during the network boot process. The files, even if the same or partially the same as those on another client, are stored on the client's own storage device(s) and hence any file system index or index of storage blocks that may be made on one client cannot be used by another client.

82. The only index that is mentioned by Murphy amounts to a listing of files with some of their attributes exchanged with the server during the boot process. The server only uses this information to determine which files need to be provided to the client in order to update and repair the client's file system. Again, since these relate only to the specific client's file system, they are not useful to

other clients to manage their file systems. Knowing what files are corrupted or out of date on one client is of no use to another client.

83. The prior art fails to meet the requirements of Claim 14.

G. Ground 3: Claims 1 and 7 Would Not Have Been Obvious In View of Birse in Combination with Rothman.

84. 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;

85. Rothman discloses only caching of whole, unmodified files, and does not remotely meet this claim.

86. 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.

87. 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.;

88. This element requires a cache that is filled with blocks of a shared root image as they are accessed by compute nodes. The cache then provides those cached blocks to other compute nodes, evicting unused blocks and bringing new blocks into the cache according to compute node access patterns. Neither Birse nor Rothman teaches this form of caching.

89. 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 claimed in the '844 Patent. Instead, Birse's figures and description (e.g., the boot-server/NC-client arrangement and OS administration focus) underscore that Birse is an OS management framework, not a demand-driven root-image cache.

90. Moreover, the hash cache of Birse is located on the compute node, not at the disk server, as in the '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.

91. Rothman discloses caching, but only at the file level at intermediate network nodes. Files are not modified at these intermediate nodes by the clients, and indeed, the clients do not access these cached files once the boot process has concluded.

92. Rothman improves network boot efficiency by caching entire boot images and related files at network devices along the communication path between a boot server and its clients. EX1007, Abstract. A device decides to cache a file based on deep packet inspection and "message indicia," or alternatively by a server command over a management channel providing the network device with up-to-date copies of relevant system files and instructing it to which clients to deliver these. Clients never update the files over the network, only their local copies.

93. The system maintains integrity and version control using checksums that are exchanged with the boot server, which consults a "boot image verification table." EX1007, 13:44-46. The only removal of a file cached by an intermediate network node that is revealed by Rothman is at the direction the boot server when

the file is replaced by an updated version. This caching is performed at the level of files rather than file blocks, it is performed at an intermediate network node, not the storage server actively accessed by a client.

94. Moreover, the replacement policy disclosed by Rothman concerns file validity and currency, not keeping only the most-recently-accessed content subject to limitations of a limited cache size.

95. Neither of the references provide a cache that speeds up access to disk blocks on a storage server by storing recently used portions of a boot image in RAM according to activity by compute nodes in the system. The references do not satisfy this claim element.

96. The references also aim at incompatible contexts. Birse handles active COW disk file systems at run-time, while Rothman only focuses on pre-boot activities. Combining the two would not be intuitive to a POSITA, nor would it be likely to produce the results of the '844 Patent.

H. Ground 4: Claims 14 and 19 Would Not Have Been Obvious In View of Birse in Combination with Murphy.

97. The shortcomings of Birse have been detailed in the previous section, and its combination with Murphy does not remedy its shortcomings as detailed above as well. The references do not disclose the following.

1. **[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**

98. 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. Birse, 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.

99. 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.

100. Murphy operates during the boot process, and the client interacts with the server one-on-one. No files are shared amongst clients after the boot process has concluded, and no file are updated by the client external to the client.

2. **[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;**

101. Birse discusses COW disk concepts with shadow volumes, but the resolution for which disk to access is done by the driver on the compute node, not on the disk server itself. The '844 Patent states that the UBDs "are effectively low-level drivers that operate as an interface between the first and second storage devices and the file system of each compute node. The file system may reside on the server side of the system 200." EX1001, 5:61-65. The UBD in the '844 Patent allows the server to resolve requests for disk blocks and resides at the storage server, not at the compute node as in Birse.

102. The sparse shadow system volume that "is written in band increments rather than as an entire file" (EX1006, 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.

103. Again, Murphy does not address live file systems, much less branching COW file systems, after the boot process is ended.

3. [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.

104. Birse fails to disclose indexing of any sort at all.

105. As discussed above, Murphy's index is little more than a list of relevant files, not an index of the type required by the '844 Patent. Moreover, it is only relevant to the particular client, as it only refers to the files located on that client. It would make no sense for the client in Birse to share its "index" with another client.

106. Furthermore, it would make no sense to combine these teachings, as one relates to the live file system and the other to the boot process, with neither of them addressing the issues solved by the '844 Patent.

107. Claims 19 and 23 are different versions of Claim 14 and the references fail to satisfy them for the same reasons.

108. Concluding, the Petition fails to establish that the challenged claims are unpatentable.

IX. Conclusion

109. 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.

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

111. 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 for the same reasons as Claim 1. Similarly, as claims dependent on claim 7, claims 8-12 would not have been obvious either.

112. 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 for the same reasons as Claim 14.

113. 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.

114. 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 the Petition for the same reasons as Claim 1.

115. 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.

____

Dr. Richard Newman