

also been replaced by intelligent graphical user interfaces (GUI) capable of self processing the received and transmitted data from/to servers.

In modern data processing systems, a client/server architecture largely in use and capable of supporting a large number of remotely located clients is the so-called 3-tier architecture. An example of such architecture is illustrated in Figure 1. The data tier 100 is traditionally built around a master database system 120, possibly a large or very large repository of all the data necessary to the daily operation of any business organization, company or enterprise in order to conduct all sorts of commercial and administrative operations. Database is mostly of the relational type, i.e., is under the control of a relational database management system or RDBMS. It is typically administrated through one or more master servers 112 by administrators of the data processing system from GUI's 140. Administrators are generally the sole users of the system authorized to update directly database contents.

The intermediate or middle tier of the exemplary 3-tier system of Figure 1 is the application tier 200 from where all the specific software applications 240 of the organization, owner of the data processing system, are run. This collection of specific applications, often globally referred to as the middleware software, is the proprietary software of the organization. It is used to serve all organization's remote clients from its repository of data 120 through the master servers 110. Remote clients form the third tier 300 of the 3-tier architecture. Queries from client tier 300 are thus processed and responded to by the specific applications of the intermediate tier 200 on data fetched from the data tier 100.

In a 3-tier architecture, when a larger number of remote clients need to be served, scalability of the system to maintain global performances is obtained by adding independent processing nodes in the middle tier so as to increase the overall processing power of the data processing system. Hence, the application tier 200 is generally comprised of several independent processing nodes that are referred to, in the following description, as slave nodes 210. Then, a common practice to prevent data tier 100 from being overwhelmed by too many data requests from an increasing number of slave nodes, is to have the applicative processes 240 working on pieces of data brought from the master database and stored in each application node as long as necessary. In the exemplary system of Figure 1 this takes the form of cache files 250 on which the applicative processes 240 can work without having to incur long delays to get them from the master database through the master servers each time they are needed. In such a data processing system processing power and software applications are thus distributed, i.e., replicated, on as many nodes 210 as necessary to reach the

– A second issue occurs during replication if, for any reason, process fails completing normally. Since active and inactive data blocks share a same data area, writing into the inactive part of the data area can also possibly corrupt the active part of the memory-mapped file. If the replication process fails writing the full list of data blocks the corresponding linked list is indeed corrupted. Unpredictable results must then be expected like addressing blocks of the active part thus breaking the data area division between active and inactive parts.

The two above issues are critical for the client applications impacted because of the inevitable service interruptions they trigger when occurring. In order to be able to notify the impacted clients and to prevent data corruption from further propagating, a standard practice of the replication process consists in locking the memory-mapped file before writing the data. The lock is released at the end of the replication unless replication does not end normally. Even though the lock mechanism prevents data corruption from further propagating and provides to clients the possibility of viewing the corrupted files, it does not however help to recover in an automated manner.

The current replication process thus suffers from a lack of resiliency and requires manual actions because, as discussed above:

- size of memory-mapped files is a static parameter that needs to be set manually and which leads to waste much memory resources if over sized for the application;
- flip-flop mechanism between files within the shared data area does not prevent corruption of linked lists from happening;
- and, lock mechanism is not recovered in an automated manner.

It is therefore a general object of the invention to bring a solution to at least some of the above drawbacks and limitations of the current replication mechanism of cache files into a shared-memory of a middleware processing node from a master database. It is a particular object of the invention to obtain that replication of cache files be unconditionally a corruption-free process for existing cache files even though a replication operation may occasionally fails or does not complete as expected.

It is another object of the invention to provide a new cache structure where data versions can be controlled in an easier way.

Further objects, features and advantages of the present invention will become apparent to the ones skilled in the art upon examination of the following description in reference to the accompanying drawings. It is intended that any additional advantages be incorporated herein.

SUMMARY

The invention relates to a cache structure comprising a data area with a plurality of cache files each storing one version of a set of data; a control area comprising at least one release area, said release area comprising at least one version area with plural version sections each storing at least one key addressing data of one single cache file, as many offsets as version sections, each offset pointing to one version section.

An intermediary structure is thus formed separate from the data area.

as
~~is~~ In one aspect of the invention the exemplary embodiments provide a control area ~~is~~ a file separate from the data area so that a better protection of the control portion versus the data portion is provided.

In another aspect of some embodiments, three offsets and three version sections are provided which enables pointing steps differing from the conventional flip flop mechanism. The use of three offsets ~~renders~~ ^{renders} the shift between versions more secure and ~~preserve~~ ^{preserves} the availability to data by applicative processes even in case of crash during an update stage. In addition, one of the three offsets may be used to maintain an access – even transitory - to the replaced version section; thus applicative processes which started working with the previously current version section can continue operating even if a new version section has been introduced and is effective (i.e. can be accessed as current version section). The shift to a new version section always starts by resetting the memory of that version section. Since the access to the key location value through version section is not atomic (like find a map key); two fast successful consecutive version shifts, using a basic flip flop mechanism, would conflict with the key read. The transitory access, provided by the use of three offsets, copes with this.

The invention also relates to a method for controlling access to data stored in a cache memory during update stages comprising the steps of:

- using a cache memory structure of the invention;
- pointing to each version section with one distinct offset;
- defining one offset as current offset pointing to one version section defined as current version section and addressing versions of the set of data defined as current data.

BRIEF DESCRIPTION OF THE DRAWINGS

The present invention will now be described in details with reference to the drawings for the purpose of illustrating preferred embodiments.

FIGURE 1 illustrates a 3-tier computing architecture of the prior art in which the invention is carried out.

FIGURE 2 illustrates a cache file structure of the prior art.

FIGURE 3 shows how the invention is implemented with the help of a version control file in the middle tier of a 3-tier computing architecture.

FIGURE 4 shows a basic structure of the version control file.

5 FIGURES 5a to 5i illustrate the steps of the method to add in an atomic way a new data snapshot in the shared-memory.

FIGURE 6 shows a first enhancement brought to the basic structure of the version control file to optimize updating process and to handle concurrent writes.

10 FIGURES 7a to 7d describe the addition of a new key in the enhanced structure of the enhanced structure of the version control file.

FIGURE 8 discusses a second enhancement brought to the version structure that allows memory size of release areas to be increased when necessary.

DETAILED DESCRIPTION

A few definitions of terms used in the following description are given below:

15 - *data area* here means a portion of cache memory where data are stored and are subject to writing and reading operations when accessed by processes. In one preferred embodiment, reading operations are made by applicative processes whereas writing operations are administrator's tasks. When data are updated/replaced/created by an administrator computer device, the data are made available to the applicative
20 processes for reading purpose. In one preferred embodiment, the data are organized in sets of data, each representing the definition of an object. By way of example, a list of rules or fare definitions or flight schedule data of an airline computerized system can be used as objects. An object can also consist in the content of a single data field. The objects defined by sets of data are typically subject to modification usually reflecting
25 modifications made in a master database by an administrator which means that the sets of data need to be updated or replaced. This also involves that the invention has to deal with plural versions of an object so that plural versions of a set of data coexist for the same object. The objects can also be grouped by package. By way of example, different lists of rules handling flight schedules can be grouped in a flight schedule
30 package. However the package can still be ^{viewed}~~view~~ as a single object, i.e. set of data. The term snapshot is also used in the description as corresponding to one version of set of data defining one object, or given versions of set of objects. One cache file is advantageously but not limitatively dedicated to each snapshot or version of a set of data. Typically the data of the data area are for access by at least one applicative
35 process - which here means any computer process run by at least one processor of at least one computer device- to which these data are helpful. Applicative processes may

comprise read processes of a search engine such as a fare search engine used in global distribution systems of the travel industry. Conventional hardware means can be implemented to support the cache of the invention. The data access mode can rely on shared-memory current techniques and on memory-mapped file concept which enables
5 a correspondence between the physical storage of data and the applicative process view of these data.

- *control area* here means a portion of the cache used to control the access by processes to the data of the data area presented above. The control area is preferably contained within one or several dedicated files of the cache, segregated from the data
10 area. Said file is hereafter called control file or version control file. It will be later described that the control area may co act with other control means in embodiments where an entry point area is implemented. And a plurality of control areas, advantageously each in a separate file, can be used.

- *area* (as used for the control area, the data area or the entry point area) refers
15 broadly to a portion of the cache memory which may not be a continuous segment of said memory.

- *keys* represent small size data specifying the address of stored data. The value of a key provides the location of the corresponding stored data. Keys ^{do}~~does~~ not necessarily address sets of data of the same size: a key can be used to access a specific field in a set of data describing a ^{bigger}~~smaller~~ object. A different key can be used to
20 access the area where the set of data of the object are, or the area where the objects are. Another key can be used to access a previous snapshot of the object. A set of keys is used to access different snapshots of same object. Each set of data in the data area is uniquely identified by a key. Keys are preferably kept small, but can have any
25 arbitrary size. The sole constraint is that each set of data must be associated to a unique fixed key.

The purpose of the version control area is to materialize each key into a way to access to the data set, by example by linking it to a memory address. Therefore a key might be considered as equivalent to an address or access link for the comprehension of the
30 invention.

Before describing in detail preferred embodiments of the invention in connection with the figures, some optional features of the invention are introduced hereafter; they can be used alternatively or cumulatively wherein:

- The version section contains a structure (like map) allowing adding or
35 removing keys.

- Snapshots of data can be grouped by domain. The same control structure is applied at domain level.
- the control area comprises now a main release area with main version sections, each of said main version sections comprising, for each domain, domain definition data including an access link to a dedicated release area, said domain release area
- the main release area comprises as many main offsets as main version sections, and, for each domain release area as many domain offsets as domain version sections.
- It is consisting in a shared memory;
- The control area can also be managed using release area concept. The entry point control area is comprising now:
 - a plurality of control areas;
 - an entry point release area comprising entry version sections, each defining an access link to one separate control area.
 - the entry point control area comprises as many entry offsets as entry version sections, each entry offset pointing to one entry version section.

According to preferred embodiments the method may include at least one of the following steps:

- defining one offset as new offset pointing to one version section defined as new version section, differing from the current version section;
- clearing the new version section;
- storing a new version of at least one set of data in a cache file of the data area;
- storing new version information in the new version section;
- letting the current offset point to the new version section.
- providing the release area with three offsets and with three version ~~section~~^{sections};
- pointing to each version section with one distinct offset;
- defining a first offset as current offset pointing to one version section defined as current version section and addressing versions of the set of data defined as current data;
- defining a second offset as new offset pointing to one version section defined as new version section, differing from the current version section;
- defining a third offset as old offset pointing to one version section defined as old version section;

3. Data from the current control area are copied into the new one. Optionally, if it is necessary to retrieve the old values, the whole content of the cache values are copied; otherwise, only the current values are copied.

4. The key is added in the new control area 730.

5 5. The current entry offset 719 is switched to entry section (717) pointing to the created control area 730.

6. Deprecated versions of data are erased to save resources.

The entry point area 710 uses the structure ^{described}~~describes~~ in Figures 4 and 5 so that to allow current control area to be switched without disturbing read accesses.

10 Because the entry point needs only to contain a structure of fixed size, it never requires new allocations and can serve as a fixed entry point.

The policy for computing the new control area capacity may be freely adapted and should result ⁱⁿ~~of~~ a compromise between the number of times this procedure needs to be applied and the overall memory space.

15 Read accesses first read the entry point area to retrieve the current control area. The latter is reached at the main release area the current main offset of which specifies the current main version section. The current main version section provides the key link to reach the right domain release area. At this level, the domain release area's current offset points to the domain version section addressing the data area 420 to be
20 accessed.

Although illustrative embodiments of the present invention have been described in detail with reference to the accompanying drawings, it is to be understood that the invention is not limited to those precise embodiments and that changes and modifications may be effected therein by those in the art without departing from the
25 scope and spirit of the invention.

CLAIMS

- 1- Cache memory structure comprising:
- a data area (410) with a plurality of cache files (250), each of the cache files (250) storing one version of a same set of data;
 - a control area configured to indicate which version of the set of data is defined as current data and comprising at least one release area (430), said release area (430, 432, 434) comprising:
 - o at least one version area (450) with plural version sections (452, 454, 456) each storing at least one key addressing data of one single cache file (250),
 - o as many offsets (442) as version sections (452, 454, 456), each offset (442) pointing to one version section (452, 454, 456).
- 2- Cache memory structure according to the preceding claim wherein the control area consists in at least one file (220) separate from the data area (410).
- 3- Cache memory structure according to any of the preceding claims comprising three offsets (442) and three version sections (452, 454, 456).
- 4- Cache memory structure according to any of the preceding claims wherein:
- each version section (452, 454, 456) stores values associated to one single key to access one cache file (250) storing one version of a set of data assigned to said key.
 - the sets of data are distributed in a plurality of domains;
 - the control area comprises a main release area (436) with main version sections (620, 630, 640), each of said main version sections (620, 630, 640) comprising, for each domain an access link (610) to a release area (432, 434) associated to said domain.
- 5- Cache memory structure according to the preceding claim wherein the control area comprises as many main offsets (605) as main version sections (620, 630, 640), each main offset (605) pointing to one main version section (620, 630, 640) of main release area (436).
- 6- Cache memory structure according to the preceding claim, the control area comprising three main offsets (605), three main version sections (620, 630, 640), and, for each domain release area (432, 434) three domain offsets and three domain version sections.
- 7- Cache memory structure according to any of the preceding claims consisting in a shared memory (230).

8- Cache memory structure according to any of the preceding claims comprising:

- a plurality of control areas (720, 730);
- an entry point area (710) comprising entry sections (715, 716, 717) each defining an access link to one separate control area (720, 730).

5 9- Cache memory structure according to the preceding claim wherein the control area comprises as many entry offsets (719) as entry sections (715, 716, 717), each entry offset (719) pointing to one entry section (715, 716, 717).

10- Method for controlling access to data stored in a cache memory during update stages using a cache memory structure of any of the preceding claims.

10 11- Method according to the preceding claim comprising the steps of :

- pointing to each version section (452, 454, 456) with one distinct offset (442);
- defining one offset (442) as current offset pointing to one version section (454) defined as current version section and addressing at least one version of the set of data defined as current data;
- enabling access by an applicative process to the data of a cache file (250) addressed by the version section pointed to by the current offset.

12- Method according to the preceding claim comprising the step of

- defining one offset (442) as new offset pointing to one version section (456) defined as new version section, differing from the current version section;
- clearing the new version section;
- storing a new version of at least one set of data in a cache file (250) of the data area;
- storing new data version definition information in the new version section;
- letting the current offset (442) point to the new version section.

13- Method according to the preceding claim comprising the further steps of:

- providing the release area with three offsets (442) and with three version section;
- pointing to each version section (452, 454, 456) with one distinct offset (442);
- defining a first offset (442) as current offset pointing to one version section (454) defined as current version section and addressing at least one version of the at least one set of data defined as current data;
- defining a second offset (442) as new offset pointing to one version section (456) defined as new version section, differing from the current version section;

- defining a third offset (442) as old offset pointing to one version section (452) defined as old version section;
 - clearing the new version section;
 - storing a new version of at least one set of data in a cache file (250) of the data area;
 - storing new data version definition information in the new version section;
 - letting the old offset point to the current version section;
 - letting the current offset (442) point to the new version section;
 - letting the new offset (442) point to the old version section.
- 10 14- Method according to the preceding claim comprising the step, after letting the old offset point to the current version section, of enabling access by an applicative process to the data of a cache file (250) addressed by the version section pointed to by the old offset.
- 15 15- Method according to the preceding claim wherein the step of enabling the access is of a finite time.
- 20 16- Method according to any of claims 10 to 15 comprising:
- using a cache memory structure according to claim 9;
 - defining one entry offset as current offset pointing to one entry section (715, 716, 717) defined as current entry section;
 - enabling an access by an applicative process to the control area addressed by the current offset.
- 17- Computer program product stored in a computer-readable memory medium and comprising instructions for executing the method of any of claims 10 to 16.

CLAIMS

1. An apparatus comprising:

a memory; and

5 a cache memory structure stored in the memory, the cache memory structure comprising:

a data area with a plurality of cache files, wherein each cache file is configured to store a version of a set of data; and

10 a control area configured to control access to the plurality of cache files, wherein the control area comprises at least one release area that comprises:

at least one version area comprising:

a first version section, a second version section, and a third version section that are each configured to store at least one key that addresses data of at least one cache file, and

15 a version offset area comprising:

a new offset configured to point to one of the first version section, the second version section, and the third version section to thereby identify the pointed-to version section as a new version section,

20 a current offset configured to point to one of the first version section, second version section, and third version section to thereby identify the pointed-to version section as a current version section, and

an old offset configured to point to one of the first version section, second version section, and third version section to thereby identify the pointed-to version section as an old version section.

25 2. The apparatus of claim 1, wherein the control area comprises at least one file separate from the data area.

30 3. The apparatus of claim 1, wherein each version section stores values associated to one single key to access one cache file storing one version of a set of data assigned to said key, a plurality of sets of data are distributed in a plurality of domains, the at least one release area is at least one domain release area, and the control area further comprises:

a main release area that includes main version sections that each store an access link corresponding to each domain release area; and

35 a domain offset area including a new main offset configured to point to a first main version section to thereby identify the first main version section as a new main version

section, a current main offset configured to point to a second main version section to thereby identify the second main version section as a current main version section, and an old main offset configured to point to a third main version section to thereby identify the third main version section as an old main version section.

5

4. The apparatus of claim 3, wherein each set of data of the plurality of sets of data is distributed in a particular domain corresponding to a particular airline, and each set of data corresponds to business rules of the particular airline.

10 5. The apparatus of claim 1, wherein the cache memory structure consists of a shared memory.

6. The apparatus of claim 1 further comprising:

a plurality of control areas; and

15 an entry point area comprising entry sections, wherein each entry section is configured to store an access link to one separate control area.

7. The apparatus of claim 6, wherein the entry point area comprises as many entry offsets as entry sections, each entry offset pointing to one entry section.

20

8. The apparatus of claim 1, wherein the set of data stored in each cache file corresponds to a flight schedule package for at least one airline, and each flight schedule package includes fare definitions and flight schedule data.

25 9. A method for controlling access to data stored in a cache memory during update stages comprising:

using a cache memory structure comprising:

a data area with a plurality of cache files, wherein each cache file is configured to store a different version of a set of data;

30 a control area configured to control access to the plurality of cache files, wherein the control area comprises at least one release area that comprises:

at least one version area comprising:

a first version section, a second version section, and a third version section that are each configured to store at least one key that addresses data of at least one cache file, and

35

a version offset area comprising:

a new offset configured to point to one of the first version section, the second version section, and the third version section to thereby identify the pointed-to version section as a new version section,

a current offset configured to point to one of the first version section, second version section, and third version section to thereby identify the pointed-to version section as a current version section, and

an old offset configured to point to one of the first version section, second version section, and third version section to thereby identify the pointed-to version section as an old version section, and

enabling access by an applicative process to the data of a cache file addressed by the version section pointed to by the current offset.

10. The method of claim 9 further comprising:

clearing the new version section;

storing a new version of at least one set of data in a cache file of the data area addressed by the key of the new version section;

storing new data version definition information in the new version section; and

modifying the current offset to point to the new version section.

11. The method of claim 10 further comprising:

modifying the old offset to point to the current version section; and

modifying the new offset to point to the old version section.

12. The method of claim 11 further comprising:

after modifying the old offset to point to the current version section, enabling access by an applicative process to the data of a cache file addressed by the version section pointed to by the old offset.

13. The method of claim 12 wherein enabling the access is of a finite time.

14. The method of claim 9 further comprising:

defining an entry offset as current offset pointing to one entry section defined as current entry section;

enabling an access by an applicative process to the control area addressed by the current entry offset.

15. A computer program product stored in a non-transitory computer-readable medium and comprising instructions for executing the method of claim 9.

16. A method for controlling access to data stored in a cache memory structure

5 comprising a data area with a plurality of cache files and a control area configured to control access to the plurality of cache files, the method comprising:

storing a new version of a particular set of data in a cache file among the plurality of cache files, wherein each cache file is configured to store a version of a set of data;

in response to storing the new version of the particular set of data in the cache file,

10 modifying a key stored in a first version section of the control area to correspond to an address of the new version of the particular set of data in the data area; and

in response to modifying the key stored in the first version section to correspond to the address of the new version of the particular set of data in the data area:

modifying a current offset of the control area to point to the first version section to

15 thereby identify the first version section as the current version section,

modifying an old offset of the control area to point to a second version section to thereby identify the second version section as the old version section, and

modifying a new offset to point to a third version section to thereby identify the third version section as the new version section.