EXHIBIT 22: SEXTON IN VIEW OF BUGNION

U.S. Patent No. 6,854,114
"Using a virtual machine instance as the basic unit of user execution in a server environment"
Inventor: Harlan Sexton et al.
Assignee: International Oracle International Corp.
Filing Date: Feb. 25, 2000
Issue Date: Feb. 8, 2005
("Sexton")

U.S. Patent No. 6,075,938[1]
"Virtual machine monitors for scalable multiprocessors"
Inventor: Bugnion et al.
Assignee: The Board of Trustees of the Leland Stanford Junior University
Filing Date: June 10, 1998
Issue Date: June 13, 2000
("Bugnion")

U.S. Patent No. 7,426,720	***Sexton* in view of *Bugnion***
1. A system for dynamic preloading of classes through memory space cloning of a master runtime system process, comprising:	*Sexton* provided a system for dynamic preloading of classes through memory space cloning of a master runtime system process. For example, *Sexton* is directed to "reducing startup costs and incremental memory requirements" associated with the instantiation of Java virtual machines; *Sexton* calls for a "the use of a shared state area [that] allows the various VM instantiations to share class definitions and other resources." *Sexton*, col. 5, ll. 53-57. *Sexton* provides techniques for instantiating separate Java virtual machines in a way that can "reduce[e] startup costs and incremental memory requirements of the Java virtual machines instantiated by the server." *Sexton*, col. 5, ll. 53-55. This streamlining of the Java process is accomplished by sharing certain data between the Virtual Machines. "Each VM instance has read-only access to the data that has been loaded into the shared state area, and therefore the VM instances do not contend with each other for access rights to that data. According to

[1] Note: *Bugnion* is cited on the cover of the *Sexton* patent.

U.S. Patent No. 7,426,720	*Sexton* in view of *Bugnion*
	one embodiment, the shared state area is used to store loaded Java classes." *Sexton*, col. 8, ll.45-48. This allows the Java process to proceed on each Virtual Machine without making two copies of the data. In other words, "[t]he non-session-specific data for the class, including the methods, method table and fields, are not duplicated in the session memory for each VM instance. Rather, all VM instances share read-only access to a single instantiation of the class, thus significantly reducing the memory requirements of VM instances (the per-session memory requirements)." *Sexton*, col. 8, ll.55-61.
A processor; A memory	*Sexton* provided a system that ran on a computer with a processor, e.g., a "microprocessor," and memory, e.g., "random access memory." "A virtual machine is software that acts as an interface between a computer program that has been compiled into instructions understood by the virtual machine and the microprocessor (or "hardware platform") that actually performs the program's instructions. Once a virtual machine has been provided for a platform, any program compiled for that virtual machine can run on that platform." *Sexton*, col. 2, ll.35-42. "FIG. 1 is a block diagram that illustrates a computer system 100 upon which an embodiment of the invention may be implemented. Computer system 100 includes a bus 102 or other communication mechanism for communicating information, and a processor 104 coupled with bus 102 for processing information. Computer system 100 also includes a main memory 106 , such as a random access memory (RAM) or other dynamic storage device, coupled to bus 102 for storing information and instructions to be executed by processor 104 . Main memory 106 also may be used for storing temporary variables or other intermediate information during execution of instructions to be executed by processor 104 . Computer system 100 further includes a read only memory (ROM) 108 or other static storage device coupled to bus 102 for storing static information and instructions for processor 104 . A storage device 110 , such as a magnetic disk or optical disk, is provided and coupled to bus 102 for storing information and instructions." *Sexton*, col. 9, ll.44-61.
a class preloader to obtain a representation of at least one class from a source definition provided as	*Sexton* provided a class preloader, e.g., the shared "state information," to obtain a representation of at least one class from a source definition provided as object-oriented program code, e.g., "the bytecode for all of the system classes."

U.S. Patent No. 7,426,720	*Sexton* in view of *Bugnion*
object-oriented program code;	"Because the threads execute within the same Java virtual machine, the user sessions share the state information required by the virtual machine. Such state information includes, for example, the bytecode for all of the system classes. While such state sharing tends to reduce the resource overhead required to concurrently service the requests, it presents reliability and security problems. Specifically, the bytecode being executed for first user in a first thread has access to information and resources that are shared with the bytecode being executed by a second user in a second thread. If either thread modifies or corrupts the shared information, or monopolizes the resources, the integrity of the other thread may be compromised." *Sexton*, col. 3, ll.51-63. "The database instance memory 220 is a shared memory area for storing data that is shared concurrently by more than one process. For example, this longer-duration memory area may be used to store the read-only data and instructions (e.g. bytecodes of JAVA classes) that are executed by the server processes 213 and 217. The database instance memory 220 is typically allocated and initialized at boot time of the database system 200 , before clients connect to the database system 200." *Sexton*, col. 6, ll.59-67. "When a database session is created, an area of the database memory 202 is allocated to store information for the database session. As illustrated in FIG. 2, session memories 222, 224, 226, and 228 have been allocated for clients 252, 254, 256, and 258, respectively, for each of which a separate database session has been created. Session memories 222, 224, 226, and 228 are memories used to store static data, i.e., data associated with a user that is preserved for the duration of a series of calls, especially between calls issued by a client during a single database session. JAVA class variables are one example of such static data." *Sexton*, col. 7, ll.1-11.
a master runtime system process to interpret and to instantiate the representation as a class definition in a memory space of the master runtime system process;	*Sexton* provided a master runtime system process to interpret and to instantiate the representation as a class definition in a memory space of the master runtime system process. *Sexton* creates a "VM data structure that is instantiated for a particular session" from a template input which is inherently a master runtime process.

U.S. Patent No. 7,426,720	*Sexton* in view of *Bugnion*
	"Techniques are provided for instantiating separate Java virtual machines for each a session established by a server. Because each session has its own virtual machine, the Java programs executed by the server for each user connected to the server are insulated from the Java programs executed by the server for all other users connected to the server. The separate VM instances can be created and run, for example, in separate units of execution that are managed by the operating system of the platform on which the server is executing. For example, the separate VM instances may be executed either as separate processes, or using separate system threads. Because the units of execution used to run the separate VM instances are provided by the operating system, the operating system is able to ensure that the appropriate degree of insulation exists between the VM instances." *Sexton*, col. 5, ll.29-44. "As mentioned above, in the conventional Java server model, each session initiated between a client and the server is handled by a single VM thread within a multi-threaded VM instance. In such an implementation, the Java virtual machine itself takes the form of a set of global variables accessible to all threads, where there is only one copy of each global variable. Unlike the conventional Java server, in one embodiment of the invention, an entire Java VM instance is spawned for every session made through the server. According to one implementation, each Java VM instance is spawned by instantiating a VM data structure in session memory. During execution, the state of a VM instance is modified by performing transformations on the VM data structure associated with the VM instance, and/or modifying the data contained therein. Specifically, the VM data structure that is instantiated for a particular session is passed as an input parameter to the server routines that are called during that session. Rather than accessing global variables that are shared among VM threads associated with different sessions, the routines access session-specific variables that are stored within the VM data structure that is passed to them. Consequently, the contention for resources that otherwise occurs between threads associated with different sessions is significantly reduced, because those threads are associated with different VM instances." *Sexton*, col. 7, l.61 – col. 8, l.18.
a runtime environment to clone the memory space as a child runtime system	*Sexton* provided a runtime environment to clone the memory space as a child runtime system process responsive to a process request and to execute the child runtime system process by,

U.S. Patent No. 7,426,720	*Sexton* in view of *Bugnion*
process responsive to a process request and to execute the child runtime system process;	for example, providing techniques for "instantiating separate Java virtual machines for each session established by a server." *Sexton* discloses that "each Java VM instance is spawned by instantiating a VM data structure in session memory." "Techniques are provided for instantiating separate Java virtual machines for each a session established by a server. Because each session has its own virtual machine, the Java programs executed by the server for each user connected to the server are insulated from the Java programs executed by the server for all other users connected to the server. The separate VM instances can be created and run, for example, in separate units of execution that are managed by the operating system of the platform on which the server is executing. For example, the separate VM instances may be executed either as separate processes, or using separate system threads. Because the units of execution used to run the separate VM instances are provided by the operating system, the operating system is able to ensure that the appropriate degree of insulation exists between the VM instances." *Sexton*, col. 5, ll.29-44. "As mentioned above, in the conventional Java server model, each session initiated between a client and the server is handled by a single VM thread within a multi-threaded VM instance. In such an implementation, the Java virtual machine itself takes the form of a set of global variables accessible to all threads, where there is only one copy of each global variable. Unlike the conventional Java server, in one embodiment of the invention, an entire Java VM instance is spawned for every session made through the server. According to one implementation, each Java VM instance is spawned by instantiating a VM data structure in session memory. During execution, the state of a VM instance is modified by performing transformations on the VM data structure associated with the VM instance, and/or modifying the data contained therein. Specifically, the VM data structure that is instantiated for a particular session is passed as an input parameter to the server routines that are called during that session. Rather than accessing global variables that are shared among VM threads associated with different sessions, the routines access session-specific variables that are stored within the VM data structure that is passed to them. Consequently, the contention for resources that otherwise occurs between threads associated with different sessions is significantly reduced, because those threads are associated with different VM instances."

U.S. Patent No. 7,426,720	***Sexton* in view of *Bugnion***
	Sexton, col. 7, l.61 – col. 8, l.18.
and a copy-on-write process cloning mechanism to instantiate the child runtime system process by copying references to the memory space of the master runtime system process into a separate memory space for the child runtime system process, and to defer copying of the memory space of the master runtime system process until the child runtime system process needs to modify the referenced memory space of the master runtime system process.	*Sexton* in view of *Bugnion* provided a copy-on-write process cloning mechanism to instantiate the child runtime system process by copying references to the memory space of the master runtime system process into a separate memory space for the child runtime system process, and to defer copying of the memory space of the master runtime system process until the child runtime system process needs to modify the referenced memory space of the master runtime system process. *Sexton* disclosed methods for a plurality of VMs to access certain "shared state" data such that "[t]he non-session-specific data for the class, including the methods, method table and fields, are not duplicated in the session memory for each VM instance." Given the goal of reducing session memory by sharing data between multiple Virtual Machines, one of ordinary skill in the art at the time of the invention could take the teachings of *Sexton* in combination with the *Bugnion* prior art and be in possession of the invention. Here, given the goal of the reduction of overhead of *Sexton*, it would be obvious to one of ordinary skill in the art to combine *Sexton* with the well-known copy on write technology, thereby placing the artisan in possession of the invention. *Bugnion* discloses that "[t]he virtual subnet and networking interfaces of Disco also use copy-on-write mappings to reduce copying and to allow for memory sharing." *Bugnion*, col. 15, l.66 - col. 16, l.1. "In addition, techniques are provided for reducing startup costs and incremental memory requirements of the Java virtual machines instantiated by the server. For example, the use of a shared state area allows the various VM instantiations to share class definitions and other resources. In addition, while it is actively processing a call, each VM instance has two components, a session-duration component and a call-duration component. Only the data that must persist in the VM between calls is stored in the session-duration component. Data that need not persist between calls is stored in the call-duration component, which is instantiated at the start of a call, and discarded at the termination of the call." *Sexton*, col. 5, ll.53-65. "According to one embodiment, the overhead associated with each VM instance is reduced by sharing certain data with other VM instances. The memory structure that contains the shared data is referred to herein as the shared state area. Each VM instance has read-only

U.S. Patent No. 7,426,720	*Sexton* in view of *Bugnion*
	access to the data that has been loaded into the shared state area, and therefore the VM instances do not contend with each other for access rights to that data. According to one embodiment, the shared state area is used to store loaded Java classes. The shared Java classes may include static variables whose values are session-specific. Therefore, according to one embodiment, a data structure, referred to herein as a "java_active_class", is instantiated in session space to store session-specific values (e.g. static variables) of a corresponding shared Java class. The non-session-specific data for the class, including the methods, method table and fields, are not duplicated in the session memory for each VM instance. Rather, all VM instances share read-only access to a single instantiation of the class, thus significantly reducing the memory requirements of VM instances (the per-session memory requirements). According to one embodiment, the java_active_class for each shared class further includes a pointer to the shared class to allow VM instances more efficient access to the shared class data." *Sexton*, col. 8, ll.40-64. "The VMM layer also maintains copy-on-write disks that allow virtual machines to transparently share main memory resources and disk storage resources, and performs dynamic page migration/replication that hides distributed characteristics of the physical memory resources from the operating systems. The VMM layer may also comprise a virtual memory resource interface to allow processes running on multiple virtual machines to share memory." *Bugnion*, col. 6, ll.29-36. "The interposition on all DMA requests offers an opportunity for Disco to share disk and memory resources among virtual machines. Disco's copy-on-write disks allow virtual machines to share both main memory and disk storage resources. Disco's virtual network devices allow virtual machines to communicate efficiently. The combination of these two mechanisms, detailed below, allows Disco to support a system-wide cache of disk blocks in memory that can be transparently shared between all the virtual machines." *Bugnion*, col. 14, ll.55-64. "Disco intercepts every disk request that DMAs data into memory. When a virtual machine requests to read a disk block that is already in main memory, Disco can process the request without going to disk. Furthermore, if the disk request is a multiple of the machine's page

U.S. Patent No. 7,426,720	***Sexton* in view of *Bugnion***
	size, Disco can process the DMA request by simply mapping the page into the virtual machine's physical memory. In order to preserve the semantics of a DMA operation, Disco maps the page read-only into the destination address page of the DMA. Attempts to modify a shared page will result in a copy-on-write fault handled internally by the monitor. Using this mechanism, multiple virtual machines accessing a shared disk end up sharing machine memory, The copy-on-write semantics means that the virtual machine is unaware of the sharing with the exception that disk requests can finish nearly instantly. Consider an environment running multiple virtual machines for scalability purposes. All the virtual machines can share the same root disk containing the kernel and application programs. The code and other read-only data stored on the disk will be DMA-ed into memory by the first virtual machine that accesses it. Subsequent requests will simply map the page specified to the DMA engine with out transferring any data. The result is shown in FIG. 4 where all virtual machines share these read-only pages. Effectively we get the memory sharing patterns expected of a single shared memory multiprocessor operating system even though the system runs multiple independent operating systems. To preserve the isolation of the virtual machines, disk writes must be kept private to the virtual machine that issues them. Disco logs the modified sectors so that the copy-on-write disk is never actually modified. For persistent disks, these modified sectors would be logged in a separate disk partition managed by Disco. To simplify our implementation, we only applied the concept of copy-on-write disks to non-persistent disks and kept the modified sectors in main memory whenever possible." *Bugnion*, col. 14, l.66 - col. 15, l.35. "The virtual subnet and networking interfaces of Disco also use copy-on-write mappings to reduce copying and to allow for memory sharing." *Bugnion*, col. 15, l.66 - col. 16, l.1.
2. A system according to claim 1, further comprising: a cache checker to determine whether the instantiated class definition is available in a local cache associated with the master runtime system process.	*Sexton* in view of *Bugnion* provided a cache checker to determine whether the instantiated class definition is available in a local cache associated with the master runtime system process. *Sexton* in view of *Bugnion* discloses the system of claim 1, as explained above with reference to claim 1. The system of *Sexton* inherently discloses a cache checker because the system checks for whether a Virtual Machine instance has been established already and, if not, instantiates one in session memory. Thus the limitation of a cache checker determining whether an instantiated class is available is inherent, as shown below.

U.S. Patent No. 7,426,720	***Sexton* in view of *Bugnion***
	"In addition, the resource manager maintains a global buffer cache that is transparently shared among the virtual machines using read-only mappings in portions of an address space of the virtual machines" *Bugnion*, col. 6, ll.25-29. "The present invention is implemented as a unique type of virtual machine monitor specially designed for scalable multiprocessors and their particular issues. The present invention differs from VM/370 and other virtual machines in several respects. Among others, it supports scalable shared-memory multiprocessors, handles modern operating systems, and transparently shares capabilities of copy-on-write disks and the global buffer cache. Whereas VM/370 mapped virtual disks to distinct volumes (partitions), the present invention has the notion of shared copy-on-write disks." *Bugnion*, col. 7, ll.2-13. "The present invention uses a combination of innovative emulation of the DMA engine and standard distributed file system protocols to support a global buffer cache that is transparently shared across all virtual machines." *Bugnion*, col. 7, ll.43-45. "The machines use a directory to maintain cache coherency, providing to the software the view of a shared-memory multiprocessor with non-uniform memory access times." *Bugnion*, col. 8, ll.31-34.
3. A system according to claim 2, further comprising: a class locator to locate the source definition if the instantiated class definition is unavailable in the local cache.	*Sexton* in view of *Bugnion* provided a class locator to locate the source definition if the instantiated class definition is unavailable in the local cache. *Sexton* in view of *Bugnion* discloses the system of claim 2, and *Sexton* further discloses the process of checking for an instantiation and, if "no VM instances bas been established . . . a VM instance for the session is instantiated in session memory." *Sexton*, col. 6, ll. 2-8. "As mentioned above, in the conventional Java server model, each session initiated between a client and the server is handled by a single VM thread within a multi-threaded VM instance. In such an implementation, the Java virtual machine itself takes the form of a set of global variables accessible to all threads, where there is only one copy of each global variable. Unlike the conventional Java server, in one embodiment of the invention, an entire Java VM instance

U.S. Patent No. 7,426,720	***Sexton* in view of *Bugnion***
	is spawned for every session made through the server. According to one implementation, each Java VM instance is spawned by instantiating a VM data structure in session memory. During execution, the state of a VM instance is modified by performing transformations on the VM data structure associated with the VM instance, and/or modifying the data contained therein. Specifically, the VM data structure that is instantiated for a particular session is passed as an input parameter to the server routines that are called during that session. Rather than accessing global variables that are shared among VM threads associated with different sessions, the routines access session-specific variables that are stored within the VM data structure that is passed to them. Consequently, the contention for resources that otherwise occurs between threads associated with different sessions is significantly reduced, because those threads are associated with different VM instances." *Sexton*, col. 7, l. 61 - col. 8, l.18.
4. A system according to claim 1, further comprising: a class resolver to resolve the class definition.	*Sexton* in view of *Bugnion* provided a class resolver to resolve the class definition. *Sexton* in view of *Bugnion* discloses the system of claim 1, and *Sexton* further discloses instantiation, resolution, and storage of the class definition by way of reference to "static data," which includes "JAVA class variables." "Because the threads execute within the same Java virtual machine, the user sessions share the state information required by the virtual machine. Such state information includes, for example, the bytecode for all of the system classes. While such state sharing tends to reduce the resource overhead required to concurrently service the requests, it presents reliability and security problems. Specifically, the bytecode being executed for first user in a first thread has access to information and resources that are shared with the bytecode being executed by a second user in a second thread. If either thread modifies or corrupts the shared information, or monopolizes the resources, the integrity of the other thread may be compromised." *Sexton*, col. 3, ll.51-63. "The database instance memory 220 is a shared memory area for storing data that is shared concurrently by more than one process. For example, this longer-duration memory area may be used to store the read-only data and instructions (e.g. bytecodes of JAVA classes) that are executed by the server processes 213 and 217. The database instance memory 220 is typically allocated and initialized at boot time of the database system 200 , before clients connect to the database system 200." *Sexton*, col. 6, ll.59-67.

U.S. Patent No. 7,426,720	***Sexton* in view of *Bugnion***
	"When a database session is created, an area of the database memory 202 is allocated to store information for the database session. As illustrated in FIG. 2, session memories 222, 224, 226, and 228 have been allocated for clients 252, 254, 256, and 258, respectively, for each of which a separate database session has been created. Session memories 222, 224, 226, and 228 are memories used to store static data, i.e., data associated with a user that is preserved for the duration of a series of calls, especially between calls issued by a client during a single database session. JAVA class variables are one example of such static data." *Sexton*, col. 7, ll.1-11.
5. A system according to claim 1, further comprising: at least one of a local and remote file system to maintain the source definition as a class file.	*Sexton* in view of *Bugnion* provided the system of claim 1, and further disclosed at least one of a local and remote file system to maintain the source definition as a class file, e.g., "a magnetic disk of a remote computer." "Various forms of computer readable media may be involved in carrying one or more sequences of one or more instructions to processor 104 for execution. For example, the instructions may initially be carried on a magnetic disk of a remote computer. The remote computer can load the instructions into its dynamic memory and send the instructions over a telephone line using a modem. A modem local to computer system 100 can receive the data on the telephone line and use an infra-red transmitter to convert the data to an infra-red signal. An infra-red detector can receive the data carried in the infra-red signal and appropriate circuitry can place the data on bus 102. Bus 102 carries the data to main memory 106, from which processor 104 retrieves and executes the instructions. The instructions received by main memory 106 may optionally be stored on storage device 110 either before or after execution by processor 104." *Sexton*, col. 10, ll. 45-60.
6. A system according to claim 1, further comprising: a process cloning mechanism to instantiate the child runtime system process by copying the memory space of the master runtime system process into a separate memory space for the child runtime system	*Sexton* in view of *Bugnion* provided the system of claim 1 further comprising a process cloning mechanism to instantiate the child runtime system process by copying the memory space of the master runtime system process into a separate memory space for the child runtime system process by employing a "shared state area," constructed based on the master runtime process and housed in a separate memory space, i.e., "not duplicated in the session memory for each VM instance."

U.S. Patent No. 7,426,720	*Sexton* in view of *Bugnion*
process.	"In addition, techniques are provided for reducing startup costs and incremental memory requirements of the Java virtual machines instantiated by the server. For example, the use of a shared state area allows the various VM instantiations to share class definitions and other resources. In addition, while it is actively processing a call, each VM instance has two components, a session-duration component and a call-duration component. Only the data that must persist in the VM between calls is stored in the session-duration component. Data that need not persist between calls is stored in the call-duration component, which is instantiated at the start of a call, and discarded at the termination of the call." *Sexton*, col. 5, ll.53-65. "According to one embodiment, the overhead associated with each VM instance is reduced by sharing certain data with other VM instances. The memory structure that contains the shared data is referred to herein as the shared state area. Each VM instance has read-only access to the data that has been loaded into the shared state area, and therefore the VM instances do not contend with each other for access rights to that data. According to one embodiment, the shared state area is used to store loaded Java classes. The shared Java classes may include static variables whose values are session-specific. Therefore, according to one embodiment, a data structure, referred to herein as a "java_active_class", is instantiated in session space to store session-specific values (e.g. static variables) of a corresponding shared Java class. The non-session-specific data for the class, including the methods, method table and fields, are not duplicated in the session memory for each VM instance. Rather, all VM instances share read-only access to a single instantiation of the class, thus significantly reducing the memory requirements of VM instances (the per-session memory requirements). According to one embodiment, the java_active_class for each shared class further includes a pointer to the shared class to allow VM instances more efficient access to the shared class data." *Sexton*, col. 8, ll.40-64. "The VMM layer also maintains copy-on-write disks that allow virtual machines to transparently share main memory resources and disk storage resources, and performs dynamic page migration/replication that hides distributed characteristics of the physical memory resources from the operating systems. The VMM layer may also comprise a virtual memory resource interface to allow processes running on multiple virtual machines to share

U.S. Patent No. 7,426,720	*Sexton* in view of *Bugnion*
	memory." *Bugnion*, col. 6, ll.29-36. "The interposition on all DMA requests offers an opportunity for Disco to share disk and memory resources among virtual machines. Disco's copy-on-write disks allow virtual machines to share both main memory and disk storage resources. Disco's virtual network devices allow virtual machines to communicate efficiently. The combination of these two mechanisms, detailed below, allows Disco to support a system-wide cache of disk blocks in memory that can be transparently shared between all the virtual machines." *Bugnion*, col. 14, ll.55-64. "Disco intercepts every disk request that DMAs data into memory. When a virtual machine requests to read a disk block that is already in main memory, Disco can process the request without going to disk. Furthermore, if the disk request is a multiple of the machine's page size, Disco can process the DMA request by simply mapping the page into the virtual machine's physical memory. In order to preserve the semantics of a DMA operation, Disco maps the page read-only into the destination address page of the DMA. Attempts to modify a shared page will result in a copy-on-write fault handled internally by the monitor. Using this mechanism, multiple virtual machines accessing a shared disk end up sharing machine memory, The copy-on-write semantics means that the virtual machine is unaware of the sharing with the exception that disk requests can finish nearly instantly. Consider an environment running multiple virtual machines for scalability purposes. All the virtual machines can share the same root disk containing the kernel and application programs. The code and other read-only data stored on the disk will be DMA-ed into memory by the first virtual machine that accesses it. Subsequent requests will simply map the page specified to the DMA engine with out transferring any data. The result is shown in FIG. 4 where all virtual machines share these read-only pages. Effectively we get the memory sharing patterns expected of a single shared memory multiprocessor operating system even though the system runs multiple independent operating systems. To preserve the isolation of the virtual machines, disk writes must be kept private to the virtual machine that issues them. Disco logs the modified sectors so that the copy-on-write disk is never actually modified. For persistent disks, these modified sectors would be logged

U.S. Patent No. 7,426,720	*Sexton* in view of *Bugnion*
	in a separate disk partition managed by Disco. To simplify our implementation, we only applied the concept of copy-on-write disks to non-persistent disks and kept the modified sectors in main memory whenever possible." *Bugnion*, col. 14, l.66 - col. 15, l.35. "The virtual subnet and networking interfaces of Disco also use copy-on-write mappings to reduce copying and to allow for memory sharing." *Bugnion*, col. 15, l.66 - col. 16, l.1.
7. A system according to claim 1, wherein the master runtime system process is caused to sleep relative to receiving the process request.	*Sexton* in view of *Bugnion* provided the system of claim 1 further comprising a system wherein the master runtime system process is caused to sleep relative to receiving the process request. *Sexton* stores necessary information in a "session-duration component" and "[a]t the end of the call, any data within the call-duration component that must persist between calls is transferred to the session-duration component in session memory, and the call-duration component is discarded." "In addition, techniques are provided for reducing startup costs and incremental memory requirements of the Java virtual machines instantiated by the server. For example, the use of a shared state area allows the various VM instantiations to share class definitions and other resources. In addition, while it is actively processing a call, each VM instance has two components, a session-duration component and a call-duration component. Only the data that must persist in the VM between calls is stored in the session-duration component. Data that need not persist between calls is stored in the call-duration component, which is instantiated at the start of a call, and discarded at the termination of the call. As shall be explained in greater detail hereafter, the state used by the VM is encapsulated into a "VM context" argument. The VM context is passed as an argument to all internal VM functions. Specifically, when the server receives a call during a session with a client, and the call requires execution of code by a virtual machine, the VM instance associated with that session is executed in a system thread or process. If no VM instance has been established for the session on which the call arrived, a VM instance for the session is instantiated in session memory. In response to the call, a call-duration component of the VM instance is instantiated in call memory. During the call, a VM context that includes pointers to the VM instance is passed as an argument to methods invoked within the VM instance. Those methods change the state of the VM by manipulating data within the VM instance. At the

U.S. Patent No. 7,426,720	***Sexton* in view of *Bugnion***
	end of the call, any data within the call-duration component that must persist between calls is transferred to the session-duration component in session memory, and the call-duration component is discarded." *Sexton*, col. 5, l.53 - col. 6, l. 17.
8. A system according to claim 1, wherein the object-oriented program code is written in the Java programming language.	*Sexton* in view of *Bugnion* provided the system of claim 1 wherein the object-oriented program code is written in the Java programming language. *Sexton* is largely directed to "Java virtual machines" and "Java source program." "A virtual machine is software that acts as an interface between a computer program that has been compiled into instructions understood by the virtual machine and the microprocessor (or "hardware platform") that actually performs the program's instructions. Once a virtual machine has been provided for a platform, any program compiled for that virtual machine can run on that platform. One popular virtual machine is known as the Java virtual machine (VM). The Java virtual machine specification defines an abstract rather than a real "machine" (or processor) and specifies an instruction set, a set of registers, a stack, a "garbage-collected heap," and a method area. The real implementation of this abstract or logically defined processor can be in other code that is recognized by the real processor or be built into the microchip processor itself. The output of "compiling" a Java source program (a set of Java language statements) is called bytecode. A Java virtual machine can either interpret the bytecode one instruction at a time (mapping it to one or more real microprocessor instructions) or the bytecode can be compiled further for the real microprocessor using what is called a just-in-time (JIT) compiler. The Java programming language supports multi-threading, and therefore Java virtual machines must incorporate multi-threading capabilities. Multi-threaded computing environments allow different parts of a program, known as threads, to execute simultaneously. In recent years, multithreaded computing environments have become more popular because of the favorable performance characteristics provided by multi-threaded applications." *Sexton*, col. 2, ll.36-65.
10. A method for dynamic preloading of classes through memory space cloning of a master runtime system process,	*Sexton* in view of *Bugnion* provided a method for dynamic preloading of classes through memory space cloning of a master runtime system process.

U.S. Patent No. 7,426,720	*Sexton* in view of *Bugnion*
comprising:	*Sexton* is directed to "reducing startup costs and incremental memory requirements" associated with the instantiation of Java virtual machines; *Sexton* calls for a "the use of a shared state area [that] allows the various VM instantiations to share class definitions and other resources." *Sexton*, col. 5, ll. 53-57. And *Bugnion* provides that "Disco can process the DMA request by simply mapping the page into the virtual machine's physical memory." *Bugnion*, col. 3, ll. 3-5.
executing a master runtime system process;	*Sexton* provided a method for executing a master runtime system process. *Sexton* discloses that "an entire Java VM instance is spawned for every session made through the server." Such an execution inherently must derive from a master runtime process. "A virtual machine is software that acts as an interface between a computer program that has been compiled into instructions understood by the virtual machine and the microprocessor (or "hardware platform") that actually performs the program's instructions. Once a virtual machine has been provided for a platform, any program compiled for that virtual machine can run on that platform." *Sexton*, col. 2, ll.35-42. "FIG. 1 is a block diagram that illustrates a computer system 100 upon which an embodiment of the invention may be implemented. Computer system 100 includes a bus 102 or other communication mechanism for communicating information, and a processor 104 coupled with bus 102 for processing information. Computer system 100 also includes a main memory 106 , such as a random access memory (RAM) or other dynamic storage device, coupled to bus 102 for storing information and instructions to be executed by processor 104 . Main memory 106 also may be used for storing temporary variables or other intermediate information during execution of instructions to be executed by processor 104 . Computer system 100 further includes a read only memory (ROM) 108 or other static storage device coupled to bus 102 for storing static information and instructions for processor 104 . A storage device 110 , such as a magnetic disk or optical disk, is provided and coupled to bus 102 for storing information and instructions." *Sexton*, col. 9, ll.44-61. "Techniques are provided for instantiating separate Java virtual machines for each a session established by a server. Because each session has its own virtual machine, the Java programs

U.S. Patent No. 7,426,720	***Sexton* in view of *Bugnion***
	executed by the server for each user connected to the server are insulated from the Java programs executed by the server for all other users connected to the server. The separate VM instances can be created and run, for example, in separate units of execution that are managed by the operating system of the platform on which the server is executing. For example, the separate VM instances may be executed either as separate processes, or using separate system threads. Because the units of execution used to run the separate VM instances are provided by the operating system, the operating system is able to ensure that the appropriate degree of insulation exists between the VM instances." *Sexton*, col. 5, ll.29-44. "As mentioned above, in the conventional Java server model, each session initiated between a client and the server is handled by a single VM thread within a multi-threaded VM instance. In such an implementation, the Java virtual machine itself takes the form of a set of global variables accessible to all threads, where there is only one copy of each global variable. Unlike the conventional Java server, in one embodiment of the invention, an entire Java VM instance is spawned for every session made through the server. According to one implementation, each Java VM instance is spawned by instantiating a VM data structure in session memory. During execution, the state of a VM instance is modified by performing transformations on the VM data structure associated with the VM instance, and/or modifying the data contained therein. Specifically, the VM data structure that is instantiated for a particular session is passed as an input parameter to the server routines that are called during that session. Rather than accessing global variables that are shared among VM threads associated with different sessions, the routines access session-specific variables that are stored within the VM data structure that is passed to them. Consequently, the contention for resources that otherwise occurs between threads associated with different sessions is significantly reduced, because those threads are associated with different VM instances." *Sexton*, col. 7, l.61 – col. 8, l.18.
obtaining a representation of at least one class from a source definition provided as object-oriented program code;	*Sexton* provided a method for obtaining a representation of at least one class from a source definition provided as object-oriented program code. *Sexton* provided a class preloader, e.g., the shared "state information," to obtain a representation of at least one class from a source definition provided as object-oriented program code, e.g., "the bytecode for all of the system classes."

U.S. Patent No. 7,426,720	*Sexton* in view of *Bugnion*
	"Because the threads execute within the same Java virtual machine, the user sessions share the state information required by the virtual machine. Such state information includes, for example, the bytecode for all of the system classes. While such state sharing tends to reduce the resource overhead required to concurrently service the requests, it presents reliability and security problems. Specifically, the bytecode being executed for first user in a first thread has access to information and resources that are shared with the bytecode being executed by a second user in a second thread. If either thread modifies or corrupts the shared information, or monopolizes the resources, the integrity of the other thread may be compromised." *Sexton*, col. 3, ll.51-63. "The database instance memory 220 is a shared memory area for storing data that is shared concurrently by more than one process. For example, this longer-duration memory area may be used to store the read-only data and instructions (e.g. bytecodes of JAVA classes) that are executed by the server processes 213 and 217. The database instance memory 220 is typically allocated and initialized at boot time of the database system 200 , before clients connect to the database system 200." *Sexton*, col. 6, ll.59-67. "When a database session is created, an area of the database memory 202 is allocated to store information for the database session. As illustrated in FIG. 2, session memories 222, 224, 226, and 228 have been allocated for clients 252, 254, 256, and 258, respectively, for each of which a separate database session has been created. Session memories 222, 224, 226, and 228 are memories used to store static data, i.e., data associated with a user that is preserved for the duration of a series of calls, especially between calls issued by a client during a single database session. JAVA class variables are one example of such static data." *Sexton*, col. 7, ll.1-11.
interpreting and instantiating the representation as a class definition in a memory space of the master runtime system process;	*Sexton* provided a method for interpreting and instantiating the representation as a class definition in a memory space of the master runtime system process. *Sexton* is directed to "reducing startup costs and incremental memory requirements" associated with the instantiation of Java virtual machines; *Sexton* calls for a "the use of a shared state area [that] allows the various VM instantiations to share class definitions and other resources."

U.S. Patent No. 7,426,720	*Sexton* in view of *Bugnion*
	Sexton, col. 5, ll. 53-57. "Techniques are provided for instantiating separate Java virtual machines for each a session established by a server. Because each session has its own virtual machine, the Java programs executed by the server for each user connected to the server are insulated from the Java programs executed by the server for all other users connected to the server. The separate VM instances can be created and run, for example, in separate units of execution that are managed by the operating system of the platform on which the server is executing. For example, the separate VM instances may be executed either as separate processes, or using separate system threads. Because the units of execution used to run the separate VM instances are provided by the operating system, the operating system is able to ensure that the appropriate degree of insulation exists between the VM instances." *Sexton*, col. 5, ll.29-44. "As mentioned above, in the conventional Java server model, each session initiated between a client and the server is handled by a single VM thread within a multi-threaded VM instance. In such an implementation, the Java virtual machine itself takes the form of a set of global variables accessible to all threads, where there is only one copy of each global variable. Unlike the conventional Java server, in one embodiment of the invention, an entire Java VM instance is spawned for every session made through the server. According to one implementation, each Java VM instance is spawned by instantiating a VM data structure in session memory. During execution, the state of a VM instance is modified by performing transformations on the VM data structure associated with the VM instance, and/or modifying the data contained therein. Specifically, the VM data structure that is instantiated for a particular session is passed as an input parameter to the server routines that are called during that session. Rather than accessing global variables that are shared among VM threads associated with different sessions, the routines access session-specific variables that are stored within the VM data structure that is passed to them. Consequently, the contention for resources that otherwise occurs between threads associated with different sessions is significantly reduced, because those threads are associated with different VM instances." *Sexton*, col. 7, l.61 – col. 8, l.18.

U.S. Patent No. 7,426,720	*Sexton* in view of *Bugnion*
and cloning the memory space as a child runtime system process responsive to a process request and executing the child runtime system process;	*Sexton* provided a method for cloning the memory space as a child runtime system process responsive to a process request and executing the child runtime system process. For example, *Sexton* discloses that "an entire Java VM instance is spawned for every session made through the server." "Techniques are provided for instantiating separate Java virtual machines for each a session established by a server. Because each session has its own virtual machine, the Java programs executed by the server for each user connected to the server are insulated from the Java programs executed by the server for all other users connected to the server. The separate VM instances can be created and run, for example, in separate units of execution that are managed by the operating system of the platform on which the server is executing. For example, the separate VM instances may be executed either as separate processes, or using separate system threads. Because the units of execution used to run the separate VM instances are provided by the operating system, the operating system is able to ensure that the appropriate degree of insulation exists between the VM instances." *Sexton*, col. 5, ll.29-44. "As mentioned above, in the conventional Java server model, each session initiated between a client and the server is handled by a single VM thread within a multi-threaded VM instance. In such an implementation, the Java virtual machine itself takes the form of a set of global variables accessible to all threads, where there is only one copy of each global variable. Unlike the conventional Java server, in one embodiment of the invention, an entire Java VM instance is spawned for every session made through the server. According to one implementation, each Java VM instance is spawned by instantiating a VM data structure in session memory. During execution, the state of a VM instance is modified by performing transformations on the VM data structure associated with the VM instance, and/or modifying the data contained therein. Specifically, the VM data structure that is instantiated for a particular session is passed as an input parameter to the server routines that are called during that session. Rather than accessing global variables that are shared among VM threads associated with different sessions, the routines access session-specific variables that are stored within the VM data structure that is passed to them. Consequently, the contention for resources that otherwise occurs between threads associated with different sessions is

U.S. Patent No. 7,426,720	***Sexton* in view of *Bugnion***
	significantly reduced, because those threads are associated with different VM instances." *Sexton*, col. 7, l.61 – col. 8, l.18.
wherein cloning the memory space as a child runtime system process involves instantiating the child runtime system process by copying references to the memory space of the master runtime system process into a separate memory space for the child runtime system process; and wherein copying references to the memory space of the master runtime system process defers copying of the memory space of the master runtime system process until the child runtime system process needs to modify the referenced memory space of the master runtime system process.	*Sexton* in view of *Bugnion* provided a method wherein cloning the memory space as a child runtime system process involves instantiating the child runtime system process by copying references to the memory space of the master runtime system process into a separate memory space for the child runtime system process; and wherein copying references to the memory space of the master runtime system process defers copying of the memory space of the master runtime system process until the child runtime system process needs to modify the referenced memory space of the master runtime system process. *Sexton* disclosed methods for a plurality of VMs to access certain "shared state" data such that "[t]he non-session-specific data for the class, including the methods, method table and fields, are not duplicated in the session memory for each VM instance." Given the goal of reducing session memory by sharing data between multiple Virtual Machines, one of ordinary skill in the art at the time of the invention could take the teachings of *Sexton* in combination with the *Bugnion* prior art and be in possession of the invention. Here, given the goal of the reduction of overhead of *Sexton*, it would be obvious to one of ordinary skill in the art to combine *Sexton* with the well-known copy on write technology, thereby placing the artisan in possession of the invention. *Bugnion* discloses that "[t]he virtual subnet and networking interfaces of Disco also use copy-on-write mappings to reduce copying and to allow for memory sharing." *Bugnion*, col. 15, l.66 - col. 16, l.1. "In addition, techniques are provided for reducing startup costs and incremental memory requirements of the Java virtual machines instantiated by the server. For example, the use of a shared state area allows the various VM instantiations to share class definitions and other resources. In addition, while it is actively processing a call, each VM instance has two components, a session-duration component and a call-duration component. Only the data that must persist in the VM between calls is stored in the session-duration component. Data that need not persist between calls is stored in the call-duration component, which is instantiated at the start of a call, and discarded at the termination of the call." *Sexton*, col. 5, ll.53-65.

U.S. Patent No. 7,426,720	*Sexton* in view of *Bugnion*
	"According to one embodiment, the overhead associated with each VM instance is reduced by sharing certain data with other VM instances. The memory structure that contains the shared data is referred to herein as the shared state area. Each VM instance has read-only access to the data that has been loaded into the shared state area, and therefore the VM instances do not contend with each other for access rights to that data. According to one embodiment, the shared state area is used to store loaded Java classes. The shared Java classes may include static variables whose values are session-specific. Therefore, according to one embodiment, a data structure, referred to herein as a "java_active_class", is instantiated in session space to store session-specific values (e.g. static variables) of a corresponding shared Java class. The non-session-specific data for the class, including the methods, method table and fields, are not duplicated in the session memory for each VM instance. Rather, all VM instances share read-only access to a single instantiation of the class, thus significantly reducing the memory requirements of VM instances (the per-session memory requirements). According to one embodiment, the java_active_class for each shared class further includes a pointer to the shared class to allow VM instances more efficient access to the shared class data." *Sexton*, col. 8, ll.40-64. "The VMM layer also maintains copy-on-write disks that allow virtual machines to transparently share main memory resources and disk storage resources, and performs dynamic page migration/replication that hides distributed characteristics of the physical memory resources from the operating systems. The VMM layer may also comprise a virtual memory resource interface to allow processes running on multiple virtual machines to share memory." *Bugnion*, col. 6, ll.29-36. "The interposition on all DMA requests offers an opportunity for Disco to share disk and memory resources among virtual machines. Disco's copy-on-write disks allow virtual machines to share both main memory and disk storage resources. Disco's virtual network devices allow virtual machines to communicate efficiently. The combination of these two mechanisms, detailed below, allows Disco to support a system-wide cache of disk blocks in memory that can be transparently shared between all the virtual machines." *Bugnion*, col. 14, ll.55-64.

U.S. Patent No. 7,426,720	***Sexton* in view of *Bugnion***
	"Disco intercepts every disk request that DMAs data into memory. When a virtual machine requests to read a disk block that is already in main memory, Disco can process the request without going to disk. Furthermore, if the disk request is a multiple of the machine's page size, Disco can process the DMA request by simply mapping the page into the virtual machine's physical memory. In order to preserve the semantics of a DMA operation, Disco maps the page read-only into the destination address page of the DMA. Attempts to modify a shared page will result in a copy-on-write fault handled internally by the monitor. Using this mechanism, multiple virtual machines accessing a shared disk end up sharing machine memory, The copy-on-write semantics means that the virtual machine is unaware of the sharing with the exception that disk requests can finish nearly instantly. Consider an environment running multiple virtual machines for scalability purposes. All the virtual machines can share the same root disk containing the kernel and application programs. The code and other read-only data stored on the disk will be DMA-ed into memory by the first virtual machine that accesses it. Subsequent requests will simply map the page specified to the DMA engine with out transferring any data. The result is shown in FIG. 4 where all virtual machines share these read-only pages. Effectively we get the memory sharing patterns expected of a single shared memory multiprocessor operating system even though the system runs multiple independent operating systems. To preserve the isolation of the virtual machines, disk writes must be kept private to the virtual machine that issues them. Disco logs the modified sectors so that the copy-on-write disk is never actually modified. For persistent disks, these modified sectors would be logged in a separate disk partition managed by Disco. To simplify our implementation, we only applied the concept of copy-on-write disks to non-persistent disks and kept the modified sectors in main memory whenever possible." *Bugnion*, col. 14, l.66 - col. 15, l.35. "The virtual subnet and networking interfaces of Disco also use copy-on-write mappings to reduce copying and to allow for memory sharing." *Bugnion*, col. 15, l.66 - col. 16, l.1.
11. A method according to claim 10, further comprising: determining whether the instantiated class definition is available in a local cache associated	*Sexton* in view of *Bugnion* disclosed the method of claim 10 further comprising a method for determining whether the instantiated class definition is available in a local cache associated with the master runtime system process. The system of *Sexton* inherently discloses a cache checker because the system checks for whether a Virtual Machine instance has been

U.S. Patent No. 7,426,720	*Sexton* in view of *Bugnion*
with the master runtime system process.	established already and, if not, instantiates one in session memory. Thus the limitation of a cache checker determining whether an instantiated class is available is inherent. "In addition, the resource manager maintains a global buffer cache that is transparently shared among the virtual machines using read-only mappings in portions of an address space of the virtual machines" *Bugnion*, col. 6, ll.25-29. "The present invention is implemented as a unique type of virtual machine monitor specially designed for scalable multiprocessors and their particular issues. The present invention differs from VM/370 and other virtual machines in several respects. Among others, it supports scalable shared-memory multiprocessors, handles modern operating systems, and transparently shares capabilities of copy-on-write disks and the global buffer cache. Whereas VM/370 mapped virtual disks to distinct volumes (partitions), the present invention has the notion of shared copy-on-write disks." *Bugnion*, col. 7, ll.2-13. "The present invention uses a combination of innovative emulation of the DMA engine and standard distributed file system protocols to support a global buffer cache that is transparently shared across all virtual machines." *Bugnion*, col. 7, ll.43-45. "The machines use a directory to maintain cache coherency, providing to the software the view of a shared-memory multiprocessor with non-uniform memory access times." *Bugnion*, col. 8, ll.31-34.
12. A method according to claim 11, further comprising: locating the source definition if the instantiated class definition is unavailable in the local cache.	*Sexton* in view of *Bugnion* provided the method of claim 11 further comprising a method step for locating the source definition if the instantiated class definition is unavailable in the local cache. For example, *Sexton* discloses the process of checking for an instantiation and, if "no VM instances bas been established . . . a VM instance for the session is instantiated in session memory." *Sexton*, col. 6, ll. 2-8. "As mentioned above, in the conventional Java server model, each session initiated between a client and the server is handled by a single VM thread within a multi-threaded VM instance. In such an implementation, the Java virtual machine itself takes the form of a set of global

U.S. Patent No. 7,426,720	*Sexton* in view of *Bugnion*
	variables accessible to all threads, where there is only one copy of each global variable. Unlike the conventional Java server, in one embodiment of the invention, an entire Java VM instance is spawned for every session made through the server. According to one implementation, each Java VM instance is spawned by instantiating a VM data structure in session memory. During execution, the state of a VM instance is modified by performing transformations on the VM data structure associated with the VM instance, and/or modifying the data contained therein. Specifically, the VM data structure that is instantiated for a particular session is passed as an input parameter to the server routines that are called during that session. Rather than accessing global variables that are shared among VM threads associated with different sessions, the routines access session-specific variables that are stored within the VM data structure that is passed to them. Consequently, the contention for resources that otherwise occurs between threads associated with different sessions is significantly reduced, because those threads are associated with different VM instances." *Sexton*, col. 7, l. 61 - col. 8, l.18.
13. A method according to claim 10, further comprising: resolving the class definition.	*Sexton* in view of *Bugnion* provided the method of claim 10 further comprising a method step for resolving the class definition. *Sexton* discloses instantiation, resolution, and storage of the class definition by way of reference to "static data," which includes "JAVA class variables." "Because the threads execute within the same Java virtual machine, the user sessions share the state information required by the virtual machine. Such state information includes, for example, the bytecode for all of the system classes. While such state sharing tends to reduce the resource overhead required to concurrently service the requests, it presents reliability and security problems. Specifically, the bytecode being executed for first user in a first thread has access to information and resources that are shared with the bytecode being executed by a second user in a second thread. If either thread modifies or corrupts the shared information, or monopolizes the resources, the integrity of the other thread may be compromised." *Sexton*, col. 3, ll.51-63. "The database instance memory 220 is a shared memory area for storing data that is shared concurrently by more than one process. For example, this longer-duration memory area may be used to store the read-only data and instructions (e.g. bytecodes of JAVA classes) that are

U.S. Patent No. 7,426,720	*Sexton* in view of *Bugnion*
	executed by the server processes 213 and 217. The database instance memory 220 is typically allocated and initialized at boot time of the database system 200 , before clients connect to the database system 200." *Sexton*, col. 6, ll.59-67. "When a database session is created, an area of the database memory 202 is allocated to store information for the database session. As illustrated in FIG. 2, session memories 222, 224, 226, and 228 have been allocated for clients 252, 254, 256, and 258, respectively, for each of which a separate database session has been created. Session memories 222, 224, 226, and 228 are memories used to store static data, i.e., data associated with a user that is preserved for the duration of a series of calls, especially between calls issued by a client during a single database session. JAVA class variables are one example of such static data." *Sexton*, col. 7, ll.1-11.
14. A method according to claim 10, further comprising: maintaining the source definition as a class file on at least one of a local and remote file system.	*Sexton* in view of *Bugnion* provided the method of claim 10 further comprising a method step for maintaining the source definition as a class file on at least one of a local and remote file system, e.g., "a magnetic disk of a remote computer." "Various forms of computer readable media may be involved in carrying one or more sequences of one or more instructions to processor 104 for execution. For example, the instructions may initially be carried on a magnetic disk of a remote computer. The remote computer can load the instructions into its dynamic memory and send the instructions over a telephone line using a modem. A modem local to computer system 100 can receive the data on the telephone line and use an infra-red transmitter to convert the data to an infra-red signal. An infra-red detector can receive the data carried in the infra-red signal and appropriate circuitry can place the data on bus 102. Bus 102 carries the data to main memory 106, from which processor 104 retrieves and executes the instructions. The instructions received by main memory 106 may optionally be stored on storage device 110 either before or after execution by processor 104." *Sexton*, col. 10, ll. 45-60.
15. A method according to claim 10, further comprising: instantiating the child runtime system process by	*Sexton* in view of *Bugnion* provided the method of claim 10 further comprising a method step for instantiating the child runtime system process by copying the memory space of the master runtime system process into a separate memory space for the child runtime system

U.S. Patent No. 7,426,720	***Sexton* in view of *Bugnion***
copying the memory space of the master runtime system process into a separate memory space for the child runtime system process.	process by employing a "shared state area," constructed based on the master runtime process and housed in a separate memory space, i.e., "not duplicated in the session memory for each VM instance." "In addition, techniques are provided for reducing startup costs and incremental memory requirements of the Java virtual machines instantiated by the server. For example, the use of a shared state area allows the various VM instantiations to share class definitions and other resources. In addition, while it is actively processing a call, each VM instance has two components, a session-duration component and a call-duration component. Only the data that must persist in the VM between calls is stored in the session-duration component. Data that need not persist between calls is stored in the call-duration component, which is instantiated at the start of a call, and discarded at the termination of the call." *Sexton*, col. 5, ll.53-65. "According to one embodiment, the overhead associated with each VM instance is reduced by sharing certain data with other VM instances. The memory structure that contains the shared data is referred to herein as the shared state area. Each VM instance has read-only access to the data that has been loaded into the shared state area, and therefore the VM instances do not contend with each other for access rights to that data. According to one embodiment, the shared state area is used to store loaded Java classes. The shared Java classes may include static variables whose values are session-specific. Therefore, according to one embodiment, a data structure, referred to herein as a "java_active_class", is instantiated in session space to store session-specific values (e.g. static variables) of a corresponding shared Java class. The non-session-specific data for the class, including the methods, method table and fields, are not duplicated in the session memory for each VM instance. Rather, all VM instances share read-only access to a single instantiation of the class, thus significantly reducing the memory requirements of VM instances (the per-session memory requirements). According to one embodiment, the java_active_class for each shared class further includes a pointer to the shared class to allow VM instances more efficient access to the shared class data." *Sexton*, col. 8, ll.40-64. "The VMM layer also maintains copy-on-write disks that allow virtual machines to

U.S. Patent No. 7,426,720	*Sexton* in view of *Bugnion*
	transparently share main memory resources and disk storage resources, and performs dynamic page migration/replication that hides distributed characteristics of the physical memory resources from the operating systems. The VMM layer may also comprise a virtual memory resource interface to allow processes running on multiple virtual machines to share memory." *Bugnion*, col. 6, ll.29-36. "The interposition on all DMA requests offers an opportunity for Disco to share disk and memory resources among virtual machines. Disco's copy-on-write disks allow virtual machines to share both main memory and disk storage resources. Disco's virtual network devices allow virtual machines to communicate efficiently. The combination of these two mechanisms, detailed below, allows Disco to support a system-wide cache of disk blocks in memory that can be transparently shared between all the virtual machines." *Bugnion*, col. 14, ll.55-64. "Disco intercepts every disk request that DMAs data into memory. When a virtual machine requests to read a disk block that is already in main memory, Disco can process the request without going to disk. Furthermore, if the disk request is a multiple of the machine's page size, Disco can process the DMA request by simply mapping the page into the virtual machine's physical memory. In order to preserve the semantics of a DMA operation, Disco maps the page read-only into the destination address page of the DMA. Attempts to modify a shared page will result in a copy-on-write fault handled internally by the monitor. Using this mechanism, multiple virtual machines accessing a shared disk end up sharing machine memory, The copy-on-write semantics means that the virtual machine is unaware of the sharing with the exception that disk requests can finish nearly instantly. Consider an environment running multiple virtual machines for scalability purposes. All the virtual machines can share the same root disk containing the kernel and application programs. The code and other read-only data stored on the disk will be DMA-ed into memory by the first virtual machine that accesses it. Subsequent requests will simply map the page specified to the DMA engine with out transferring any data. The result is shown in FIG. 4 where all virtual machines share these read-only pages. Effectively we get the memory sharing patterns expected of a single shared memory multiprocessor operating system even though the system runs multiple independent operating systems.

U.S. Patent No. 7,426,720	***Sexton* in view of *Bugnion***
	To preserve the isolation of the virtual machines, disk writes must be kept private to the virtual machine that issues them. Disco logs the modified sectors so that the copy-on-write disk is never actually modified. For persistent disks, these modified sectors would be logged in a separate disk partition managed by Disco. To simplify our implementation, we only applied the concept of copy-on-write disks to non-persistent disks and kept the modified sectors in main memory whenever possible." *Bugnion*, col. 14, l.66 - col. 15, l.35. "The virtual subnet and networking interfaces of Disco also use copy-on-write mappings to reduce copying and to allow for memory sharing." *Bugnion*, col. 15, l.66 - col. 16, l.1.
16. A method according to claim 10, further comprising: causing the master runtime system process to sleep relative to receiving the process request.	*Sexton* in view of *Bugnion* provided the method of claim 10 further comprising a method step for causing the master runtime system process to sleep relative to receiving the process request. For example, *Sexton* stores necessary information in a "session-duration component" and "[a]t the end of the call, any data within the call-duration component that must persist between calls is transferred to the session-duration component in session memory, and the call-duration component is discarded." "In addition, techniques are provided for reducing startup costs and incremental memory requirements of the Java virtual machines instantiated by the server. For example, the use of a shared state area allows the various VM instantiations to share class definitions and other resources. In addition, while it is actively processing a call, each VM instance has two components, a session-duration component and a call-duration component. Only the data that must persist in the VM between calls is stored in the session-duration component. Data that need not persist between calls is stored in the call-duration component, which is instantiated at the start of a call, and discarded at the termination of the call. As shall be explained in greater detail hereafter, the state used by the VM is encapsulated into a "VM context" argument. The VM context is passed as an argument to all internal VM functions. Specifically, when the server receives a call during a session with a client, and the call requires execution of code by a virtual machine, the VM instance associated with that session is executed in a system thread or process. If no VM instance has been established for the session on which the call arrived, a VM instance for the session is instantiated in session memory. In response to the call, a call-duration component of the VM instance is

U.S. Patent No. 7,426,720	***Sexton* in view of *Bugnion***
	instantiated in call memory. During the call, a VM context that includes pointers to the VM instance is passed as an argument to methods invoked within the VM instance. Those methods change the state of the VM by manipulating data within the VM instance. At the end of the call, any data within the call-duration component that must persist between calls is transferred to the session-duration component in session memory, and the call-duration component is discarded." *Sexton*, col. 5, l.53 - col. 6, l. 17.
17. A method according to claim 10, wherein the object-oriented program code is written in the Java programming language.	*Sexton* in view of *Bugnion* provided a method wherein the object-oriented program code is written in the Java programming language. *Sexton* is largely directed to "Java virtual machines" and "Java source program," as shown in the excerpts below. "A virtual machine is software that acts as an interface between a computer program that has been compiled into instructions understood by the virtual machine and the microprocessor (or "hardware platform") that actually performs the program's instructions. Once a virtual machine has been provided for a platform, any program compiled for that virtual machine can run on that platform. One popular virtual machine is known as the Java virtual machine (VM). The Java virtual machine specification defines an abstract rather than a real "machine" (or processor) and specifies an instruction set, a set of registers, a stack, a "garbage-collected heap," and a method area. The real implementation of this abstract or logically defined processor can be in other code that is recognized by the real processor or be built into the microchip processor itself. The output of "compiling" a Java source program (a set of Java language statements) is called bytecode. A Java virtual machine can either interpret the bytecode one instruction at a time (mapping it to one or more real microprocessor instructions) or the bytecode can be compiled further for the real microprocessor using what is called a just-in-time (JIT) compiler. The Java programming language supports multi-threading, and therefore Java virtual machines must incorporate multi-threading capabilities. Multi-threaded computing environments allow different parts of a program, known as threads, to execute simultaneously. In recent years, multithreaded computing environments have become more popular because of the favorable performance characteristics provided by multi-threaded

U.S. Patent No. 7,426,720	***Sexton* in view of *Bugnion***
	applications." *Sexton*, col. 2, ll.36-65.
19. A computer-readable storage medium holding code for performing the method according to claim 10.	*Sexton* in view of *Bugnion* provided the method steps of claim 10, and *Sexton* further discloses a computer-readable storage medium holding code, e.g., "random access memory," for performing the method according to claim 10. "A virtual machine is software that acts as an interface between a computer program that has been compiled into instructions understood by the virtual machine and the microprocessor (or "hardware platform") that actually performs the program's instructions. Once a virtual machine has been provided for a platform, any program compiled for that virtual machine can run on that platform." *Sexton*, col. 2, ll.35-42. "FIG. 1 is a block diagram that illustrates a computer system 100 upon which an embodiment of the invention may be implemented. Computer system 100 includes a bus 102 or other communication mechanism for communicating information, and a processor 104 coupled with bus 102 for processing information. Computer system 100 also includes a main memory 106 , such as a random access memory (RAM) or other dynamic storage device, coupled to bus 102 for storing information and instructions to be executed by processor 104 . Main memory 106 also may be used for storing temporary variables or other intermediate information during execution of instructions to be executed by processor 104 . Computer system 100 further includes a read only memory (ROM) 108 or other static storage device coupled to bus 102 for storing static information and instructions for processor 104 . A storage device 110 , such as a magnetic disk or optical disk, is provided and coupled to bus 102 for storing information and instructions." *Sexton*, col. 9, ll.44-61. "Various forms of computer readable media may be involved in carrying one or more sequences of one or more instructions to processor 104 for execution. For example, the instructions may initially be carried on a magnetic disk of a remote computer. The remote computer can load the instructions into its dynamic memory and send the instructions over a telephone line using a modem. A modem local to computer system 100 can receive the data on the telephone line and use an infra-red transmitter to convert the data to an infra-red signal. An infra-red detector can receive the data carried in the infra-red signal and

U.S. Patent No. 7,426,720	*Sexton* in view of *Bugnion*
	appropriate circuitry can place the data on bus 102. Bus 102 carries the data to main memory 106, from which processor 104 retrieves and executes the instructions. The instructions received by main memory 106 may optionally be stored on storage device 110 either before or after execution by processor 104." *Sexton*, col. 10, ll. 45-60.
20. An apparatus for dynamic preloading of classes through memory space cloning of a master runtime system process, comprising:	*Sexton* in view of *Bugnion* provided an apparatus for dynamic preloading of classes through memory space cloning of a master runtime system process. *Sexton* is directed to "reducing startup costs and incremental memory requirements" associated with the instantiation of Java virtual machines; *Sexton* calls for a "the use of a shared state area [that] allows the various VM instantiations to share class definitions and other resources." *Sexton*, col. 5, ll. 53-57.
A processor; A memory means for executing a master runtime system process;	*Sexton* provided an apparatus with a processor, e.g., a "microprocessor," and a memory, e.g., "random access memory," means for executing a master runtime system process. "A virtual machine is software that acts as an interface between a computer program that has been compiled into instructions understood by the virtual machine and the microprocessor (or "hardware platform") that actually performs the program's instructions. Once a virtual machine has been provided for a platform, any program compiled for that virtual machine can run on that platform." *Sexton*, col. 2, ll.35-42. "FIG. 1 is a block diagram that illustrates a computer system 100 upon which an embodiment of the invention may be implemented. Computer system 100 includes a bus 102 or other communication mechanism for communicating information, and a processor 104 coupled with bus 102 for processing information. Computer system 100 also includes a main memory 106 , such as a random access memory (RAM) or other dynamic storage device, coupled to bus 102 for storing information and instructions to be executed by processor 104 . Main memory 106 also may be used for storing temporary variables or other intermediate information during execution of instructions to be executed by processor 104 . Computer system 100 further includes a read only memory (ROM) 108 or other static storage device coupled to bus 102 for storing static information and instructions for processor 104 . A storage device 110 , such as a magnetic disk or optical disk, is provided and coupled to bus 102 for storing information and instructions." *Sexton*, col. 9, ll.44-61.

U.S. Patent No. 7,426,720	*Sexton* in view of *Bugnion*
means for obtaining a representation of at least one class from a source definition provided as object-oriented program code;	*Sexton* provided an apparatus with a means for obtaining a representation of at least one class from a source definition provided as object-oriented program code. *Sexton* provided a class preloader, e.g., the shared "state information," to obtain a representation of at least one class from a source definition provided as object-oriented program code, e.g., "the bytecode for all of the system classes." "Because the threads execute within the same Java virtual machine, the user sessions share the state information required by the virtual machine. Such state information includes, for example, the bytecode for all of the system classes. While such state sharing tends to reduce the resource overhead required to concurrently service the requests, it presents reliability and security problems. Specifically, the bytecode being executed for first user in a first thread has access to information and resources that are shared with the bytecode being executed by a second user in a second thread. If either thread modifies or corrupts the shared information, or monopolizes the resources, the integrity of the other thread may be compromised." *Sexton*, col. 3, ll.51-63. "The database instance memory 220 is a shared memory area for storing data that is shared concurrently by more than one process. For example, this longer-duration memory area may be used to store the read-only data and instructions (e.g. bytecodes of JAVA classes) that are executed by the server processes 213 and 217. The database instance memory 220 is typically allocated and initialized at boot time of the database system 200 , before clients connect to the database system 200." *Sexton*, col. 6, ll.59-67. "When a database session is created, an area of the database memory 202 is allocated to store information for the database session. As illustrated in FIG. 2, session memories 222, 224, 226, and 228 have been allocated for clients 252, 254, 256, and 258, respectively, for each of which a separate database session has been created. Session memories 222, 224, 226, and 228 are memories used to store static data, i.e., data associated with a user that is preserved for the duration of a series of calls, especially between calls issued by a client during a single database session. JAVA class variables are one example of such static data." *Sexton*, col. 7, ll.1-11.

U.S. Patent No. 7,426,720	*Sexton* in view of *Bugnion*
means for interpreting and means for instantiating the representation as a class definition in a memory space of the master runtime system process;	*Sexton* provided an apparatus with a means for interpreting and means for instantiating the representation as a class definition in a memory space of the master runtime system process. *Sexton* creates a "VM data structure that is instantiated for a particular session" from a template input which is inherently a master runtime process. This can include a wide range of implementations, including class definitions. "Techniques are provided for instantiating separate Java virtual machines for each a session established by a server. Because each session has its own virtual machine, the Java programs executed by the server for each user connected to the server are insulated from the Java programs executed by the server for all other users connected to the server. The separate VM instances can be created and run, for example, in separate units of execution that are managed by the operating system of the platform on which the server is executing. For example, the separate VM instances may be executed either as separate processes, or using separate system threads. Because the units of execution used to run the separate VM instances are provided by the operating system, the operating system is able to ensure that the appropriate degree of insulation exists between the VM instances." *Sexton*, col. 5, ll.29-44. "As mentioned above, in the conventional Java server model, each session initiated between a client and the server is handled by a single VM thread within a multi-threaded VM instance. In such an implementation, the Java virtual machine itself takes the form of a set of global variables accessible to all threads, where there is only one copy of each global variable. Unlike the conventional Java server, in one embodiment of the invention, an entire Java VM instance is spawned for every session made through the server. According to one implementation, each Java VM instance is spawned by instantiating a VM data structure in session memory. During execution, the state of a VM instance is modified by performing transformations on the VM data structure associated with the VM instance, and/or modifying the data contained therein. Specifically, the VM data structure that is instantiated for a particular session is passed as an input parameter to the server routines that are called during that session. Rather than accessing global variables that are shared among VM threads associated with different sessions, the routines access session-specific variables that are

U.S. Patent No. 7,426,720	*Sexton* in view of *Bugnion*
	stored within the VM data structure that is passed to them. Consequently, the contention for resources that otherwise occurs between threads associated with different sessions is significantly reduced, because those threads are associated with different VM instances." *Sexton*, col. 7, l.61 – col. 8, l.18.
and means for cloning the memory space as a child runtime system process responsive to a process request and means for executing the child runtime system process;	*Sexton* provided an apparatus with a means for cloning the memory space as a child runtime system process responsive to a process request and means for executing the child runtime system process. *Sexton* provided a process cloning mechanism to instantiate the child runtime system process by copying the memory space of the master runtime system process into a separate memory space for the child runtime system process by employing a "shared state area," constructed based on the master runtime process and housed in a separate memory space, i.e., "not duplicated in the session memory for each VM instance." "Techniques are provided for instantiating separate Java virtual machines for each a session established by a server. Because each session has its own virtual machine, the Java programs executed by the server for each user connected to the server are insulated from the Java programs executed by the server for all other users connected to the server. The separate VM instances can be created and run, for example, in separate units of execution that are managed by the operating system of the platform on which the server is executing. For example, the separate VM instances may be executed either as separate processes, or using separate system threads. Because the units of execution used to run the separate VM instances are provided by the operating system, the operating system is able to ensure that the appropriate degree of insulation exists between the VM instances." *Sexton*, col. 5, ll.29-44. "As mentioned above, in the conventional Java server model, each session initiated between a client and the server is handled by a single VM thread within a multi-threaded VM instance. In such an implementation, the Java virtual machine itself takes the form of a set of global variables accessible to all threads, where there is only one copy of each global variable. Unlike the conventional Java server, in one embodiment of the invention, an entire Java VM instance is spawned for every session made through the server. According to one implementation, each Java VM instance is spawned by instantiating a VM data structure in

U.S. Patent No. 7,426,720	*Sexton* in view of *Bugnion*
	session memory. During execution, the state of a VM instance is modified by performing transformations on the VM data structure associated with the VM instance, and/or modifying the data contained therein. Specifically, the VM data structure that is instantiated for a particular session is passed as an input parameter to the server routines that are called during that session. Rather than accessing global variables that are shared among VM threads associated with different sessions, the routines access session-specific variables that are stored within the VM data structure that is passed to them. Consequently, the contention for resources that otherwise occurs between threads associated with different sessions is significantly reduced, because those threads are associated with different VM instances." *Sexton*, col. 7, l.61 – col. 8, l.18.
wherein the means for cloning the memory space is configured to clone the memory space of a child runtime system process using a copy-on-write process cloning mechanism that instantiates the child runtime system process by copying references to the memory space of the master runtime system process into a separate memory space for the child runtime system process and that defers copying of the memory space of the master runtime system process until the child runtime system process needs to modify the referenced memory space of the master runtime system process.	*Sexton* in view of *Bugnion* provided an apparatus wherein the means for cloning the memory space is configured to clone the memory space of a child runtime system process using a copy-on-write process cloning mechanism that instantiates the child runtime system process by copying references to the memory space of the master runtime system process into a separate memory space for the child runtime system process and that defers copying of the memory space of the master runtime system process until the child runtime system process needs to modify the referenced memory space of the master runtime system process. *Sexton* disclosed methods for a plurality of VMs to access certain "shared state" data such that "[t]he non-session-specific data for the class, including the methods, method table and fields, are not duplicated in the session memory for each VM instance." Given the goal of reducing session memory by sharing data between multiple Virtual Machines, one of ordinary skill in the art at the time of the invention could take the teachings of *Sexton* in combination with the *Bugnion* prior art and be in possession of the invention. Here, given the goal of the reduction of overhead of *Sexton*, it would be obvious to one of ordinary skill in the art to combine *Sexton* with the well-known copy on write technology, thereby placing the artisan in possession of the invention. And *Bugnion* discloses that "[t]he virtual subnet and networking interfaces of Disco also use copy-on-write mappings to reduce copying and to allow for memory sharing." *Bugnion*, col. 15, l.66 - col. 16, l.1. "In addition, techniques are provided for reducing startup costs and incremental memory requirements of the Java virtual machines instantiated by the server. For example, the use of

U.S. Patent No. 7,426,720	***Sexton* in view of *Bugnion***
	a shared state area allows the various VM instantiations to share class definitions and other resources. In addition, while it is actively processing a call, each VM instance has two components, a session-duration component and a call-duration component. Only the data that must persist in the VM between calls is stored in the session-duration component. Data that need not persist between calls is stored in the call-duration component, which is instantiated at the start of a call, and discarded at the termination of the call." *Sexton*, col. 5, ll.53-65. "According to one embodiment, the overhead associated with each VM instance is reduced by sharing certain data with other VM instances. The memory structure that contains the shared data is referred to herein as the shared state area. Each VM instance has read-only access to the data that has been loaded into the shared state area, and therefore the VM instances do not contend with each other for access rights to that data. According to one embodiment, the shared state area is used to store loaded Java classes. The shared Java classes may include static variables whose values are session-specific. Therefore, according to one embodiment, a data structure, referred to herein as a "java_active_class", is instantiated in session space to store session-specific values (e.g. static variables) of a corresponding shared Java class. The non-session-specific data for the class, including the methods, method table and fields, are not duplicated in the session memory for each VM instance. Rather, all VM instances share read-only access to a single instantiation of the class, thus significantly reducing the memory requirements of VM instances (the per-session memory requirements). According to one embodiment, the java_active_class for each shared class further includes a pointer to the shared class to allow VM instances more efficient access to the shared class data." *Sexton*, col. 8, ll.40-64. "The VMM layer also maintains copy-on-write disks that allow virtual machines to transparently share main memory resources and disk storage resources, and performs dynamic page migration/replication that hides distributed characteristics of the physical memory resources from the operating systems. The VMM layer may also comprise a virtual memory resource interface to allow processes running on multiple virtual machines to share memory." *Bugnion*, col. 6, ll.29-36.

U.S. Patent No. 7,426,720	***Sexton* in view of *Bugnion***
	"The interposition on all DMA requests offers an opportunity for Disco to share disk and memory resources among virtual machines. Disco's copy-on-write disks allow virtual machines to share both main memory and disk storage resources. Disco's virtual network devices allow virtual machines to communicate efficiently. The combination of these two mechanisms, detailed below, allows Disco to support a system-wide cache of disk blocks in memory that can be transparently shared between all the virtual machines." *Bugnion*, col. 14, ll.55-64. "Disco intercepts every disk request that DMAs data into memory. When a virtual machine requests to read a disk block that is already in main memory, Disco can process the request without going to disk. Furthermore, if the disk request is a multiple of the machine's page size, Disco can process the DMA request by simply mapping the page into the virtual machine's physical memory. In order to preserve the semantics of a DMA operation, Disco maps the page read-only into the destination address page of the DMA. Attempts to modify a shared page will result in a copy-on-write fault handled internally by the monitor. Using this mechanism, multiple virtual machines accessing a shared disk end up sharing machine memory, The copy-on-write semantics means that the virtual machine is unaware of the sharing with the exception that disk requests can finish nearly instantly. Consider an environment running multiple virtual machines for scalability purposes. All the virtual machines can share the same root disk containing the kernel and application programs. The code and other read-only data stored on the disk will be DMA-ed into memory by the first virtual machine that accesses it. Subsequent requests will simply map the page specified to the DMA engine with out transferring any data. The result is shown in FIG. 4 where all virtual machines share these read-only pages. Effectively we get the memory sharing patterns expected of a single shared memory multiprocessor operating system even though the system runs multiple independent operating systems. To preserve the isolation of the virtual machines, disk writes must be kept private to the virtual machine that issues them. Disco logs the modified sectors so that the copy-on-write disk is never actually modified. For persistent disks, these modified sectors would be logged in a separate disk partition managed by Disco. To simplify our implementation, we only applied the concept of copy-on-write disks to non-persistent disks and kept the modified sectors in main memory whenever possible." *Bugnion*, col. 14, l.66 - col. 15, l.35.

U.S. Patent No. 7,426,720	***Sexton* in view of *Bugnion***
	"The virtual subnet and networking interfaces of Disco also use copy-on-write mappings to reduce copying and to allow for memory sharing." *Bugnion*, col. 15, l.66 - col. 16, l.1.
21. A system according to claim 1, further comprising: a resource controller to set operating system level resource management parameters on the child runtime system process.	*Sexton* in view of *Bugnion* provided an apparatus with a resource controller to set operating system level resource management parameters on the child runtime system process. This limitation is inherent in the *Sexton* disclosure of "units of execution that are managed by the operating system." "The separate VM instances can be created and run, for example, in separate units of execution that are managed by the operating system of the platform on which the server is executing. *Sexton*, col. 5, ll. 34-38. This limitation is also disclosed by *Bugnion*: "The unique virtual machine monitor of the present invention virtualizes all the resources of the machine, exporting a more conventional hardware interface to the operating system. The monitor manages all the resources so that multiple virtual machines can coexist on the same multiprocessor. The virtual machine monitor allows multiple copies of potentially different operating systems to coexist on the multiprocessor. Some virtual machines can run commodity uniprocessor or multiprocessor operating systems, and others can run specialized operating systems fine-tuned for specific workloads. The virtual machine monitor schedules the virtual resources (processor and memory) or the virtual machines on the physical resources of the scalable multiprocessor." *Bugnion*, col. 4, ll. 25-38. "Although the system looks like a cluster of loosely-coupled machines, the virtual machine monitor uses global policies to manage all the resources of the machine, allowing workloads to exploit the fine-grain resource sharing potential of the hardware. For example, the monitor can move memory between virtual machines to keep applications from paging to disk when free memory is available in the machine. Similarly, the monitor dynamically schedules virtual processors on the physical processors to balance the load across the machine. The use of commodity software leverages the significant engineering effort invested in these operating systems and allows CC-NUMA machines to support their large application base.

U.S. Patent No. 7,426,720	***Sexton* in view of *Bugnion***
	Since the monitor is a relatively simple piece of code compared to large operating systems, this can be done with a small implementation effort as well as with a low risk of introducing software bugs and incompatibilities." *Bugnion*, col. 4, ll. 51-67. "In one aspect of the invention, a computational system is provided that comprises a multiprocessor hardware layer, a virtual machine monitor layer, and a plurality of operating systems. The multiprocessor hardware layer comprises a plurality of computer processors, a plurality of physical resources associated with the processors, and an interconnect providing mutual communication between the processors and resources. The virtual machine monitor (VMM) layer executes directly on the hardware layer and comprises a resource manager that manages the physical resources of the multiprocessor, a processor manager that manages the computer processors, and a hardware emulator that creates and manages a plurality of virtual machines. The operating systems execute on the plurality of virtual machines and transparently share the plurality of computer processors and physical resources through the VMM layer. In a preferred embodiment, the VMM layer further comprises a virtual network device providing communication between the operating systems executing on the virtual machines, and allowing for transparent sharing optimizations between a sender operating system and a receiver operating system. In addition, the resource manager maintains a global buffer cache that is transparently shared among the virtual machines using read-only mappings in portions of an address space of the virtual machines. The VMM layer also maintains copy-on-write disks that allow virtual machines to transparently share main memory resources and disk storage resources, and performs dynamic page migration/replication that hides distributed characteristics of the physical memory resources from the operating systems. The VMM layer may also comprise a virtual memory resource interface to allow processes running on multiple virtual machines to share memory." *Bugnion*, col. 6, ll. 6-36.
22. A method according to claim 10, further comprising: setting operating system level resource management parameters on the child runtime system process.	*Sexton* in view of *Bugnion* provided an apparatus with a setting operating system level resource management parameters on the child runtime system process. The *Sexton* disclosure focuses on separate VM instances rather than a parent and child runtime instance, but the resource control is still present, as controlled by the operating system of the platform. It is inherent that if a resource control management system is in place there must be resource

U.S. Patent No. 7,426,720	***Sexton* in view of *Bugnion***
	management parameters in place to guide the operating system. "The separate VM instances can be created and run, for example, in separate units of execution that are managed by the operating system of the platform on which the server is executing. *Sexton*, col. 5, ll. 34-38. "The unique virtual machine monitor of the present invention virtualizes all the resources of the machine, exporting a more conventional hardware interface to the operating system. The monitor manages all the resources so that multiple virtual machines can coexist on the same multiprocessor. The virtual machine monitor allows multiple copies of potentially different operating systems to coexist on the multiprocessor. Some virtual machines can run commodity uniprocessor or multiprocessor operating systems, and others can run specialized operating systems fine-tuned for specific workloads. The virtual machine monitor schedules the virtual resources (processor and memory) or the virtual machines on the physical resources of the scalable multiprocessor." *Bugnion*, col. 4, ll. 25-38. "Although the system looks like a cluster of loosely-coupled machines, the virtual machine monitor uses global policies to manage all the resources of the machine, allowing workloads to exploit the fine-grain resource sharing potential of the hardware. For example, the monitor can move memory between virtual machines to keep applications from paging to disk when free memory is available in the machine. Similarly, the monitor dynamically schedules virtual processors on the physical processors to balance the load across the machine. The use of commodity software leverages the significant engineering effort invested in these operating systems and allows CC-NUMA machines to support their large application base. Since the monitor is a relatively simple piece of code compared to large operating systems, this can be done with a small implementation effort as well as with a low risk of introducing software bugs and incompatibilities." *Bugnion*, col. 4, ll. 51-67. "In one aspect of the invention, a computational system is provided that comprises a multiprocessor hardware layer, a virtual machine monitor layer, and a plurality of operating systems. The multiprocessor hardware layer comprises a plurality of computer processors, a plurality of physical resources associated with the processors, and an interconnect providing

U.S. Patent No. 7,426,720	***Sexton* in view of *Bugnion***
	mutual communication between the processors and resources. The virtual machine monitor (VMM) layer executes directly on the hardware layer and comprises a resource manager that manages the physical resources of the multiprocessor, a processor manager that manages the computer processors, and a hardware emulator that creates and manages a plurality of virtual machines. The operating systems execute on the plurality of virtual machines and transparently share the plurality of computer processors and physical resources through the VMM layer. In a preferred embodiment, the VMM layer further comprises a virtual network device providing communication between the operating systems executing on the virtual machines, and allowing for transparent sharing optimizations between a sender operating system and a receiver operating system. In addition, the resource manager maintains a global buffer cache that is transparently shared among the virtual machines using read-only mappings in portions of an address space of the virtual machines. The VMM layer also maintains copy-on-write disks that allow virtual machines to transparently share main memory resources and disk storage resources, and performs dynamic page migration/replication that hides distributed characteristics of the physical memory resources from the operating systems. The VMM layer may also comprise a virtual memory resource interface to allow processes running on multiple virtual machines to share memory." *Bugnion*, col. 6, ll. 6-36.