(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
5 November 2015 (05.11.2015)

WIPO | PCT

(10) International Publication Number
WO 2015/165073 A1

(51) International Patent Classification:
G06F 15/167 (2006.01)

(21) International Application Number:
PCT/CN2014/076594

(22) International Filing Date:
30 April 2014 (30.04.2014)

(25) Filing Language: English

(26) Publication Language: English

(71) Applicant: **ORACLE INTERNATIONAL CORPORATION** [US/US]; 500 Oracle Parkway, Redwood Shores, California 94065 (US).

(72) Inventor; and
(71) Applicant *(for CA only)*: **JIN, Yongshun** [CN/CN]; Oracle Building, Building No. 24, Zhongguancun Software Park, Haidian District, Beijing 100193 (CN).

(72) Inventors: **SHEN, Xugang**; Oracle Building, Building No. 24, Zhongguancun Software Park, Haidian District, Beijing 100193 (CN). **ZHANG, Qingsheng**; Oracle Building, Building No. 24, Zhongguancun Software Park, Haidian District, Beijing 100193 (CN).

(74) Agent: **CCPIT PATENT AND TRADEMARK LAW OFFICE**; 8th Floor, Vantone New World Plaza, 2 Fuchengmenwai Street, Xicheng District, Beijing 100037 (CN).

(81) Designated States *(unless otherwise indicated, for every kind of national protection available)*: AE, AG, AL, AM, AO, AT, AU, AZ, BA, BB, BG, BH, BN, BR, BW, BY, BZ, CA, CH, CL, CN, CO, CR, CU, CZ, DE, DK, DM, DO, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, GT, HN, HR, HU, ID, IL, IN, IR, IS, JP, KE, KG, KN, KP, KR, KZ, LA, LC, LK, LR, LS, LT, LU, LY, MA, MD, ME, MG, MK, MN, MW, MX, MY, MZ, NA, NG, NI, NO, NZ, OM, PA, PE, PG, PH, PL, PT, QA, RO, RS, RU, RW, SA, SC, SD, SE, SG, SK, SL, SM, ST, SV, SY, TH, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, ZA, ZM, ZW.

(84) Designated States *(unless otherwise indicated, for every kind of regional protection available)*: ARIPO (BW, GH, GM, KE, LR, LS, MW, MZ, NA, RW, SD, SL, SZ, TZ, UG, ZM, ZW), Eurasian (AM, AZ, BY, KG, KZ, RU, TJ, TM), European (AL, AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HR, HU, IE, IS, IT, LT, LU, LV, MC, MK, MT, NL, NO, PL, PT, RO, RS, SE, SI, SK, SM, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, KM, ML, MR, NE, SN, TD, TG).

(54) Title: SYSTEM AND METHOD FOR SUPPORTING ADAPTIVE SELF-TUNING LOCKING MECHANISM IN TRANSACTIONAL MIDDLEWARE MACHINE ENVIRONMENT

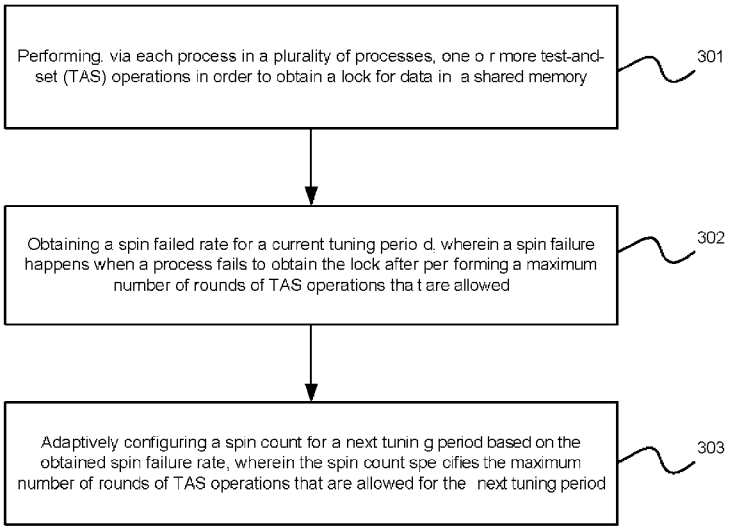

FIGURE 3

(57) Abstract: A system and method can support an adaptive self-tuning locking mechanism in a transactional middleware machine environment. The system allows each process in a plurality of processes to perform one or more test-and-set (TAS) operations in order to obtain a lock for data in a shared memory. Then, the system can obtain a spin failed rate for a current tuning period, wherein a spin failure happens when a process fails to obtain the lock after performing a maximum number of rounds of TAS operations that are allowed. Furthermore, the system can adaptively configuring a spin count for a next tuning period based on the obtained spin failure rate, wherein the spin count specifies the maximum number of rounds of TAS operations that are allowed for the next tuning period.

WO 2015/165073 A1

Published:

— *with international search report (Art. 21(3))*

SYSTEM AND METHOD FOR SUPPORTING ADAPTIVE SELF-TUNING LOCKING MECHANISM IN TRANSACTIONAL MIDDLEWARE MACHINE ENVIRONMENT

Copyright Notice

Cross Reference to Related Applications

[0002] This application is related to the following patent application(s), each of which is hereby incorporated by reference in its entirety:

[0003] U.S. Patent Application titled "SYSTEM AND METHOD FOR SUPPORTING A SELF-TUNING LOCKING MECHANISM IN A TRANSACTIONAL MIDDLEWARE MACHINE ENVIRONMENT", Application No. 13/414,593, filed March 7, 2012 (Attorney Docket No. ORACL-05255US1).

Field of the Invention

[0004] The present invention is generally related to computer systems and software such as middleware, and is particularly related to supporting a transactional middleware machine environment.

Background

[0005] A transactional middleware system, or transaction oriented middleware, includes enterprise application servers that can process various transactions within an organization. With the developments in new technologies such as high performance network and multiprocessor computers, there is a need to further improve the performance of the transactional middleware. These are the generally areas that embodiments of the invention are intended to address.

Summary

[0006] Described herein are systems and methods that can support an adaptive self-tuning locking mechanism in a transactional middleware machine environment. The system allows each process in a plurality of processes to perform one or more test-and-set (TAS) operations in order to obtain a lock for data in a shared memory. Then, the system can obtain a spin failed rate for a current tuning period, wherein a spin failure happens when a process fails to obtain the lock after

performing a maximum number of rounds of TAS operations that are allowed. Furthermore, the system can adaptively configuring a spin count for a next tuning period based on the obtained spin failure rate, wherein the spin count specifies the maximum number of rounds of TAS operations that are allowed for the next tuning period.

Brief Description of the Drawings

[0007] **Figure 1** shows an illustration of a transactional middleware machine environment that supports a lock mechanism, in accordance with an embodiment of the invention.

[0008] **Figure 2** shows an illustration of supporting an adaptive self-tuning lock mechanism in a transactional middleware machine environment, in accordance with an embodiment of the invention.

[0009] **Figure 3** illustrates an exemplary flow chart for supporting an adaptive self- tuning lock mechanism in a transactional middleware machine environment, in accordance with an embodiment of the invention.

[0010] **Figure 4** shows an illustration of dynamically increasing the spin count value in a transactional middleware machine environment that supports an adaptive self-tuning lock mechanism, in accordance with an embodiment of the invention.

[0011] **Figure 5** shows an illustration of dynamically decreasing the spin count value in a transactional middleware machine environment that supports an adaptive self-tuning lock mechanism, in accordance with an embodiment of the invention.

[0012] **Figure 6** shows an illustration of maintaining the spin count value unchanged in a transactional middleware machine environment that supports an adaptive self-tuning lock mechanism, in accordance with an embodiment of the invention.

[0013] **Figure 7** shows an illustration of configuring spin count with load surge protection in a transactional middleware machine environment that supports an adaptive self-tuning lock mechanism, in accordance with an embodiment of the invention.

[0014] **Figure 8** illustrates an exemplary flow chart for configuring spin count with load surge protection in a transactional middleware machine environment, in accordance with an embodiment of the invention.

Detailed Description

[0015] Described herein is a system and method for supporting an adaptive self-tuning locking mechanism in a transactional middleware machine environment.

[0016] In accordance with an embodiment of the invention, the system comprises a

combination of high performance hardware, e.g. 64-bit processor technology, high performance large memory, and redundant InfiniBand and Ethernet networking, together with an application server or middleware environment, such as WebLogic Suite, to provide a complete Java EE application server complex which includes a massively parallel in-memory grid, that can be provisioned quickly, and can scale on demand. In accordance with an embodiment, the system can be deployed as a full, half, or quarter rack, or other configuration, that provides an application server grid, storage area network, and InfiniBand (IB) network. The middleware machine software can provide application server, middleware and other functionality such as, for example, WebLogic Server, JRockit or Hotspot JVM, Oracle Linux or Solaris, and Oracle VM. In accordance with an embodiment, the system can include a plurality of compute nodes, IB switch gateway, and storage nodes or units, communicating with one another via an IB network. When implemented as a rack configuration, unused portions of the rack can be left empty or occupied by fillers.

[0017] In accordance with an embodiment of the invention, referred to herein as “Sun Oracle Exalogic” or “Exalogic”, the system is an easy-to-deploy solution for hosting middleware or application server software, such as the Oracle Middleware SW suite, or Weblogic. As described herein, in accordance with an embodiment the system is a “grid in a box” that comprises one or more servers, storage units, an IB fabric for storage networking, and all the other components required to host a middleware application. Significant performance can be delivered for all types of middleware applications by leveraging a massively parallel grid architecture using, e.g. Real Application Clusters and Exalogic Open storage. The system delivers improved performance with linear I/O scalability, is simple to use and manage, and delivers mission-critical availability and reliability.

[0018] In accordance with an embodiment of the invention, Tuxedo is a set of software modules that enables the construction, execution, and administration of high performance, distributed business applications and has been used as transactional middleware by a number of multi-tier application development tools. Tuxedo is a middleware platform that can be used to manage distributed transaction processing in distributed computing environments. It is a proven platform for unlocking enterprise legacy applications and extending them to a services oriented architecture, while delivering unlimited scalability and standards-based interoperability.

[0019] In accordance with an embodiment of the invention, a transactional middleware system, such as a Tuxedo system, can take advantage of fast machines with multiple processors, such as an Exalogic middleware machine, and a high performance network connection, such as an Infiniband (IB) network.

Lock Mechanisms

[0020] **Figure 1** shows an illustration of a transactional middleware machine environment that supports a lock mechanism, in accordance with an embodiment of the invention. As shown in Figure 1, a transactional middleware environment 100 can employ a lock mechanism 103 for protecting various transaction data 102 in a shared memory 101, e.g. the bulletin board (BB) in the Tuxedo environment, when there are concurrent transactions (i.e. on processes 111-115).

[0021] In accordance with an embodiment of the invention, the transactional middleware environment 100 can take advantage of the multi-processor machines by using an atomic TAS (Test-And-Set) 104 assembly component for implementing an effective locking mechanism. Additionally, a process in a transactional application can use a semaphore mechanism 107 that is provided by the operating system (OS) to obtain a lock on data 102, if necessary.

[0022] For example, when a process 111 wants to get a lock on data 102, the process 111 can perform a TAS operation for a number of rounds. The system can specify a spin count 105, which is the maximum number of rounds of TAS operation that are allowed.

[0023] As shown in Figure 1, if the lock 103 becomes available before the spin count 105 is reached, the process 111 can obtain the lock 103 with much less cost than the semaphore 107 mechanism provided by the OS.

[0024] Otherwise, if the lock 103 is not available after the process 111 has performed the maximum number of rounds of TAS operations that are allowed, the process 111 can block on the semaphore 107 and can wait until a lock owner releases the lock 103. For example, the lock requests blocking on the semaphore 107 can be put into a queue, e.g. a semaphore waiting queue 106.

[0025] Additionally, the lock requests blocking on the semaphore 107 can have a higher priority than the lock requests that are based on the TAS assembly component 104. Thus, the lock holder will first release the lock 103 to a process in the semaphore waiting queue 106 as long as the semaphore waiting queue 106 is not empty.

Adaptive Self-Tuning Lock Mechanism

[0026] **Figure 2** shows an illustration of supporting an adaptive self-tuning lock mechanism in a transactional middleware machine environment, in accordance with an embodiment of the invention. As shown in Figure 2, a transactional middleware environment 200 can employ a lock mechanism 203 for protecting various transaction data 202 in a shared memory 201, when there are concurrent transactions (i.e. on processes 211-215).

[0027] For example, when the processes 211-215 want to get a lock 203 on data 202, each of the process 211-215 can perform a TAS 204 operation for a number of rounds. The system can specify a spin count 205, which is the maximum number of rounds of TAS operations that are allowed. In accordance with an embodiment of the invention, metadata, such as a SPINCOUNT parameter in the Tuxedo configuration file, can be used to specify a default value and/or an initial value for the spin count 205.

[0028] As shown in Figure 2, if the lock 203 becomes available before the spin count 205 is reached, a process (e.g. one of the processes 211-212 and 214-215) can obtain the lock 203 with much less cost than the semaphore mechanism provided by the OS.

[0029] Otherwise, if the lock 203 is not available before the spin count 205 is reached (i.e. a spin failure happens), then a process (e.g. the process 213) can be configured to block on the semaphore, and wait until the lock owner releases the lock 203 and wakes it up.

[0030] Furthermore, the spin count value 205 can be stored in the shared memory 201. A special process, such as a Tuxedo daemon process, can periodically tune (or change) the spin count value 205 according to operation information collected in the previous tuning period. For example, the Tuxedo daemon can update the target spin count value once every five seconds by default.

[0031] In accordance with one embodiment, different algorithms can be used to calculate and configure the spin count 205 value. For example, when the CPU idle ratio for the current tuning period is sufficient, a simple algorithm 206 can increase the spin count 205 value, if the spin failed rate 208 is higher than the target (i.e. too many TAS 204 operations have failed to obtain the lock 203 in the current tuning period and have switched to the semaphore). Furthermore, the simple algorithm can decrease the spin count 205 value if the CPU idle ratio is too high.

[0032] While the simple algorithm 206 is easy to implement, the simple algorithm 206 may encounter different problems when it runs on a real resource manager (RM), e.g. an Oracle database. For example, the simple algorithm 206 may generate an extreme large spin count value 205, since the simple algorithm 206 will increase the spin count value 205 as long as the spin failed ratio is sub-standard. Also, the simple algorithm 206 may not be able to fine tune the spin count value 205, since the step taken by the simple algorithm 206 to increase the spin count value 205 tends to be too big and the spin count value 205 may reach the top limitation in a few rounds of tuning. Additionally, the simple algorithm 206 may generate a large spin count value 205, which actually cause a worse throughput in the system when running on a real RM. Furthermore, the simple algorithm 206 may not decrease the spin count value 205 when the idle CPU ratio is high enough. Moreover, it can be difficult for the simple algorithm 206 to configure the

default original spin count value 205 and the default target of spin failed rate 208 accordingly to the real RM environment.

[0033] Alternatively, the system can employ an adaptive algorithm 207 to dynamically calculate the spin count 205 value in real time. The adaptive algorithm 207 can avoid various problems that may occur when the system runs on a real resource manager (RM), e.g. an Oracle database, using the simple algorithm 206.

[0034] In accordance with an embodiment of the invention, the adaptive algorithm 207 can prevent bad tuning by keeping good tuning. For example, the adaptive algorithm 207 can store a spin count value 205 and a spin failed rate 208 from a last good tuning period.

[0035] For each tuning period 210, the system can check whether the current spin failed rate 208 is better than the stored last good spin failed ratio. If the current spin failed rate 208 is better than the last good spin failed ratio, the system can consider the current tuning period 210 as a good tuning. Then, the system can cache the current spin count value 205 and the current spin failed rate 208. On the other hand, if the current spin failed rate increases (i.e. becomes worse) after a tuning, the system can use the stored last good spin count value 205 for the next tuning period 210.

[0036] Thus, transactional middleware environment 200 can support massive concurrent transactions scenarios and achieve high throughput.

[0037] **Figure 3** illustrates an exemplary flow chart for supporting an adaptive self- tuning lock mechanism in a transactional middleware machine environment, in accordance with an embodiment of the invention. As shown in Figure 3, at step 301, each process can perform one or more test-and-set (TAS) operations in order to obtain a lock for data in a shared memory. Additionally, at step 302, the system can obtain a spin failed rate for a current tuning period, wherein a spin failure happens when a process fails to obtain the lock after performing a maximum number of rounds of TAS operations that are allowed. Furthermore, at step 303, the system can adaptively configure the spin count value for a next tuning period based on the obtained spin failure rate, wherein the spin count specifies the maximum number of rounds of TAS operations that are allowed for the next tuning period.

Adaptive Algorithm to Dynamically Calculate the Target Spin Count Value

[0038] In accordance with an embodiment of the invention, the system can use an adaptive algorithm to dynamically determine the target spin count value in real time. Also, the system can calculate the target spin count value in the context of the hardware configuration and the application scenario.

[0039] For example, in the Tuxedo environment, the system can calculate the spin count value by calling a function as shown in the following.

```
static int _calc_spintuning(_TCADEF)
```

[0040] In Tuxedo, an application can call the above function in each tuning period, such as in each scan unit (which can be configured using the parameter SCANUNIT in the RESOURCE section).

[0041] Additionally, as shown in the following, the system can call another function, which is responsible for retrieving the CPU ratios.

```
static int getCPUrate(int type, float* rate, int size)
```

[0042] The implementation for the above function can be platform-dependent. For example, the above function can obtain the CPU ratios via system tools, such as the file /proc/stat tools in the Exalogic Elastic Cloud (Linux 64bit) platform. Alternatively, the above function can obtain the CPU ratios via system libraries, such as the kstat library In the SPARC SuperCluster (Sparc 64bit) platform.

[0043] **Figure 4** shows an illustration of dynamically increasing the spin count value in a transactional middleware machine environment that supports an adaptive self-tuning lock mechanism, in accordance with an embodiment of the invention. As shown in Figure 4, the adaptive algorithm can dynamically increase the spin count when it is appropriate.

[0044] At step 401, the system can check whether the current idle CPU is sufficient, i.e. whether an idle CPU rate for the current tuning period is more than a user-configured minimum idle CPU rate. Also, at step 402, the system can check whether the current spin failed ratio is below a user-configured target.

[0045] Then, the adaptive algorithm may decide not to increase the spin count, when either the current idle CPU is sufficient or the current spin failed ratio for the current tuning period already meets the user-configured target.

[0046] Otherwise, at step 403, the adaptive algorithm can check whether the current spin failed ratio is better than the stored last good spin failed ratio. Also, at step 404, the adaptive algorithm can check whether the current spin failed ratio becomes worse without tuning.

[0047] As a result, at step 405, the system can increase the spin count, if the spin failed ratio for the current tuning period is less than the last good spin failed ratio, or if the spin failed ratio for

the current tuning period becomes worse without tuning.

[0048] Finally, at step 406, the system can proceed to the next tuning period, which can lead the process back to repeat the above steps 401-405 for the next tuning period.

[0049] In Tuxedo, the system can use the following algorithm to determine the increase of the spin count (i.e. calculating the tuned SPINCOUTN for the next tuning period).

```
Tuned SPINCOUNT +=
        (SPINCOUNT * base_factor) * min(max_times,(idle CPU ratio/ user CPU ratio))
```

[0050] The above algorithm uses two factors to fine tune the increase of the SPINCOUTN, which depends on the current SPINCOUTN value and the value of idle CPU ratio / user CPU ratio. The first factor, base_factor, which can be used to reduce the contribution of the current SPINCOUNT, is smaller than 1. The second factor, max_times, can be used as the top limitation of idle CPU ratio / user CPU ratio.

[0051] Additionally, the range of possible SPINCOUNT values can be divided into several intervals. The following Table 1 shows an exemplary division of several intervals.

SPINCOUNT interval	[bottom limitation, 8*1024)	[8*1024, 64*1024)	[64*1024, 512*1024)	[512*1024, top limitation)
Base_factor	1.0	0.75	0.5	0.25
max_times	1.0	1.0	1.0	1.0

Table 1

[0052] As shown in the above Table 1, the different intervals can be configured with different base_factor and max_times values. In order to ensure that the SPINCOUNT can gradually reach the target value, the values for the base_factor and the max_times can be set to be smaller as the SPINCOUNT becomes bigger.

[0053] **Figure 5** shows an illustration of dynamically decreasing the spin count value in a transactional middleware machine environment that supports an adaptive self-tuning lock mechanism, in accordance with an embodiment of the invention. As shown in Figure 5, the adaptive algorithm can dynamically decrease the spin count when it is appropriate.

[0054] At step 501, the adaptive algorithm can check whether application is idle. Also, at step 502, the adaptive algorithm can check whether current idle CPU ratio is larger than the limitation and the user CPU ratio is sufficient, and at step 503, the adaptive algorithm can check whether

current spin count has been kept unchanged (or stable) for a period of time that is long enough.

[0055] Then, at step 504, the algorithm can decrease the spin count if the application is idle. For example, Tuxedo can use the following formula to decrease the SPINCOUNT value.

```
Tuned SPINCOUNT -= Tuned SPINCOUNT>>3
```

[0056] Using the above algorithm, Tuxedo can bring a high SPINCOUNT value automatically back to the original SPINCOUNT value when the application becomes idle.

[0057] Also, at step 504, the algorithm can decrease the spin count if the current idle CPU ratio is larger than the limitation (as configured by the user) and the user CPU ratio is sufficient. For example, Tuxedo can use the following formula to decrease the SPINCOUNT value.

```
Tuned SPINCOUNT = Tuned SPINCOUNT>>2
```

[0058] Additionally, at step 504, the algorithm can decrease the spin count if the current SPINCOUNT has been kept stable for a period time that is long enough. For example, Tuxedo can use the following formula to decrease the SPINCOUNT value.

```
Tuned SPINCOUNT -= Tuned SPINCOUNT>>3
```

[0059] Using the above algorithm, Tuxedo allows the SPINCOUNT value to be decreased after a long stable time. Thus, Tuxedo can automatically bring a high SPINCOUNT back to a proper value when the load becomes lighter.

[0060] Finally, at step 505, the system can proceed to the next tuning period, which can lead the process back to repeat the above steps 501-504 for the next tuning period.

[0061] **Figure 6** shows an illustration of maintaining the spin count value unchanged in a transactional middleware machine environment that supports an adaptive self-tuning lock mechanism, in accordance with an embodiment of the invention. As shown in Figure 6, the adaptive algorithm can maintain the spin count unchanged in different scenarios.

[0062] At step 601, the adaptive algorithm can check whether the current spin failed ratio meets the requirement. Also, at step 602, the adaptive algorithm can check whether the current spin count reaches the top or bottom limitation when tuning, and at step 603, the adaptive algorithm can check whether the current spin failed ratio is kept stable.

[0063] Then, at step 604, the algorithm can keep the spin count unchanged if the current spin failed ratio meets the requirement, or if the current SPINCOUNT reaches the top or bottom limitation when tuning, or if the current spin failed ratio is kept stable.

Configuring the Spin Count with Load Surge Protection

[0064] **Figure 7** shows an illustration of configuring spin count with load surge protection in a transactional middleware machine environment that supports an adaptive self-tuning lock mechanism, in accordance with an embodiment of the invention. As shown in Figure 7, a transactional middleware environment 700 can employ a lock 703 for protecting various transaction data 702 in a shared memory 701, when there are concurrent transactions (i.e. on processes 711-715).

[0065] Furthermore, the transactional middleware environment 700 can use an atomic TAS (Test-And-Set) 704 assembly component to implement an effective locking mechanism. Additionally, the system can put a process (e.g. the processes 711-713) in a semaphore waiting queue 706, when a spin failure happens.

[0066] As shown in Figure 7, the lock requests by the processes 711-713, which wait in the semaphore waiting queue 706, can have a higher priority than the lock requests by the processes 714-715, which are based on the TAS assembly component 704.

[0067] Thus, a lock holder may first release the lock 703 to the processes 711-713 in the semaphore waiting queue 706, and the processes 714-715 may not have access to the lock 703 as long as the semaphore waiting queue 706 is not empty.

[0068] In accordance with an embodiment of the invention, the system can dynamically determine the spin count value in real time, since the length of the semaphore waiting queue 706 may vary from time to time.

[0069] As shown in Figure 7, the system can apply extra spins on the processes 714-715, which use the TAS assembly component 704. For example, in Tuxedo, the system can use the following algorithm to determine the actual used spin count (i.e. the used SPINCOUTN) in real time.

Used SPINCOUNT = tuned SPINCOUNT+ extraspin * depth of the semaphore waiting queue

[0070] As shown in the above, when calculating the used SPINCOUNT, Tuxedo can take into account of the current depth of the semaphore waiting queue. The deeper the semaphore waiting queue is, the bigger the used SPINCOUNT can be set.

[0071] In accordance with an embodiment of the invention, when a load surge occurs in the

transactional middleware machine environment 700, there can be a sudden increase of spin failures at the TAS assembly component 704, which may lead to more processes waiting for the lock 703 in the semaphore waiting queue 706.

[0072] By applying extra spins on the processes 714-715, which use the TAS assembly component 704, the system can reduce the spin failed ratio when the semaphore queue is deep. Furthermore, when the load in the system eventually becomes light, the length of the semaphore waiting queue 706 is shortened. Thus, the actual used spin count can be reduced in the real time within a tuning period.

[0073] **Figure 8** illustrates an exemplary flow chart for configuring spin count with load surge protection in a transactional middleware machine environment, in accordance with an embodiment of the invention. As shown in Figure 8, at step 801, a process with a spin failure can wait in a semaphore waiting queue for a release of a lock on the data in the shared memory. Then, at step 802, the process can access the data in the shared memory, when a lock owner releases the lock on the data. Furthermore, at step 803, the system can apply extra spin counts on each process performing the TAS operation, when the semaphore waiting queue is not empty.

[0074] The present invention may be conveniently implemented using one or more conventional general purpose or specialized digital computer, computing device, machine, or microprocessor, including one or more processors, memory and/or computer readable storage media programmed according to the teachings of the present disclosure. Appropriate software coding can readily be prepared by skilled programmers based on the teachings of the present disclosure, as will be apparent to those skilled in the software art.

[0075] In some embodiments, the present invention includes a computer program product which is a storage medium or computer readable medium (media) having instructions stored thereon/in which can be used to program a computer to perform any of the processes of the present invention. The storage medium can include, but is not limited to, any type of disk including floppy disks, optical discs, DVD, CD-ROMs, microdrive, and magneto-optical disks, ROMs, RAMs, EPROMs, EEPROMs, DRAMs, VRAMs, flash memory devices, magnetic or optical cards, nanosystems (including molecular memory ICs), or any type of media or device suitable for storing instructions and/or data.

[0076] The foregoing description of the present invention has been provided for the purposes of illustration and description. It is not intended to be exhaustive or to limit the invention to the precise forms disclosed. Many modifications and variations will be apparent to the practitioner skilled in the art. The modifications and variations include any relevant combination of the disclosed features. The embodiments were chosen and described in order to best explain the

principles of the invention and its practical application, thereby enabling others skilled in the art to understand the invention for various embodiments and with various modifications that are suited to the particular use contemplated. It is intended that the scope of the invention be defined by the following claims and their equivalence.

Claims

What is claimed is:

1. A method for supporting an adaptive locking mechanism in a transactional middleware machine environment, comprising:

performing, via each process in a plurality of processes, one or more test-and-set (TAS) operations in order to obtain a lock for data in a shared memory;

obtaining a spin failed rate for a current tuning period, wherein a spin failure happens when a process fails to obtain the lock after performing a maximum number of rounds of TAS operations that are allowed; and

adaptively configuring a spin count for a next tuning period based on the obtained spin failure rate, wherein the spin count specifies the maximum number of rounds of TAS operations that are allowed for the next tuning period.

2. The method of claim 1, further comprising:

using an assembly component to perform said one or more TAS operations.

3. The method of claim 1, further comprising:

preconfiguring the spin count using metadata.

4. The method of claim 1, further comprising:

determining whether an idle CPU rate for the current tuning period is more than a user-configured minimum idle CPU rate and whether a spin failed ratio for the current tuning period meets a user-configured target.

5. The method of claim 4, further comprising:

when an idle CPU rate for the current tuning period is more than a user-configured minimum idle CPU rate and a spin failed ratio for the current tuning period does not meet a user-configured target, increasing the spin count

if the spin failed ratio for the current tuning period is better than a last good spin failed ratio, or

if the spin failed ratio for the current tuning period becomes worse without tuning.

6. The method of claim 5, further comprising:

dividing a range for possible spin count values into several intervals, and

using a different formula to calculate a tuned spin count value in each different interval.

7. The method of claim 1, further comprising:

decreasing the spin count, if at least one of

an application is idle,

the idle CPU ratio is larger than a user-configured minimum idle CPU rate and the user CPU ratio is sufficient, and

the spin count has been kept stable for a period of time.

8. The method of claim 1, further comprising:

keeping the spin count, if at least one of

the spin failed ratio for the current tuning period meets the requirement,

the spin count for the current tuning period reaches the top or bottom limitation, and

the spin failed ratio for the current tuning period is kept stable.

9. The method of claim 1, further comprising:

waiting, via a process with a spin failure, in a semaphore waiting queue for a release of a lock on the data in the shared memory; and

accessing the data in the shared memory, when a lock owner releases the lock on the data.

10. The method of claim 9, further comprising:

applying extra spin counts on each process performing the TAS operation, when the semaphore waiting queue is not empty.

11. A system for providing an adaptive locking mechanism in a transactional middleware machine environment, comprising:

one or more processors;

a plurality of processes, wherein each process in the plurality of processes operates to perform one or more test-and-set (TAS) operations in order to obtain a lock for data in a shared

memory;

a transactional server, running on the one or more processors, operate to

obtain a spin failed rate for a current tuning period, wherein a spin failure happens when a process fails to obtain the lock after performing a maximum number of rounds of TAS operations that are allowed; and

adaptively configure a spin count for a next tuning period based on the obtained spin failure rate, wherein the spin count specifies the maximum number of rounds of TAS operations that are allowed for the next tuning period.

12. The system of claim 11, wherein:

an assembly component is used to perform said one or more TAS operations.

13. The system of claim 11, wherein:

the spin count is preconfigured using metadata.

14. The system of claim 11, wherein:

the managing component operates to determine whether an idle CPU rate for the current tuning period is more than a user-configured minimum idle CPU rate and whether a spin failed ratio for the current tuning period meets a user-configured target.

15. The system of claim 14, wherein:

when an idle CPU rate for the current tuning period is more than a user-configured minimum idle CPU rate and a spin failed ratio for the current tuning period does not meet a user-configured target, the managing component operates to increase the spin count

if the spin failed ratio for the current tuning period is better than a last good spin failed ratio, or

if the spin failed ratio for the current tuning period becomes worse without tuning.

16. The system of claim 15, wherein:

a range for possible spin count values is divided into several intervals, and the managing component operates to use a different formula to calculate a tuned spin count value in each different interval.

17. The system of claim 11, wherein:

the managing component operates to decrease the spin count, if at least one of

an application is idle,

the idle CPU ratio is larger than a user-configured minimum idle CPU rate and the user CPU ratio is sufficient, and

the spin count has been kept stable for a period of time.

18. The system of claim 11, wherein:

the managing component operates to keep the spin count, if at least one of

the spin failed ratio for the current tuning period meets the requirement,

the spin count for the current tuning period reaches the top or bottom limitation, and

the spin failed ratio for the current tuning period is kept stable.

19. The system of claim 11, wherein:

the managing component operates to

put a process with a spin failure to wait in a semaphore waiting queue for a release of a lock on the data in the shared memory;

access the data in the shared memory, when a lock owner releases the lock on the data; and

apply extra spin counts on the processes performing the TAS operation, when the semaphore waiting queue is not empty.

20. A non-transitory machine readable storage medium having instructions stored thereon that when executed cause a system to perform the steps comprising:

performing, via each process in a plurality of processes, one or more test-and-set (TAS) operations in order to obtain a lock for data in a shared memory;

obtaining a spin failed rate for a current tuning period, wherein a spin failure happens when a process fails to obtain the lock after performing a maximum number of rounds of TAS operations that are allowed; and

adaptively configuring a spin count for a next tuning period based on the obtained spin failure rate, wherein the spin count specifies the maximum number of rounds of TAS operations that are allowed for the next tuning period.

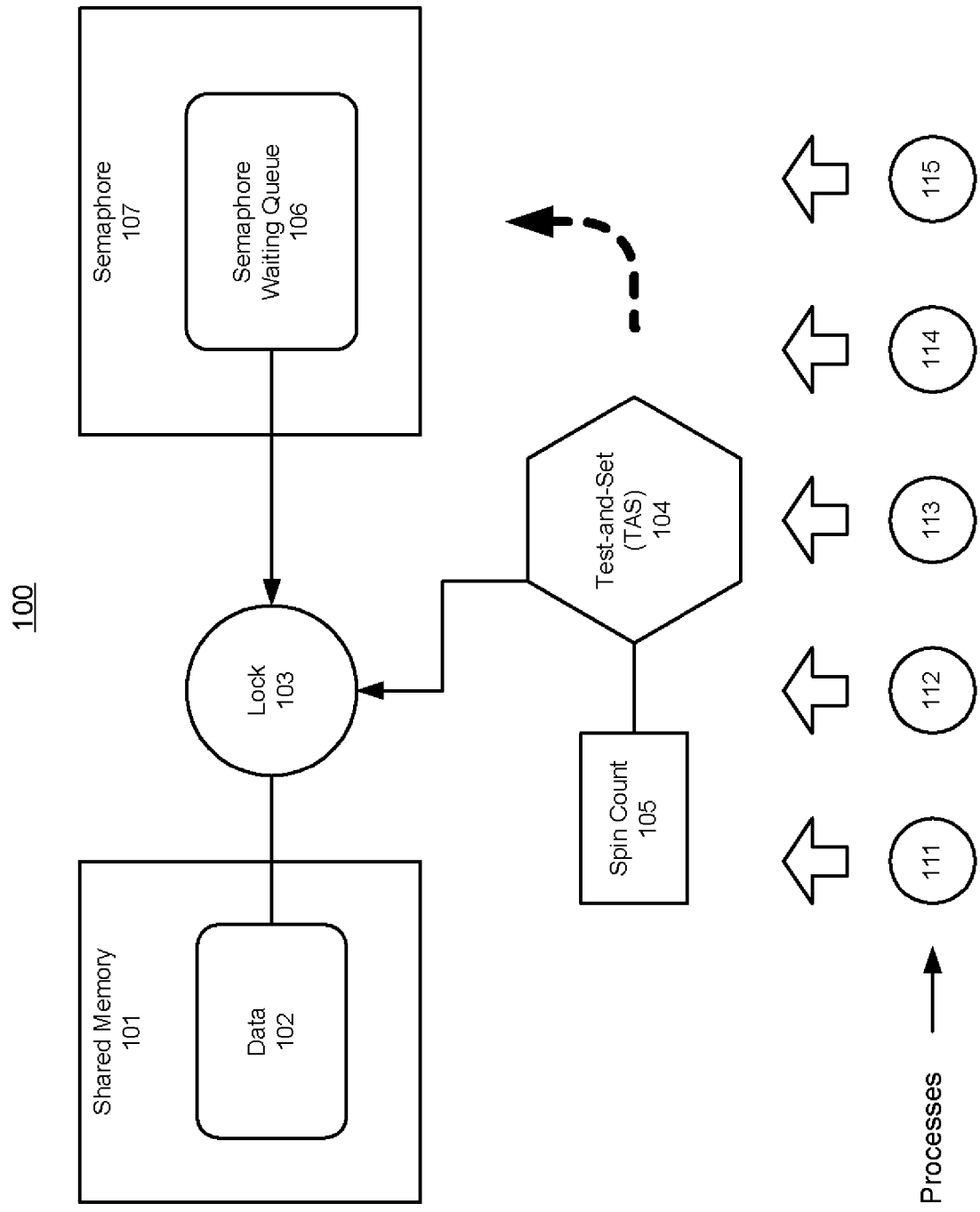
100
Shared Memory
101
Data
102
Lock
103
Spin Count
105
Test-and-Set
(TAS)
104
Semaphore
107
Semaphore
Waiting Queue
106
Processes
111
112
113
114
115

FIGURE 1

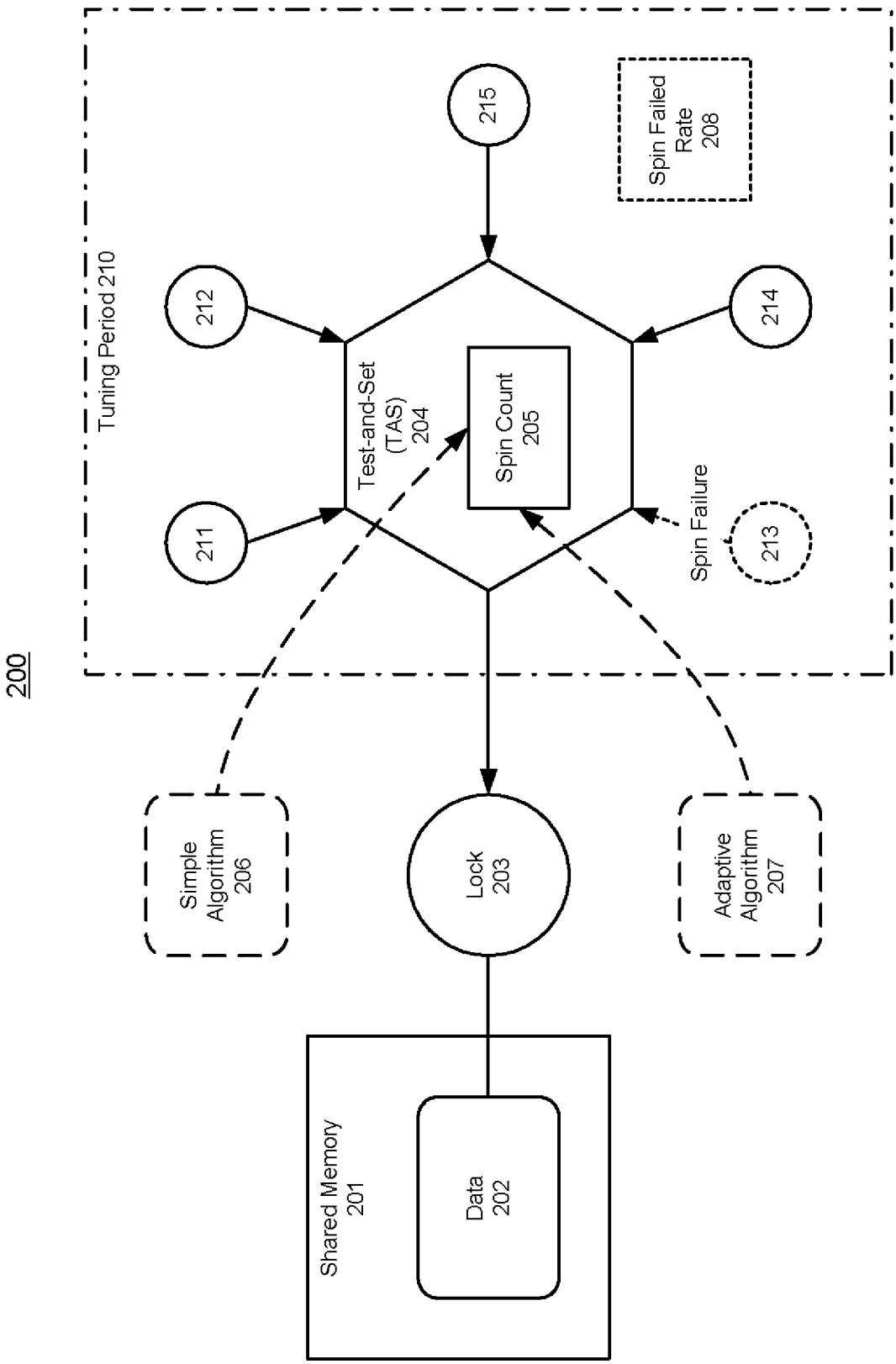
200
Tuning Period 210
215
Spin Failed Rate 208
212
214
Test-and-Set (TAS) 204
Spin Count 205
211
Spin Failure
213
Simple Algorithm 206
Lock 203
Adaptive Algorithm 207
Shared Memory 201
Data 202

FIGURE 2

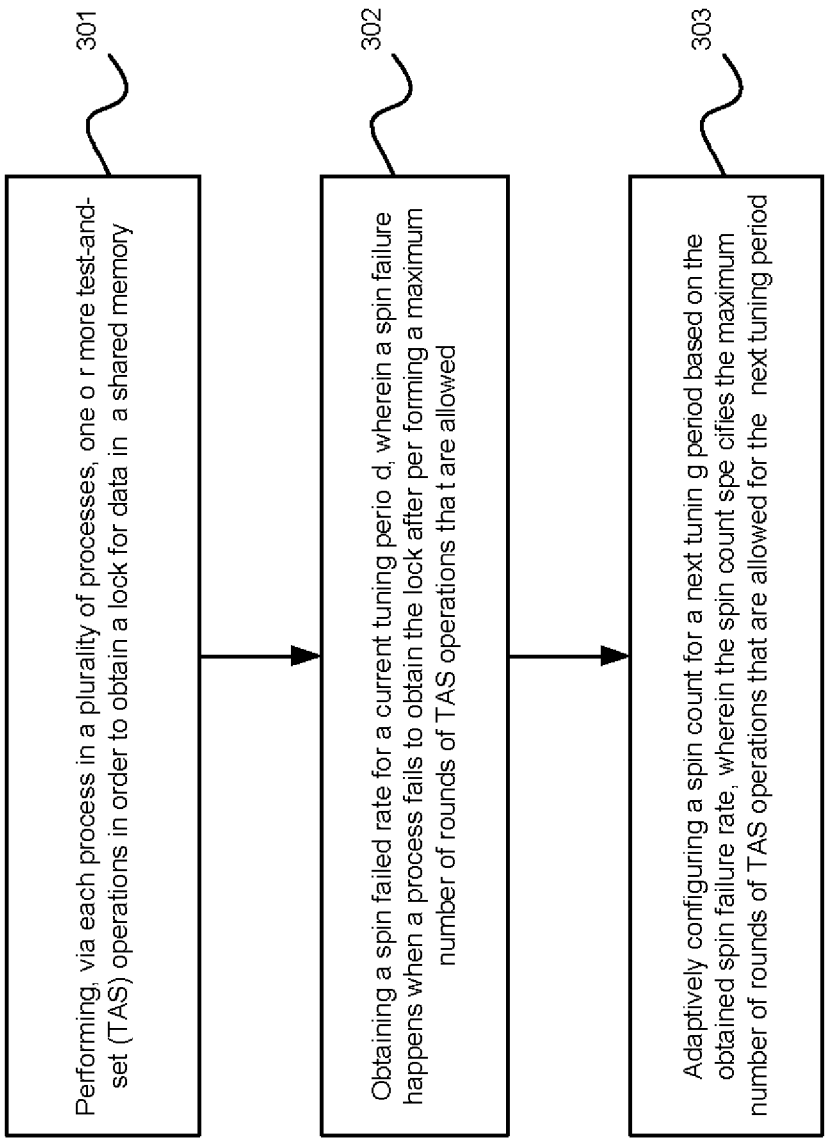
301
Performing, via each process in a plurality of processes, one o r more test-and-set (TAS) operations in order to obtain a lock for data in a shared memory
302
Obtaining a spin failed rate for a current tuning perio d, wherein a spin failure happens when a process fails to obtain the lock after per forming a maximum number of rounds of TAS operations tha t are allowed
303
Adaptively configuring a spin count for a next tunin g period based on the obtained spin failure rate, wherein the spin count spe cifies the maximum number of rounds of TAS operations that are allowed for the next tuning period

FIGURE 3

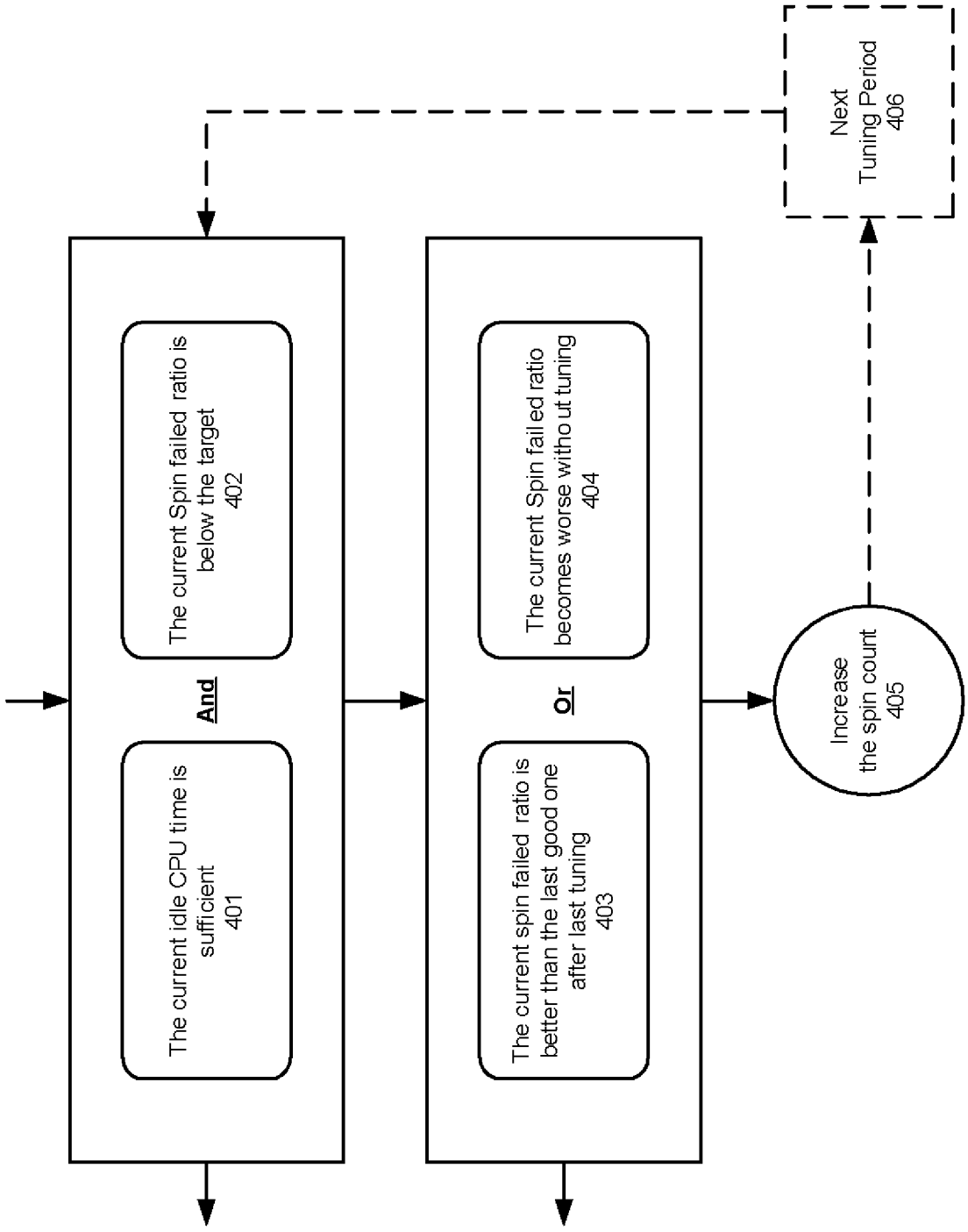
Next
Tuning Period
406
The current Spin failed ratio is
below the target
402
And
The current idle CPU time is
sufficient
401
The current Spin fail ed ratio
becomes worse witho ut tuning
404
Or
The current spin failed ratio is
better than the last good one
after last tuning
403
Increase
the spin count
405

FIGURE 4

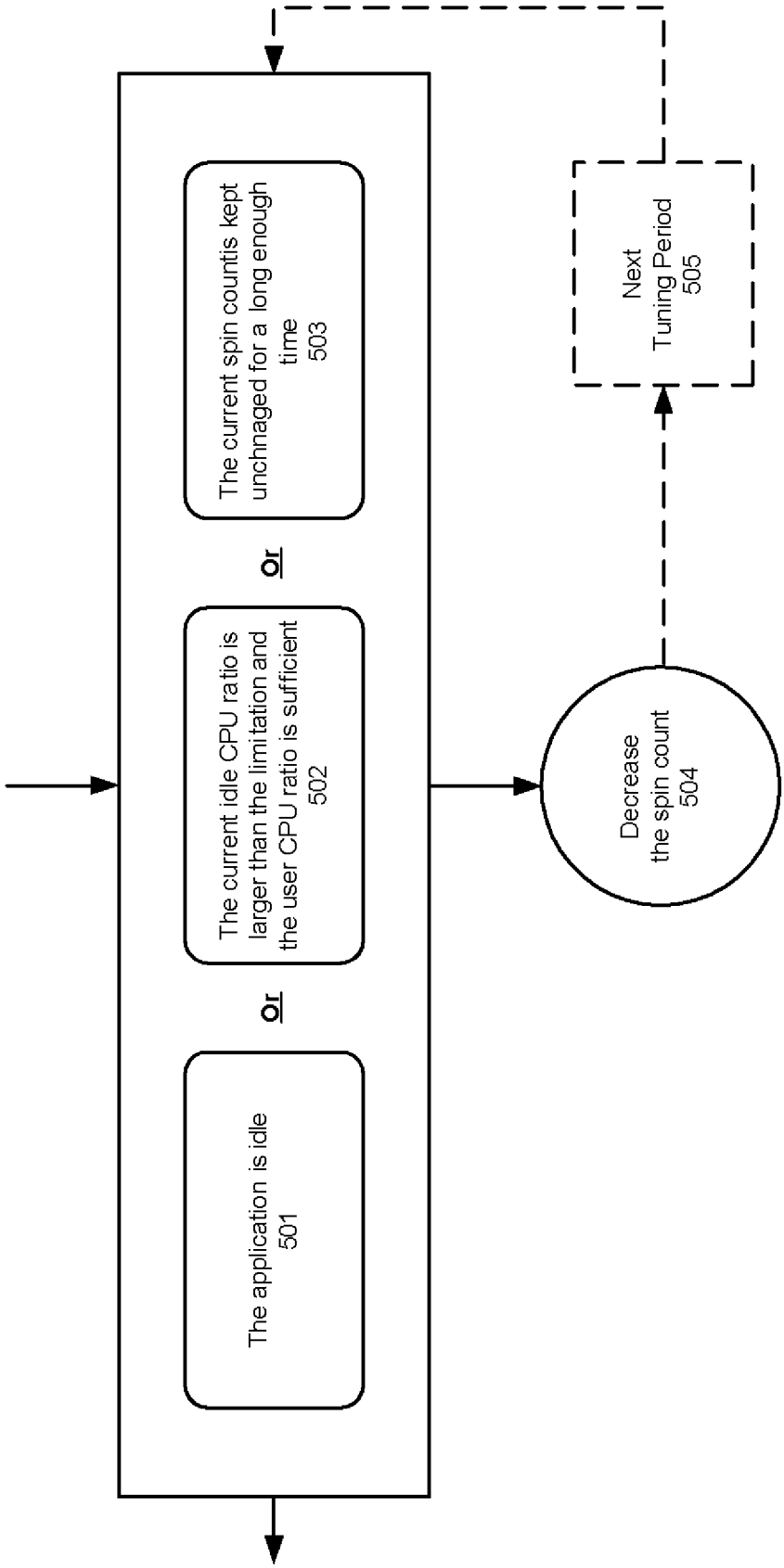
The application is idle
501
Or
The current idle CPU ratio is larger than the limitation and the user CPU ratio is sufficient
502
Or
The current spin countis kept unchnaged for a long enough time
503
Decrease the spin count
504
Next Tuning Period
505

FIGURE 5

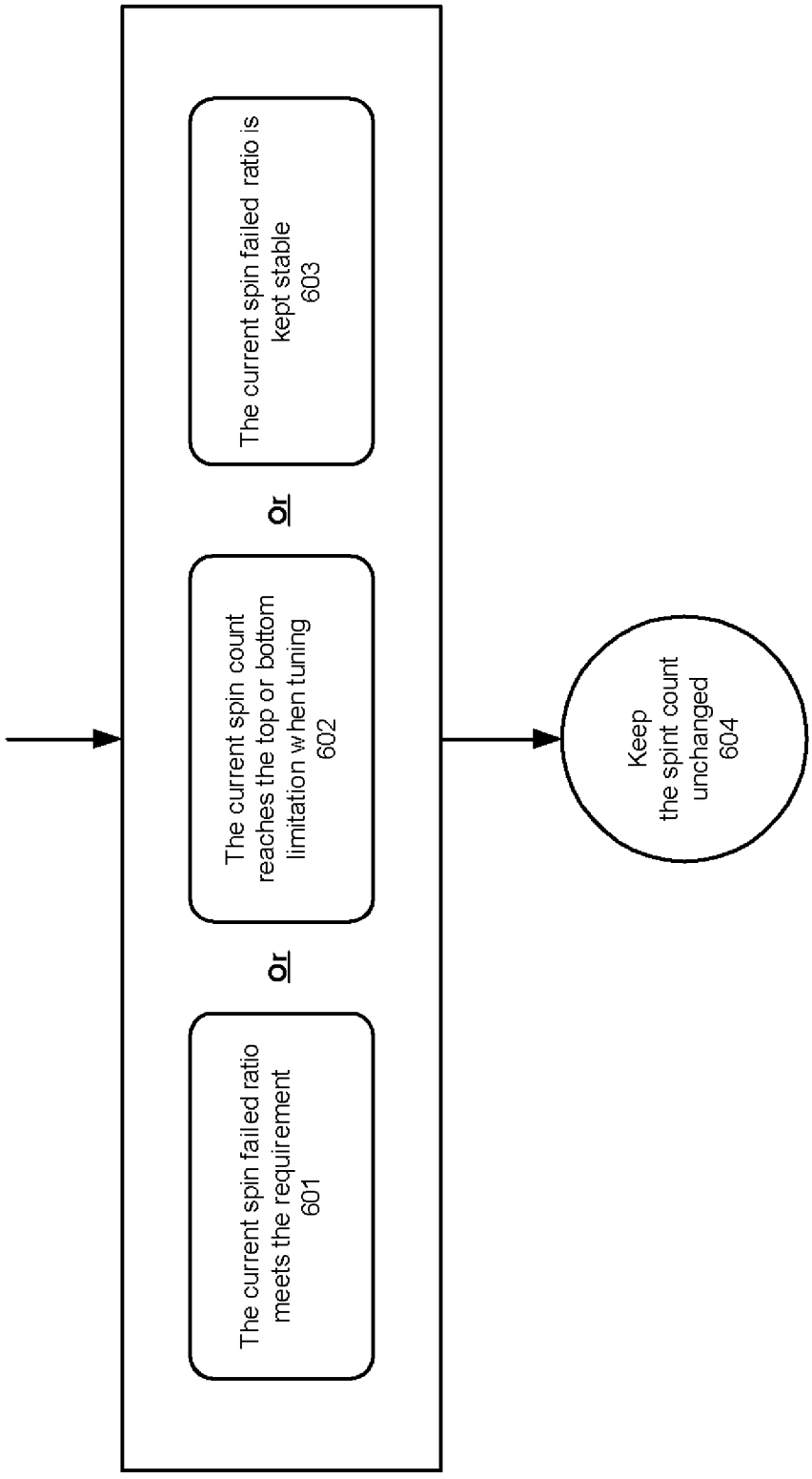
The current spin failed ratio meets the requirement
601
Or
The current spin count reaches the top or bottom limitation when tuning
602
Or
The current spin failed ratio is kept stable
603
Keep the spint count unchanged
604

FIGURE 6

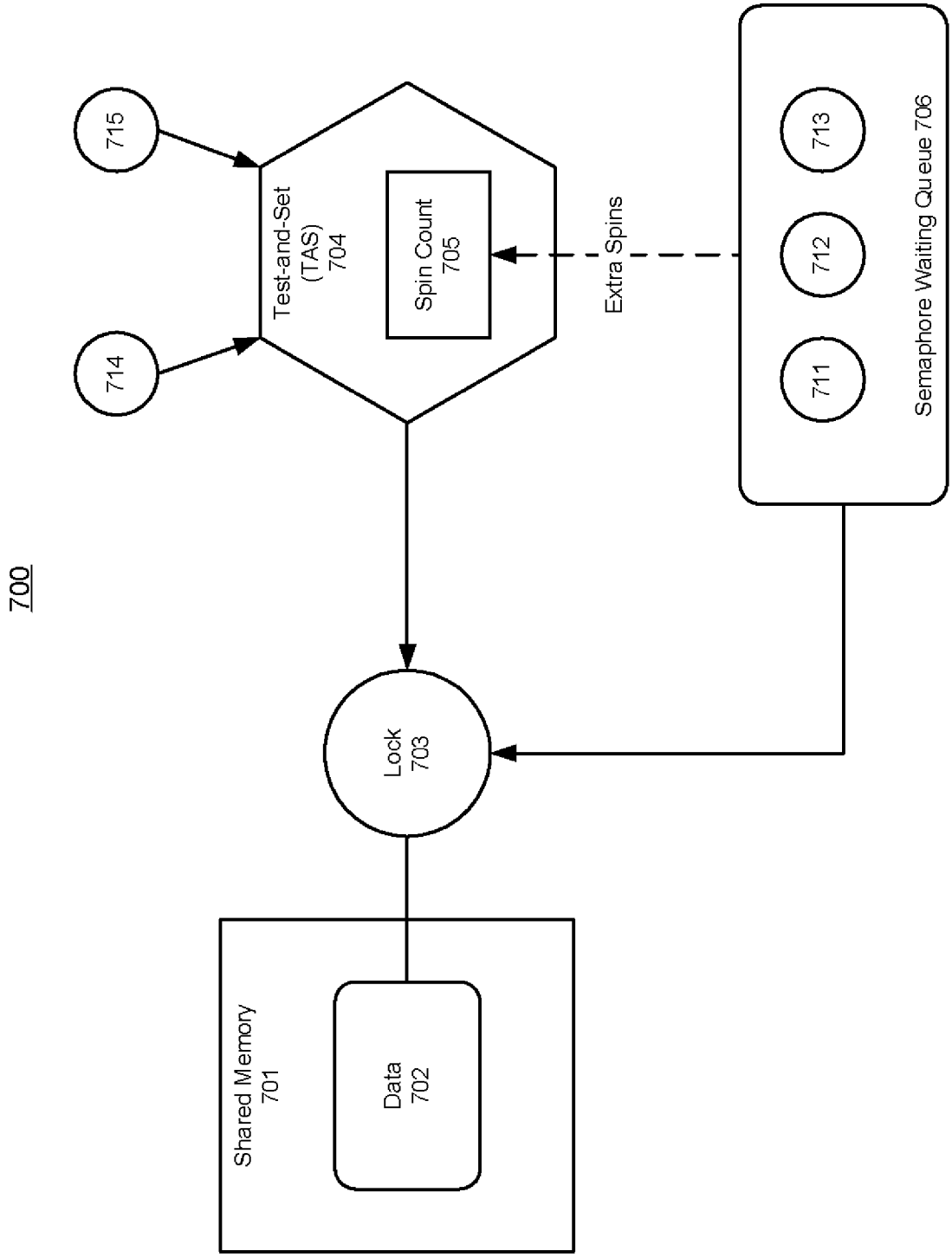
700
715
714
Test-and-Set
(TAS)
704
Spin Count
705
Extra Spins
713
712
711
Semaphore Waiting Queue 706
Lock
703
Shared Memory
701
Data
702

FIGURE 7

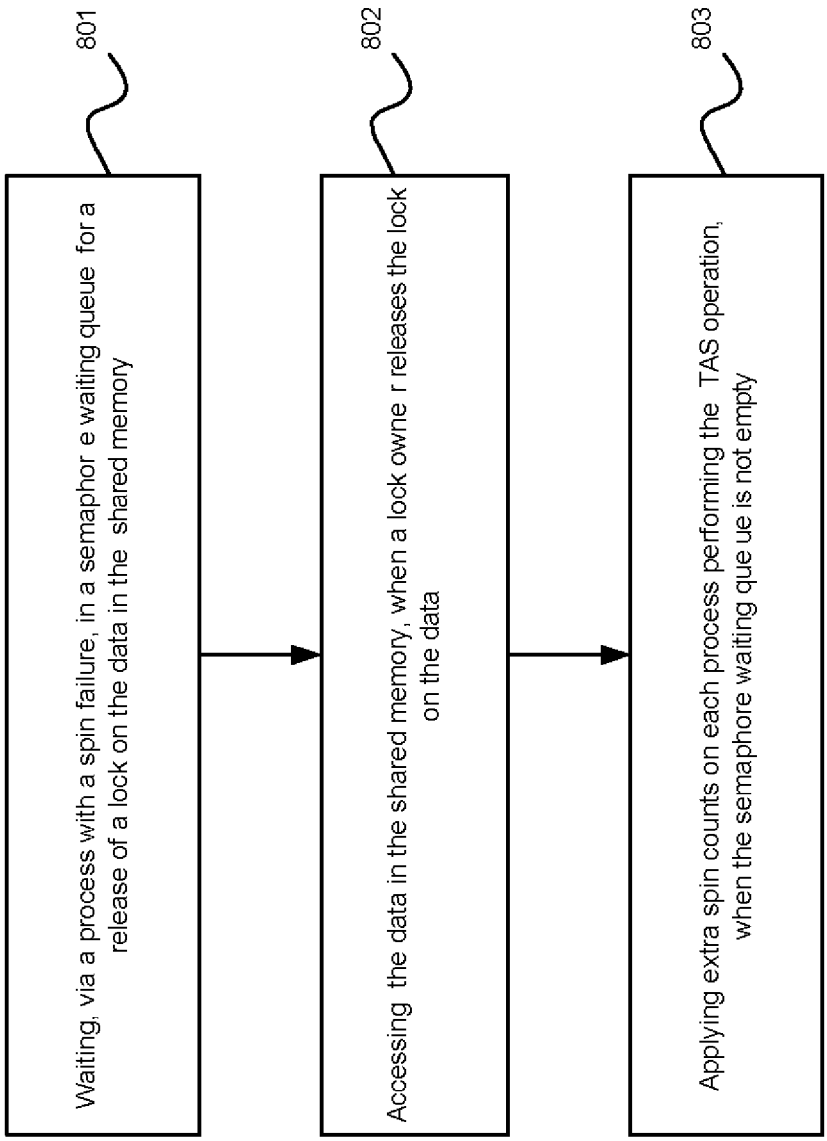
801
Waiting, via a process with a spin failure, in a semaphor e waiting queue for a release of a lock on the data in the shared memory
802
Accessing the data in the shared memory, when a lock owne r releases the lock on the data
803
Applying extra spin counts on each process performing the TAS operation, when the semaphore waiting que ue is not empty

FIGURE 8

INTERNATIONAL SEARCH REPORT

International application No.
PCT/CN2014/076594

A. CLASSIFICATION OF SUBJECT MATTER

G06F 15/167(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; H04L

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

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

WPI,EPODOC,CNKI,CNPAT: lock+, test?and?set, TAS, share+, atomic, fail+, count?, number?, round?, time?, tun+, chang+, verify+, decrease+, increase+, transaction+, rate, ratio, spin, last, next, current, period

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	WO 2013048826 A1 (ORACLE INTERNATIONAL CORPORATION) 04 April 2013 (2013-04-04) abstract, claims 11-19,21,22 description, paragraphs 0005,0015-0017,0022-0029,0066, 0067, figures 1-2	1-4,9-14,19,20
A	WO 2013048826 A1 (ORACLE INTERNATIONAL CORPORATION) 04 April 2013 (2013-04-04) abstract, claims 11-19,21,22 description, paragraphs 0005,0015-0017,0022-0029,0066, 0067, figures 1-2	5-8,15-18
A	US 2010088702 A1 (MICROSOFT CORPORATION) 08 April 2010 (2010-04-08) the whole document	1-20
A	US 7594234 B1 (SUN MICROSYSTEMS, INC.) 22 September 2009 (2009-09-22) the whole document	1-20
A	CN 101256509 A (ZTE CORP.) 03 September 2008 (2008-09-03) the whole document	1-20

☑ 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	Date of mailing of the international search report
14 January 2015	**28 January 2015**

Name and mailing address of the ISA/CN	Authorized officer
STATE INTELLECTUAL PROPERTY OFFICE OF THE P.R.CHINA(ISA/CN) **6,Xitucheng Rd., Jimen Bridge, Haidian District, Beijing 100088 China**	**DONG,Hongmei**
Facsimile No. **(86-10)62019451**	Telephone No. **(86-10)82245426**

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	US 2010332769 A1 (INTERNATIONAL BUSINESS MACHINES CORPORATION) 30 December 2010 (2010-12-30) the whole document	1-20

INTERNATIONAL SEARCH REPORT
Information on patent family members

International application No.
PCT/CN2014/076594

Patent document cited in search report			Publication date (day/month/year)	Patent family member(s)			Publication date (day/month/year)
WO	2013048826	A1	04 April 2013	US	2014344529	A1	20 November 2014
				CN	103842986	A	04 June 2014
				US	2013086333	A1	04 April 2013
				KR	20140068909	A	09 June 2014
				JP	2014528609	A	27 October 2014
US	2010088702	A1	08 April 2010		Non	e	
US	7594234	B1	22 September 2009	US	2009328053	A1	31 December 2009
CN	101256509	A	03 September 2008		Non	e	
US	2010332769	A1	30 December 2010		Non	e	