Title: COPY-ON-WRITE UPDATE-TRIGGERED CONSISTENCY

Abstract: According to an example, copy-on-write (COW) update-triggered consistency may include determining a first shared version related to a shared set of data related to a process involving a first application and a process involving a second application. First and second local versions may be respectively assigned to the shared set of data associated with the first and second applications. Initiation of a commit process by the first or second applications may be determined respectively based on a write to the first or second local versions respectively related to the shared set of data associated with the first or second applications. In response to initiation of the commit process, a second shared version may be generated related to the shared set of data based on the respective processes involving the first and the second applications.
COPY-ON-WRITE UPDATE-TRIGGERED CONSISTENCY

BACKGROUND

[0001] Transactional memory (TM) typically attempts to simplify concurrent programming by allowing a set of load and store instructions to execute in an atomic manner. Software transactional memory (STM) is a concurrency control process for controlling access to shared memory in concurrent computing. For STM, a transaction occurs when a piece of code executes a series of reads and writes to shared memory. These reads and writes logically occur at a single instant in time, with intermediate states being invisible to other transactions.
BRIEF DESCRIPTION OF DRAWINGS

[0002] Features of the present disclosure are illustrated by way of example and not limited in the following figure(s), in which like numerals indicate like elements, in which:

[0003] Figure 1 illustrates an environment including a copy-on-write (COW) update-triggered consistency apparatus, according to an example of the present disclosure;

[0004] Figure 2 illustrates a memory bank for the COW update-triggered consistency apparatus, according to an example of the present disclosure;

[0005] Figure 3 illustrates a flowchart for implementing a commit process for the COW update-triggered consistency apparatus, according to an example of the present disclosure;

[0006] Figure 4 illustrates a flowchart for implementing a write page fault process for the COW update-triggered consistency apparatus, according to an example of the present disclosure;

[0007] Figure 5 illustrates a flowchart for implementing an update process for the COW update-triggered consistency apparatus, according to an example of the present disclosure;

[0008] Figure 6 illustrates a flowchart for a method for COW update-triggered consistency, according to an example of the present disclosure;

[0009] Figure 7 illustrates further details of the flowchart for the method for COW update-triggered consistency, according to an example of the present disclosure; and

[0010] Figure 8 illustrates a computer system for implementing the COW update-triggered consistency apparatus, according to an example of the present disclosure.
DETAILED DESCRIPTION

[0011] For simplicity and illustrative purposes, the present disclosure is described by referring mainly to examples. In the following description, numerous specific details are set forth in order to provide a thorough understanding of the present disclosure. It will be readily apparent however, that the present disclosure may be practiced without limitation to these specific details. In other instances, some methods and structures have not been described in detail so as not to unnecessarily obscure the present disclosure.

[0012] Throughout the present disclosure, the terms "a" and "an" are intended to denote at least one of a particular element. As used herein, the term "includes" means includes but not limited to, the term "including" means including but not limited to. The term "based on" means based at least in part on.

[0013] TM allows parallelism, and may be used to detect and resolve conflicts without resorting to the complexities of fine grained locking or non-blocking synchronization. Further, STM allows programs to take advantage of disjoint access parallelism on both the read-side and write-side. However, STM based systems can have a higher overhead compared to systems based on other synchronization techniques. To reduce some of the challenges associated with STM, data may be accessed non-transactionally. For example, privatization may be used to access data non-transactionally. However, the non-transactional accesses may negate the isolation guarantees of a transactional memory system.

[0014] According to examples, a copy-on-write (COW) update-triggered consistency apparatus and a method for COW update-triggered consistency are disclosed herein. Generally, the COW process may be based on the principle that when multiple separate tasks use identical copies of the same information (i.e., data stored in a computer memory bank), separate copies of that information are not needed for each process. Instead of the separate copies, each process may be provided with a pointer to the same resource. The apparatus and method disclosed herein may provide for COW to be applied to a shared memory for
version control. Each application that is to access data stored in a shared central repository may be provided with an isolated local copy (i.e., a local version) of a shared memory segment (i.e., shared data from the shared central repository). A process associated an application may be free to read/write to the isolated local copy of its shared memory segment. Changes may be maintained locally until the process associated with the application uses a commit process to commit the changed results to the shared central repository. An update process may be invoked in order for the process associated with the application to update its view of its shared memory segment. Similarly, the update process may be invoked in order for other processes associated with other applications to update their respective views of their shared memory segments.

[0015] According to examples, the apparatus and method disclosed herein may be implemented as part of a virtual memory manager of an operating system. With respect to the commit process disclosed herein, a call to commit from any of the processes sharing a memory segment may make all the isolated local copies visible to the apparatus and method disclosed herein, which, in turn, may make all the isolated local copies available to other processes. A most recent version in the memory bank may be located by maintaining a page table (e.g., a hash table). For the page table, the key may be designated as the creation date of a page, and the value may be designated as the actual content of the memory segment. A lookup table (i.e., a translation lookaside buffer (TLB)) may be flushed after a commit operation is successfully completed. Updates in local copies may be update/commit driven.

[0016] Figure 1 illustrates an environment 100 including a COW update-triggered consistency apparatus 102, according to an example of the present disclosure. Referring to Figure 1, the environment 100 of the COW update-triggered consistency apparatus 102 is depicted as including an operating system 104 that may include a collection of software (i.e., machine readable instructions) to manage computer hardware resources and to provide common services for computer programs. The OS 104 may provide access for applications 106 to a
memory bank 108. The applications 106 may include applications (1) to (N). The memory bank 108 may include a shared central repository 110 to store a shared set of data associated with the applications 106. The memory bank 108 may further include segments of memory assigned to process applications 112 to store a local version and a page table associated with the shared set of data, as described in further detail with reference to Figure 2. The COW update-triggered consistency apparatus 102 may include a commit module 114 to implement a commit process from the segments of memory assigned to process applications 112 to the shared central repository 110, as described in further detail with reference to Figure 3. A write page fault module 116 may implement a write process associated with one of the applications 106, as described in further detail with reference to Figure 4. An update module 118 may implement an update process associated with one of the applications 106, as described in further detail with reference to Figure 5.

[0017] As described herein, the modules and other elements of the apparatus 102 may be machine readable instructions stored on a non-transitory computer readable medium. In addition, or alternatively, the modules and other elements of the apparatus 102 may be hardware or a combination of machine readable instructions and hardware.

[0018] Referring to Figures 1 and 2, Figure 2 illustrates a particular example of the memory bank 108, according to an example of the present disclosure. The COW update-triggered consistency apparatus 102 may maintain memory-mapped segments as a local version 200 (i.e., a private region) to a process involving one of the applications 106. In this manner, each process involving one of the applications 106 may maintain the local version 200 of a shared memory segment (i.e., shared data from the shared central repository 110) of the main memory bank 108. The local version 200 may generally include a page and frame based designation (e.g., {(page(1),frame(1)), ..., (page(j),frame(j)))}. Further, page(1) may be designated as P1, page(2) may be designated as P2, etc. Similarly, frame(1) may be designated as F1, frame(2) may be designated as F2, etc. A page may
represent a logical page in the memory bank 108. A frame may represent a physical frame number where the most recent version of that logical page is stored. [0019] With respect to the local version 200 for the particular example of Figure 2, assuming that one of the applications 106 (e.g., App(1)) that is associated with the segment of memory assigned to process application 112 obtains an original version 202 (e.g., V1) from the shared central repository 110, and another one of the applications 106 (e.g., App(2)) also modifies V1, when the App(1) commits its local version 200 to the shared central repository 110, the App(1) has modified V1 by deleting (P3,F3), and (P4,F4), and adding (P5,F5). Assume that the App(2) has also modified V1 to retain (P3,F3) and (P4,F4), and committed its local version 200 to the shared central repository 110. Thus, for the App(2), the local version 200 would include (P2,F2), (P3,F3), and (P4,F4). Further, for the App(1), the local version 200 would include (P2,F2) and (P5,F5). When the local version 200 for the App(1) is committed to the shared central repository 110, a new version 204 (e.g., V2) would therefore further include (P5,F5) appended as shown in Figure 2. [0020] The local version 200 may be based on a duplicate mapping of the same physical memory frames (e.g., from the shared central repository 110) referenced by a page table 206. The page table 206 may generally indicate a location of the appropriate page and frame in the shared central repository 110. For the page table 206, a key may be designated as the creation date, and a value (e.g., P2, P5, etc., for the example of Figure 2) may be designated as the actual content of the memory segment (e.g., the segment of memory assigned to process applications 112). Shared frames between processes (e.g., any of the frames in any of the versions in the shared central repository 110), or between processes and the page table 206, may be flagged as read-only in the table entries for the page table 206. For the example of Figure 2, frame F2 may be flagged as read-only in the table entries for the page table 206. If another process associated with another one of the applications 106 tries to write, a new frame may be allocated with the content of the original frame. A writer may write to this new frame rather than to the initial frame. A new pair (page, frame#) may added to the local set of changes. Further,
a reference to the original page may be stored locally for merging, for example, in the page table 206. For the example of Figure 2 for App(1), a new frame F5 may be allocated with the content of the original frame F2.

According to a further example, assume V1 in the shared central repository 110 includes (P6,F6) that is a read-only frame. If an application attempts to write to (P6,F6), a new frame F7 may be allocated with the content P6 of the original frame F6. A new pair (P7,F7) may added to the local set of changes, where P7 may represent the new content originally associated with the content P6.

When a process involving one of the applications 106 attempts to write, the COW update-triggered consistency apparatus 102 may execute a COW page fault, as disclosed herein with respect to a write page fault process 400 described in further detail with reference to Figure 4. A snapshot process may be used by the process involving the particular application 106 to create a new segment (e.g., (P7,F7) in the previous example may be allocated to V2). Any write to this new segment may remain private, using COW memory protection, until the commit module 114 issues a commit to place the local modification (e.g., deletion of (P3.F3) and (P4.F4), and addition of (P5.F5) for App(1) for the example of Figure 2) to the local version 200 in the central version (e.g., V2 for the example of Figure 2) of the shared central repository 110, making these changes visible to other processes. For the example of Figure 2, committing local changes to the shared central repository 110 may prompt the addition of the new version 204 (i.e., the version V2) to the shared central repository 110. A commit process 300 is described in further detail with reference to Figure 3. Changes may be tracked at page level as shown for the example of Figure 2. Alternatively or additionally, changes may be tracked at a different granularity level (e.g., byte level, etc.). The update module 118 may issue an update to pull data from the shared central repository 110 into the process-exclusive location (e.g., the segment of memory assigned to process applications 112 for the particular application 106). An update process 500 is described in further detail with reference to Figure 5. For page disjoint modifications, the update module 118 may modify the page table 206 of the
calling process to the latest snapshot (e.g., V2 for the example of Figure 2) of the shared central repository 110.

[0023] Referring to Figures 1 and 3, Figure 3 illustrates a flowchart of the commit process 300 for the COW update-triggered consistency apparatus 102, according to an example of the present disclosure.

[0024] With respect to the commit process 300 that is implemented by the commit module 114, generally, committing local changes to the shared central repository 110 may prompt the addition of the new version 204 to the shared central repository 110. If needed, a new pair (page, frame#) may be added to the new version 204. For the example of Figure 2, a new pair (P5, F5) may be added to the version V2 kept in the shared central repository 110.

[0025] For the commit process 300, at block 302, the commit module 114 may further determine whether there is a previous reference to the offset pair (e.g., (P2, F2) and (P5, F5) from the local version 200 for the App(1) for the example of Figure 2). For example, at block 302, the commit module 114 may determine whether a previous version (i.e., the version in the shared central repository 110) of a given pair (page, frame#) existed for the page from the given pair in the shared central repository 110. Thus, for the example of Figure 2, the offset pairs (P2, F2) and (P5, F5) from the local version 200 for the App(1) may be separately compared to V1 and V2 in the shared central repository 110 at each invocation of block 302.

[0026] At block 304, if the determination at block 302 is yes, the reference in the previous version may be removed. For the example of Figure 2, the reference in the previous version 202 (e.g., V1) in the shared central repository 110 may be removed for the offset pair (P2, F2).

[0027] At block 306, if the determination at block 302 is no or following block 304, the page and frame may be added to the central version. For the example of Figure 2, for the App(1), following block 304, the page and frame (P2, F2) may be added to the central version (e.g., V2). Similarly, for the App(1), following block 302, the page and frame (P5, F5) may be added to the central version (e.g., V2).

[0028] At block 308, the entry for a page and frame (e.g., (P2,F2) and (P5,F5)
for App(1) for the example of Figure 2) may be write-protected in the page table 206 of the calling process. In other words, the entry for the page table 206 corresponding to a page and frame may be marked as protected, forcing a COW page fault on the next write attempt to this page.

[0029] With respect to the commit process 300, the local version 200 may represent a duplicate mapping of the same physical memory frames referenced by the page table 206. Any frames shared between processes may be marked as read-only in their respective page entries.

[0030] The shared central repository 110 may be subject to extensive contention between processes. The COW update-triggered consistency apparatus 102 may use a single read-write lock to protect the shared central repository 110 from race conditions. Race conditions may generally represent behavior of a system where the output is dependent on a sequence or timing of other uncontrollable events.

[0031] With respect to scalability for the shared central repository 110, it may not be thread safe to allow an update to traverse the page table 206 while a commit may be removing old entries for the very same pages being committed (e.g., P2 for the App(1) for the example of Figure 2). The commit process 300 may therefore mark old entries as obsolete, and deletion deferred whereby a new field is added to the pair (e.g., (P2,F2) for the App(1) for the example of Figure 2) for the new version (e.g., V2 for the example of Figure 2), indicating the version (e.g., V1 for the example of Figure 2) that made old entries obsolete. An update, as described in further detail with reference to update process 500 of Figure 5, may now check this new field. If the entry is obsolete, then nothing may be done on that entry. Removal of obsolete entries may be performed, for example, as a background kernel space process.

[0032] With respect to concurrency, for the COW update-triggered consistency apparatus 102, concurrency may be implemented by providing committers (e.g., different ones of the applications 106 that initiate a commit process) access to different pages in a concurrent manner. Pages with a single committer (e.g., a
single application 106 that initiates a commit process) may be committed concurrently. Since a thread acquires a new version on calling commit, pages with more than one committer may let the committing proceed in version order. A version map (not shown) may store the arrival order so that committers themselves can write when it is safe to do so (e.g., they are first on the version map).

[0033] Referring to Figures 1 and 4, Figure 4 illustrates a flowchart of the write page fault process 400 for the COW update-triggered consistency apparatus 102, according to an example of the present disclosure.

[0034] With respect to the write page fault process 400 that is implemented by the write page fault module 116, at block 402, a new page and frame may be allocated. For the example of Figure 2, for the App(1), a new page and frame (P5,F5) may be allocated.

[0035] At block 404, the new page and frame may be initialized with contents from the read-only frame. For the example of Figure 2, for the App(1), the new page and frame (P5,F5) may be initialized with contents from the read-only frame (e.g., based on the write-protection applied by block 308 for the commit process 300).

[0036] At block 406, the new and page frame may be written to. For the example of Figure 2, for the App(1), the new and page frame (P5,F5) may be written to.

[0037] At block 408, the new and page frame may be added to the local version. For the example of Figure 2, for the App(1), the new and page frame (P5,F5) may be added to the local version 200.

[0038] At block 410, the COW update-triggered consistency apparatus 102 may retain reference to the original page and frame in the page table 206. For the example of Figure 2, for the App(1), the COW update-triggered consistency apparatus 102 may retain reference to the original page and frame (P2,F2) in the page table 206.

[0039] Referring to Figures 1 and 5, Figure 5 illustrates a flowchart of an update process 500 for the COW update-triggered consistency apparatus 102, according
to an example of the present disclosure.

[0040] With respect to the update process 500 that is implemented by the update module 118, when a process needs an updated view of the shared memory bank 108, and thus the shared central repository 110, the caller's view (e.g., the local version 200 for the example of Figure 2) of the segment may be updated to reflect the most current version. The update process 500 may modify the caller's page table (e.g., page table 206 for the example of Figure 2) to point to the most recent version (e.g., V2 for the example of Figure 2) of each page. When updating, the central array (e.g., V2 for the example of Figure 2) may provide a set of changes \{(page1,frame1),...,\(page\),\(frame\)} that occurred on the central array since the last time that process updated and the current time.

[0041] For the update process 500, at block 502, the update module 118 may determine if there are any local changes or central changes. For the example of Figure 2, the update module 118 may determine if there are any local changes to the local version 200 or central changes (e.g., to V1) in the shared central repository 110.

[0042] If the determination at block 502 is no, at block 504, the update module 118 may obtain a set of changes from the central version. For the example of Figure 2, the update module 118 may obtain a set of changes between V1 and V2.

[0043] For each entry in the set of changes, at block 506, the update module 118 may modify the entries in the page table. For the example of Figure 2, the update module 118 may modify the entries in the page table 206.

[0044] For each entry in the set of changes, at block 508, the update module 118 may flush the TLB (not shown) for the address at the given page number (page).

[0045] With respect to the update process 500, according to an example, the local version 200 of the process may not change (e.g., the central array V2 may be changed by receiving commits from other processes). Alternatively, if the local copy per the local version 200 and the central version (e.g., V1 for the example of Figure 2) in the shared central repository 110 are both changed between calls to
update, a conflict may arise. A reference to the original version (e.g., a local copy of V1 in example of Figure 2) of each page may be kept in the process exclusive space (e.g., segment of memory assigned to process application 112 for the example of Figure 2).

[0046] The merging of the conflicting local and central versions may then proceed such that if the local version differs from the local copy of the original, the local version is used. For example, if the determination at block 502 is yes, at block 510, the update module 118 may determine if the current local version differs from the local copy of the original. For the example of Figure 2, the update module 118 may determine if the current local version 200 differs from the local copy of the original (e.g., V1).

[0047] If the determination at block 510 is yes, at block 512, the update module 118 may use the local version. For the example of Figure 2, the update module 118 may use the local version 200.

[0048] If the determination at block 510 is no, at block 504, the version from the set of changes (e.g., from the shared central repository 110) may be used. In other words, at block 504, the update module 118 may obtain a set of changes from the central version (e.g., difference between V1 and V2 for the example of Figure 2).

[0049] The update module 118 may allow users to indicate a modification unit (i.e., granularity level) for any given segment, and merge any disjoint set of changes of the given unit size. If the sets are not disjoint, the local version (e.g., local version 200 for the example of Figure 2) may be preferred. Further, the update module 118 may also use mutual exclusion techniques to avoid concurrent writes. The result of the update may be a failed status that triggers a retry of the local computation.

[0050] Figures 6 and 7 respectively illustrate flowcharts of methods 600 and 700 for COW update-triggered consistency, corresponding to the example of the COW update-triggered consistency apparatus 102 whose construction is described in detail above. The methods 600 and 700 may be implemented on the COW update-triggered consistency apparatus 102 with reference to Figures 1-5 by way
of example and not limitation. The methods 600 and 700 may be practiced in other apparatus.

[0051] Referring to Figure 6, for the method 600, at block 602, the method may include determining a first shared version related to a shared set of data related to a process involving a first application and a process involving a second application. For example, referring to Figures 1 and 2, the method may include determining a first shared version (e.g., V1) related to a shared set of data related to a process involving a first application (e.g., App(1)) and a process involving a second application (e.g., App(2)).

[0052] At block 604, the method may include assigning first and second local versions respectively to the shared set of data associated with the first and second applications. For example, referring to Figures 1 and 2, the method may include assigning first and second local versions (e.g., local versions 200 for App(1) and App(2)) respectively to the shared set of data associated with the first and second applications (e.g., App(1) and App(2)).

[0053] At block 606, the method may include determining initiation of a commit process by the first or second applications respectively related to the shared set of data associated with the first or second applications. For example, referring to Figures 1 and 2, the method may include determining initiation of a commit process (e.g., the commit process 300) by the first or second applications (e.g., App(1) or App(2)) respectively (e.g., based on a write to the local versions 200 for App(1) or App(2)) related to the shared set of data associated with the first or second applications.

[0054] At block 608, in response to initiation of the commit process, the method may include generating a second shared version related to the shared set of data based on the respective processes involving the first and the second applications. For example, referring to Figures 1 and 2, in response to initiation of the commit process, the method may include generating a second shared version (e.g., V2) related to the shared set of data based on the respective processes involving the first and the second applications (e.g., App(1) and App(2)).
According to an example, for the method 600, the commit process may further includes determining (e.g., at block 302 of Figure 3) if there is a previous reference to an offset pair including a page and frame associated with a respective one of the first and second local versions in the shared central repository of the memory bank. In response to a determination that there is a previous reference to the offset pair, the method 600 may include removing (e.g., at block 304 of Figure 3) a reference to the offset pair in the first shared version from the shared central repository of the memory bank. Further, in response to a determination that there is no previous reference to the offset pair, the method 600 may further include adding (e.g., at block 306 of Figure 3) the offset pair to the second shared version in the shared central repository of the memory bank, and marking (e.g., at block 308 of Figure 3) an entry for the offset pair as write-protected in a page table.

According to an example, the method 600 may further include determining the write associated with the process involving the first application or the process involving the second application, allocating (e.g., at block 402 of Figure 4) a new page and frame in the local version of the first application or the second application associated with the write, initializing (e.g., at block 404 of Figure 4) the new page and frame with contents of a read-only frame, entering write contents (e.g., at block 406 of Figure 4) to the new page and frame, and adding (e.g., at block 408 of Figure 4) the new page and frame to the local version of the first application or the second application associated with the write.

According to an example, the method 600 may further include determining an update associated with the process involving the first application or the process involving the second application, and determining (e.g., at block 502 of Figure 5) if there are changes to the first or second local versions, or if there are changes to the shared set of data resident in the shared central repository of the memory bank. In response to a determination that there are changes to the first or second local versions, or that there are changes to the shared set of data resident in the shared central repository of the memory bank, the method 600 may further include determining (e.g., at block 510 of Figure 5) if a current local version differs
from a local copy of the shared set of data resident in the shared central repository of the memory bank. In response to a determination that the current local version differs from a local copy of the shared set of data resident in the shared central repository of the memory bank, the method 600 may further include using (e.g., at block 512 of Figure 5) the current local version as an updated local version. In response to a determination that the current local version does not differ from a local copy of the shared set of data resident in the shared central repository of the memory bank, the method 600 may further include obtaining (e.g., at block 504 of Figure 5) a set of changes from a central version resident in the shared central repository of the memory bank.

[0058] According to an example, the method 600 may further include determining an update associated with the process involving the first application or the process involving the second application, and determining (e.g., at block 502 of Figure 5) if there are changes to the first or second local versions, or if there are changes to the shared set of data resident in the shared central repository of the memory bank. In response to a determination that there are no changes to the first or second local versions, or that there are no changes to the shared set of data resident in the shared central repository of the memory bank, the method 600 may further include obtaining (e.g., at block 504 of Figure 5) a set of changes from a central version resident in the shared central repository of the memory bank.

[0059] Referring to Figure 7, for the method 700, at block 702, the method may include determining a first shared version related to a shared set of data related to a process involving a first application and a process involving a second application.

[0060] At block 704, the method may include assigning first and second local versions respectively to the shared set of data associated with the first and second applications.

[0061] At block 706, the method may include determining initiation of a commit process by the first or second applications respectively based on a write to the first or second local versions respectively related to the shared set of data associated with the first or second applications.
[0062] At block 708, in response to initiation of the commit process, the method may include generating a second shared version related to the shared set of data based on the respective processes involving the first and the second applications.

[0063] At block 710, the method may include assigning first and second segments of the memory bank respectively to the first and second local versions respectively associated with the first and second applications. For example, referring to Figures 1 and 2, the method may include assigning first and second segments (e.g., see Figure 2, segments for App(1) and App(2)) of the memory bank 108 respectively to the first and second local versions respectively associated with the first and second applications.

[0064] At block 712, the method may include generating a page table to indicate a location of the page and the frame in the shared central repository of the memory bank. For example, referring to Figures 1 and 2, the method may include generating a page table 206 to indicate a location of the page and the frame in the shared central repository 110 of the memory bank 108.

[0065] Figure 8 shows a computer system 800 that may be used with the examples described herein. The computer system may represent a generic platform that includes components that may be in a server or another computer system. The computer system 800 may be used as a platform for the apparatus 102. The computer system 800 may execute, by a processor (i.e., a single or multiple processors) or other hardware processing circuit, the methods, functions and other processes described herein. These methods, functions and other processes may be embodied as machine readable instructions stored on a computer readable medium, which may be non-transitory, such as hardware storage devices (e.g., RAM (random access memory), ROM (read only memory), EPROM (erasable, programmable ROM), EEPROM (electrically erasable, programmable ROM), hard drives, and flash memory).

[0066] The computer system 800 may include a processor 802 that may implement or execute machine readable instructions performing some or all of the methods, functions and other processes described herein. Commands and data
from the processor 802 may be communicated over a communication bus 804. The computer system may also include a main memory 806, such as a random access memory (RAM), where the machine readable instructions and data for the processor 802 may reside during runtime, and a secondary data storage 808, which may be non-volatile and stores machine readable instructions and data. The memory and data storage are examples of computer readable mediums. The memory 806 may include a COW update-triggered consistency module 820 including machine readable instructions residing in the memory 806 during runtime and executed by the processor 802. The COW update-triggered consistency module 820 may include the modules of the apparatus 102 shown in Figure 1.

[0067] The computer system 800 may include an I/O device 810, such as a keyboard, a mouse, a display, etc. The computer system may include a network interface 812 for connecting to a network. Other known electronic components may be added or substituted in the computer system.

[0068] What has been described and illustrated herein is an example along with some of its variations. The terms, descriptions and figures used herein are set forth by way of illustration only and are not meant as limitations. Many variations are possible within the spirit and scope of the subject matter, which is intended to be defined by the following claims -- and their equivalents -- in which all terms are meant in their broadest reasonable sense unless otherwise indicated.
What is claimed is:

1. A non-transitory computer readable medium having stored thereon machine readable instructions to provide copy-on-write (COW) update-triggered consistency, the machine readable instructions, when executed, cause at least one processor to:

   determine a first shared version related to a shared set of data related to a process involving a first application and a process involving a second application, wherein the shared set of data is resident in a shared central repository of a memory bank;

   assign first and second local versions respectively to the shared set of data associated with the first and second applications;

   determine initiation of a commit process by the first or second applications respectively based on a write to the first or second local versions respectively related to the shared set of data associated with the first or second applications; and

   in response to initiation of the commit process, generate a second shared version related to the shared set of data based on the respective processes involving the first and the second applications.

2. The non-transitory computer readable medium of claim 1, the machine readable instructions, when executed, further cause the at least one processor to:

   assign first and second segments of the memory bank respectively to the first and second local versions respectively associated with the first and second applications.

3. The non-transitory computer readable medium of claim 1, wherein the first and second local versions each include a page that represents a logical page in the memory bank, and a frame that represents a physical frame number where a most recent version of the logical page is stored in the memory bank.
4. The non-transitory computer readable medium of claim 3, the machine readable instructions, when executed, further cause the at least one processor to:
   generate a page table to indicate a location of the page and the frame in the shared central repository of the memory bank.

5. The non-transitory computer readable medium of claim 1, wherein the processes involving the first and the second applications further include a read associated with the first or second local versions.

6. The non-transitory computer readable medium of claim 1, wherein the commit process further comprises the machine readable instructions to cause the at least one processor to:
   determine if there is a previous reference to an offset pair including a page and frame associated with a respective one of the first and second local versions in the shared central repository of the memory bank;
   in response to a determination that there is a previous reference to the offset pair, remove a reference to the offset pair in the first shared version from the shared central repository of the memory bank;
   in response to a determination that there is no previous reference to the offset pair, add the offset pair to the second shared version in the shared central repository of the memory bank; and
   mark an entry for the offset pair as write-protected in a page table.

7. The non-transitory computer readable medium of claim 1, the machine readable instructions, when executed, further cause the at least one processor to:
   determine the write associated with the process involving the first application or the process involving the second application;
   allocate a new page and frame in the local version of the first application or the second application associated with the write;
initialize the new page and frame with contents of a read-only frame;
enter write contents to the new page and frame; and
add the new page and frame to the local version of the first application or the
second application associated with the write.

8. The non-transitory computer readable medium of claim 1, the machine
readable instructions, when executed, further cause the at least one processor to:
determine an update associated with the process involving the first application
or the process involving the second application;
determine if there are changes to the first or second local versions, or if there
are changes to the shared set of data resident in the shared central repository of
the memory bank;
in response to a determination that there are changes to the first or second
local versions, or that there are changes to the shared set of data resident in the
shared central repository of the memory bank, determine if a current local version
differs from a local copy of the shared set of data resident in the shared central
repository of the memory bank;
in response to a determination that the current local version differs from a
local copy of the shared set of data resident in the shared central repository of the
memory bank, use the current local version as an updated local version; and
in response to a determination that the current local version does not differ
from a local copy of the shared set of data resident in the shared central repository
of the memory bank, obtain a set of changes from a central version resident in the
shared central repository of the memory bank.

9. The non-transitory computer readable medium of claim 1, the machine
readable instructions, when executed, further cause the at least one processor to:
determine an update associated with the process involving the first application
or the process involving the second application;
determine if there are changes to the first or second local versions, or if there
are changes to the shared set of data resident in the shared central repository of the memory bank; and

in response to a determination that there are no changes to the first or second local versions, or that there are no changes to the shared set of data resident in the shared central repository of the memory bank, obtain a set of changes from a central version resident in the shared central repository of the memory bank.

10. A copy-on-write (COW) update-triggered consistency apparatus comprising:

   at least one processor; and

   a memory storing machine readable instructions that when executed by the at least one processor cause the at least one processor to:

   determine a first shared version related to a shared set of data related to a process involving a first application and a process involving a second application, wherein the shared set of data is resident in a shared central repository of a memory bank of the memory;

   assign first and second local versions respectively to the shared set of data associated with the first and second applications;

   assign first and second segments of the memory bank respectively to the first and second local versions respectively associated with the first and second applications;

   determine initiation of a commit process by the first or second applications respectively based on a write to the first or second local versions respectively related to the shared set of data associated with the first or second applications; and

   in response to initiation of the commit process, generate a second shared version related to the shared set of data based on the respective processes involving the first and the second applications.

11. The COW update-triggered consistency apparatus according to claim 10, wherein the first and second local versions each include a page that represents a
logical page in the memory bank, and a frame that represents a physical frame number where a most recent version of the logical page is stored in the memory bank.

12. The COW update-triggered consistency apparatus according to claim 10, wherein the commit process further comprises the machine readable instructions to cause the at least one processor to:

- determine if there is a previous reference to an offset pair including a page and frame associated with a respective one of the first and second local versions in the shared central repository of the memory bank;
- in response to a determination that there is a previous reference to the offset pair, remove a reference to the offset pair in the first shared version from the shared central repository of the memory bank; and
- in response to a determination that there is no previous reference to the offset pair, add the offset pair to the second shared version in the shared central repository of the memory bank.

13. A method for copy-on-write (COW) update-triggered consistency, the method comprising:

- determining a first shared version related to a shared set of data related to a process involving a first application and a process involving a second application;
- assigning first and second local versions respectively to the shared set of data associated with the first and second applications;
- determining initiation of a commit process by the first or second applications respectively related to the shared set of data associated with the first or second applications; and
- in response to initiation of the commit process, generating, by at least one processor, a second shared version related to the shared set of data based on the respective processes involving the first and the second applications.
14. The method of claim 13, further comprising:
   determining an update associated with the process involving the first application or the process involving the second application;
   determining if there are changes to the first or second local versions, or if there are changes to the shared set of data resident in the shared central repository of the memory bank;
   in response to a determination that there are changes to the first or second local versions, or that there are changes to the shared set of data resident in the shared central repository of the memory bank, determining if a current local version differs from a local copy of the shared set of data resident in the shared central repository of the memory bank;
   in response to a determination that the current local version differs from a local copy of the shared set of data resident in the shared central repository of the memory bank, using the current local version as an updated local version; and
   in response to a determination that the current local version does not differ from a local copy of the shared set of data resident in the shared central repository of the memory bank, obtaining a set of changes from a central version resident in the shared central repository of the memory bank.

15. The method of claim 13, further comprising:
   determining an update associated with the process involving the first application or the process involving the second application;
   determining if there are changes to the first or second local versions, or if there are changes to the shared set of data resident in the shared central repository of the memory bank; and
   in response to a determination that there are no changes to the first or second local versions, or that there are no changes to the shared set of data resident in the shared central repository of the memory bank, obtaining a set of changes from a central version resident in the shared central repository of the memory bank.
**FIG. 3**

COMPAT PROCESS

IS THERE A PREVIOUS REFERENCE TO THE OFFSET PAIR?

NO

ADD PAGE AND FRAME TO CENTRAL VERSION

YES

REMOVE REFERENCE IN PREVIOUS VERSION

ENTRY FOR PAGE AND FRAME IS WRITE-PROTECTED IN THE PAGE TABLE OF THE CALLING PROCESS
WRITE PAGE FAULT
PROCESS 400

ALLOCATE NEW PAGE AND FRAME 402

INITIALIZE THE NEW AND PAGE FRAME WITH CONTENTS OF THE READ-ONLY FRAME 404

WRITER WRITES TO THE NEW PAGE AND FRAME 406

ADD THE NEW PAGE AND FRAME TO LOCAL VERSION 408

RETAIN REFERENCE TO ORIGINAL PAGE AND FRAME 410

FIG. 4
FIG. 5

UPDATE PROCESS

ARE THERE LOCAL CHANGES OR CENTRAL CHANGES?

No

Obtain set of changes from central version

Yes

Does the current local version differ from the local copy of the original?

No

Modify entries in page table

Yes

Use local version

Flush TLB for old page offset
600

Determine a first shared version related to a shared set of data related to a process involving a first application and a process involving a second application

602

Assign first and second local versions respectively to the shared set of data associated with the first and second applications

604

Determine initiation of a commit process by the first or second applications respectively related to the shared set of data associated with the first or second applications

606

In response to initiation of the commit process, generate a second shared version related to the shared set of data based on the respective processes involving the first and the second applications

608

FIG. 6
700

DETERMINE A FIRST SHARED VERSION RELATED TO A SHARED SET OF DATA RELATED TO A PROCESS INVOLVING A FIRST APPLICATION AND A PROCESS INVOLVING A SECOND APPLICATION

702

ASSIGN FIRST AND SECOND LOCAL VERSIONS RESPECTIVELY TO THE SHARED SET OF DATA ASSOCIATED WITH THE FIRST AND SECOND APPLICATIONS

704

DETERMINE INITIATION OF A COMMIT PROCESS BY THE FIRST OR SECOND APPLICATIONS RESPECTIVELY BASED ON A WRITE TO THE FIRST OR SECOND LOCAL VERSIONS RESPECTIVELY RELATED TO THE SHARED SET OF DATA ASSOCIATED WITH THE FIRST OR SECOND APPLICATIONS

706

IN RESPONSE TO INITIATION OF THE COMMIT PROCESS, GENERATE A SECOND SHARED VERSION RELATED TO THE SHARED SET OF DATA BASED ON THE RESPECTIVE PROCESSES INVOLVING THE FIRST AND THE SECOND APPLICATIONS

708

ASSIGN FIRST AND SECOND SEGMENTS OF THE MEMORY BANK RESPECTIVELY TO THE FIRST AND SECOND LOCAL VERSIONS RESPECTIVELY ASSOCIATED WITH THE FIRST AND SECOND APPLICATIONS

710

GENERATE A PAGE TABLE TO INDICATE A LOCATION OF THE PAGE AND THE FRAME IN THE SHARED CENTRAL REPOSITORY OF THE MEMORY BANK

712

FIG. 7
**A. CLASSIFICATION OF SUBJECT MATTER**

G06F 9/46(2006.01)i, G06F 9/38(2006.01)i

According to International Patent Classification (IPC) or to both national classification and IPC

**B. FIELDS SEARCHED**

Minimum documentation searched (classification system followed by classification symbols)

G06F 9/46; G06F 12/00; G06F 12/14; G06F 9/455; G06F 9/38

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Korean utility models and applications for utility models

Japanese utility models and applications for utility models

Electronic database consulted during the international search (name of database and, where practicable, search terms used)

eKOMPASS(KIPO internal) & Keywords: copy-on-write, version, shared, application

**C. DOCUMENTS CONSIDERED TO BE RELEVANT**

<table>
<thead>
<tr>
<th>Category</th>
<th>Citation of document, with indication, where appropriate, of the relevant passages</th>
<th>Relevant to claim No.</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>US 2003-0200398 AL (KENNETH, M. HARRIS) 23 October 2003 See abst ract ; c laims 1-7 ; and figures 1, 2.</td>
<td>1-15</td>
</tr>
<tr>
<td>A</td>
<td>US 2005-0262313 AL (JOHN HOLT) 24 November 2005 See abst ract ; c laims 1-19 ; and figures 5-8.</td>
<td>1-15</td>
</tr>
<tr>
<td>A</td>
<td>US 2011-0307668 AL (FBREGAS ANTONI FERR et al.) 15 December 2011 See abst ract ; c laims 1-5 ; and figure 1.</td>
<td>1-15</td>
</tr>
<tr>
<td>A</td>
<td>US 2008-0034364 AL (MONICA LAM et al.) 07 February 2008 See abst ract ; c laims 1, 2, 21-27 ; and figures 1A-3.</td>
<td>1-15</td>
</tr>
</tbody>
</table>

Further documents are listed in the continuation of Box C.

See patent family annex.

* Special categories of cited documents:
  "A" document defining the general state of the art which is not considered to be of particular relevance
  "E" earlier application or patent but published on or after the international filing date
  "L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)
  "O" document referring to an oral disclosure, use, exhibition or other means
  "P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art

"&" document member of the same patent family

Date of the actual completion of the international search: 30 May 2014 (30.05.2014)

Date of mailing of the international search report: 30 May 2014 (30.05.2014)

Name and mailing address of the ISA/KR

International Application Division
Korean Intellectual Property Office
189 Cheongna-ro, Seo-gu, Daejeon Metropolitan City, 302-701, Republic of Korea

Facsimile No. +82-42-472-7140

Authorized officer

LEE, Seok Hyung

Telephone No. +82-42-481-5983

Form PCT/ISA/210 (second sheet) (July 2009)
<table>
<thead>
<tr>
<th>Patent document cited in search report</th>
<th>Publication date</th>
<th>Patent family member(s)</th>
<th>Publication date</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>AU 2005-236086 Al</td>
<td>03/11/2005</td>
</tr>
<tr>
<td></td>
<td></td>
<td>AU 2005-236086 B2</td>
<td>04/01/2007</td>
</tr>
<tr>
<td></td>
<td></td>
<td>AU 2005-236087 Al</td>
<td>03/11/2005</td>
</tr>
<tr>
<td></td>
<td></td>
<td>AU 2005-236087 B2</td>
<td>01/09/2011</td>
</tr>
<tr>
<td></td>
<td></td>
<td>AU 2005-236088 Al</td>
<td>03/11/2005</td>
</tr>
<tr>
<td></td>
<td></td>
<td>AU 2005-236088 B2</td>
<td>04/01/2007</td>
</tr>
<tr>
<td></td>
<td></td>
<td>AU 2005-236089 Al</td>
<td>03/11/2005</td>
</tr>
<tr>
<td></td>
<td></td>
<td>AU 2006-238334 Al</td>
<td>26/10/2006</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CA 2563900 Al</td>
<td>03/11/2005</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CN 101908001 A</td>
<td>08/12/2010</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CN 1965308 A</td>
<td>16/05/2007</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CN 1965308 B</td>
<td>04/08/2010</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CN 1965308 co</td>
<td>16/05/2007</td>
</tr>
<tr>
<td></td>
<td></td>
<td>EP 1763771 Al</td>
<td>21/03/2007</td>
</tr>
<tr>
<td></td>
<td></td>
<td>EP 1763772 Al</td>
<td>21/03/2007</td>
</tr>
<tr>
<td></td>
<td></td>
<td>EP 1763773 Al</td>
<td>21/03/2007</td>
</tr>
<tr>
<td></td>
<td></td>
<td>EP 1763774 Al</td>
<td>21/03/2007</td>
</tr>
<tr>
<td></td>
<td></td>
<td>EP 1763774 B1</td>
<td>05/12/2012</td>
</tr>
<tr>
<td></td>
<td></td>
<td>EP 1880303 Al</td>
<td>23/01/2008</td>
</tr>
<tr>
<td></td>
<td></td>
<td>EP 2341430 Al</td>
<td>06/07/2011</td>
</tr>
<tr>
<td></td>
<td></td>
<td>KR 10-1209023 B1</td>
<td>10/12/2012</td>
</tr>
<tr>
<td></td>
<td></td>
<td>us 2005-0240737 Al</td>
<td>27/10/2005</td>
</tr>
<tr>
<td></td>
<td></td>
<td>us 2005-0257219 Al</td>
<td>17/11/2005</td>
</tr>
<tr>
<td></td>
<td></td>
<td>us 2005-0262513 Al</td>
<td>24/11/2005</td>
</tr>
<tr>
<td></td>
<td></td>
<td>us 2006-0020913 Al</td>
<td>26/01/2006</td>
</tr>
<tr>
<td></td>
<td></td>
<td>us 2006-0095483 Al</td>
<td>04/05/2006</td>
</tr>
<tr>
<td></td>
<td></td>
<td>us 2006-0242464 Al</td>
<td>26/10/2006</td>
</tr>
<tr>
<td></td>
<td></td>
<td>us 2009-0055603 Al</td>
<td>26/02/2009</td>
</tr>
<tr>
<td></td>
<td></td>
<td>us 2009-0198776 Al</td>
<td>06/08/2009</td>
</tr>
<tr>
<td></td>
<td></td>
<td>us 2009-0235033 Al</td>
<td>17/09/2009</td>
</tr>
<tr>
<td></td>
<td></td>
<td>us 2009-0235034 Al</td>
<td>17/09/2009</td>
</tr>
<tr>
<td></td>
<td></td>
<td>us 2010-0262590 Al</td>
<td>14/10/2010</td>
</tr>
<tr>
<td></td>
<td></td>
<td>us 7707179 B2</td>
<td>27/04/2010</td>
</tr>
<tr>
<td></td>
<td></td>
<td>us 7788314 B2</td>
<td>31/08/2010</td>
</tr>
<tr>
<td></td>
<td></td>
<td>us 7818296 B2</td>
<td>19/10/2010</td>
</tr>
<tr>
<td></td>
<td></td>
<td>us 7844665 B2</td>
<td>30/11/2010</td>
</tr>
<tr>
<td>Patent document cited in search report</td>
<td>Publication date</td>
<td>Patent family member(s)</td>
<td>Publication date</td>
</tr>
<tr>
<td>----------------------------------------</td>
<td>-----------------</td>
<td>-------------------------</td>
<td>-----------------</td>
</tr>
<tr>
<td>US 2011-0307668 AL</td>
<td>15/12/2011</td>
<td>CN 102279807 A</td>
<td>14/12/2011</td>
</tr>
<tr>
<td>US 2008-0034364 AL</td>
<td>07/02/2008</td>
<td>DE102011075776 Al</td>
<td>15/12/2011</td>
</tr>
</tbody>
</table>