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

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
4 September 2014 (04.09.2014)

WIPO | PCT

(10) International Publication Number
WO 2014/133595 A1

(51) International Patent Classification:
G06F 9/52 (2006.01) *G06F 9/54* (2006.01)

(21) International Application Number:
PCT/US2013/067108

(22) International Filing Date:
28 October 2013 (28.10.2013)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
13/781,493 28 February 2013 (28.02.2013) US

(71) Applicant: **ORACLE INTERNATIONAL CORPORATION** [US/US]; 500 Oracle Parkway, M/S 5op7, Redwood Shores, California 94065 (US).

(72) Inventor: **OTENKO, Oleksandr**; 24 Bathurst Road, Winnersh Berkshire RG41 5JB (GB).

(74) Agents: **MEYER, Sheldon R.** et al.; Fliesler Meyer LLP, 410 Pacific Avenue, San Francisco, California 94133 (US).

(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).

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

(54) Title: SYSTEM AND METHOD FOR USING A SEQUENCER IN A CONCURRENT PRIORITY QUEUE

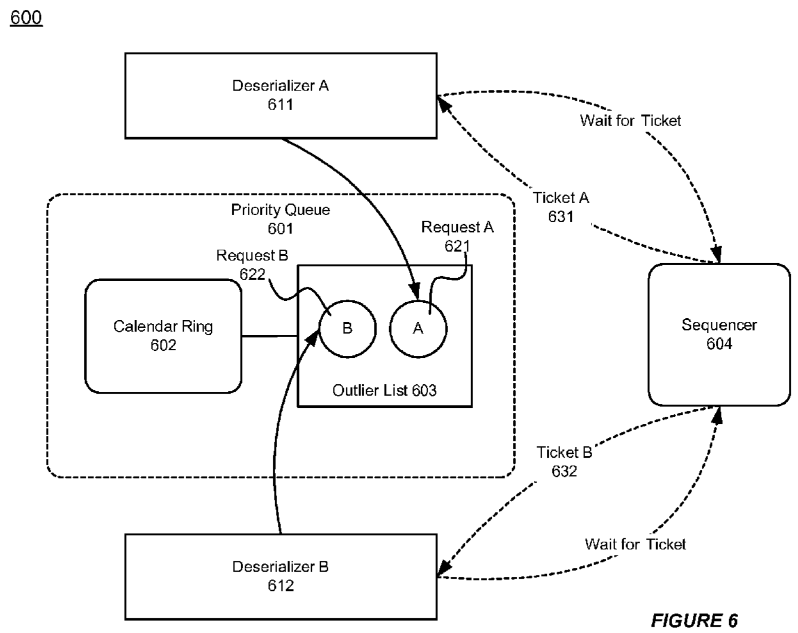

FIGURE 6

(57) **Abstract**: A system and method can support a concurrent priority queue. The concurrent priority queue allows a plurality of threads to interact with the priority queue. The priority queue can use a sequencer to detect and order a plurality of threads that contend for one or more requests in the priority queue. Furthermore, the priority queue operates to reduce the contention among the plurality of threads.

SYSTEM AND METHOD FOR USING A SEQUENCER IN A CONCURRENT PRIORITY QUEUE

Copyright Notice:

Field of Invention:

[0002] The present invention is generally related to computer systems and software such as middleware, and is particularly related to systems and methods for supporting queue in a middleware machine environment.

Background:

[0003] Within any large organization, over the span of many years, the organization often finds itself with a sprawling IT infrastructure that encompasses a variety of different computer hardware, operating systems, and application software. Although each individual component of such infrastructure might itself be well-engineered and well-maintained, when attempts are made to interconnect such components, or to share common resources, it is often a difficult administrative task. In recent years, organizations have turned their attention to technologies such as virtualization and centralized storage, and even more recently cloud computing, which can provide the basis for a shared infrastructure. However, there are few all-in-one platforms that are particularly suited for use in such environments. These are the general areas that embodiments of the invention are intended to address.

Summary:

[0004] Systems and methods are provided for supporting a concurrent priority queue. The concurrent priority queue allows a plurality of threads to interact with the priority queue. The priority queue can use a sequencer to detect and order a plurality of threads that contend for one or more requests in the priority queue. Furthermore, the priority queue operates to reduce the contention among the plurality of threads.

[0005] Other objects and advantages of the present invention will become apparent to those skilled in the art from the following detailed description of the various embodiments, when read in light of the accompanying drawings.

Brief Description of the Figures:

[0006] **Figure 1** shows an illustration of a middleware machine environment 100, in accordance with an embodiment of the invention.

[0007] **Figure 2** shows another illustration of a middleware machine platform or environment, in accordance with an embodiment of the invention.

[0008] **Figure 3** shows an illustration of using a priority queue for handling requests in a middleware machine environment, in accordance with various embodiments of the invention.

[0009] **Figure 4** shows an illustration of supporting a non-blocking queue in a middleware machine environment.

[0010] **Figure 5** shows an illustration of supporting a concurrent priority queue in accordance with various embodiments of the invention.

[0011] **Figure 6** shows an illustration of guaranteeing a first-in-first-out (FIFO) order for the outlier list in accordance with various embodiments of the invention.

[0012] **Figure 7** illustrates an exemplary interactive chart for adding multiple requests into an outlier list in a concurrent priority queue in accordance with an embodiment of the invention.

[0013] **Figure 8** shows an illustration of detecting contention among different consumers of a concurrent priority queue in accordance with various embodiments of the invention.

[0014] **Figure 9** is an exemplary interactive chart for illustrating interaction between a victim and a contender when a contention is detected in accordance with an embodiment of the invention.

[0015] **Figure 10** is an exemplary interactive chart for illustrating interaction between a victim and a contender when no contention is detected in accordance with an embodiment of the invention.

[0016] **Figure 11** illustrates an exemplary flow chart for supporting cooperative concurrency in a priority queue in accordance with an embodiment of the invention.

Detailed Description:

[0017] Described herein are systems and methods that can support work sharing muxing in a cluster.

[0018] **Figure 1** shows an illustration of a middleware machine environment 100, in accordance with an embodiment of the invention. As shown in Figure 1, each middleware machine system 102 includes several middleware machine rack components 104, each of which includes a combination of high-performance middleware machine hardware nodes 106 (e.g., 64-bit processors, high performance large memory, and redundant InfiniBand and Ethernet networking), and a middleware machine software environment 108. The result is a complete application server environment which can be provisioned in minutes rather than days or months, and which can scale on demand. In accordance with an embodiment, each middleware machine system can be deployed as a full, half, or quarter rack, or other configuration of rack components, and several middleware machine systems can be coupled together, again using InfiniBand, to create larger environments. Each middleware machine software environment can be provisioned with several application server or other software instances. For example as shown in Figure 1, an application server instance 109 could comprise a virtual machine 116, operating system 120, virtualization layer 124, and application server layer 128 (e.g. WebLogic, including servlet 132, EJB 134, and Gridlink 136 containers). Another application server instance 110 could comprise a virtual machine 118, operating system 122, virtualization layer 126, and data grid layer 140 (e.g. Coherence, including an active cache 142). Each of the instances can communicate with one another, and with both its middleware machine hardware node, and other nodes, using a middleware machine integration component 150, such as an ExaLogic integration pack, which itself provides several optimization features, such as support for InfiniBand and other features, as described in further detail below.

[0019] **Figure 2** shows another illustration of a middleware machine platform or environment, in accordance with an embodiment of the invention. As shown in Figure 2, each application server instance can act as a sender and/or receiver 160, 161 within the middleware machine environment. Each application server instance is also associated with a muxer 162, 163, that allows the application servers to communicate with one another via an InfiniBand network 164. In the example shown in Figure 2, an application server instance can include a kernel space 165, user space 167, and application server (e.g. WebLogic space) 166, which in turn can be associated with a sockets direct protocol 168, a JVM (e.g. JRockit/Hotspot layer) 170, a WLS core 172, a servlet container 174, and a JSP compiler 176. In accordance with other examples, other combinations of middleware-type software can be included. In accordance with various embodiments, the machine integration component can provide features 180 such as Zero Buffer Copies, Scatter/Gather I/O, T3 Connections, Lazy Deserialization, and GridLink DataSource, to provide the basis for, and improve performance within, the shared infrastructure.

Priority Queue

[0020] In accordance with various embodiments of the invention, a concurrent system can use a priority queue to prioritize incoming requests in order to provide service with an appropriate service level agreement (SLA).

[0021] **Figure 3** shows an illustration of using a priority queue for handling requests in a middleware machine environment, in accordance with various embodiments of the invention. As shown in Figure 3, one or more threads, e.g. deserializing threads A-B 311-312, can deserialize the incoming network traffic 310 that contains one or more requests 320. The deserializing threads A-B 311-312 can place the requests 320 in a priority queue 301, e.g. using add() methods. Then, a plurality of worker threads, e.g. worker threads A-C 321-323, can access the priority queue 301 concurrently and can claim the requests 320, e.g. using delete_min() methods.

[0022] The priority queue 301 can be designed to meet demanding concurrency criteria, so that the interaction between the contenders does not cause degradation in the throughput of the system as a whole. Additionally, the priority queue 301 can be implemented to have a fixed memory footprint, so that the JVM is able to better optimize its operations on fixed-size arrays of primitives, and can achieve substantial cache efficiency.

[0023] In accordance with various embodiments of the invention, the priority queue 301 can be implemented based on a calendar queue, e.g. the calendar queue provided in the WebLogic Application Server. The calendar queue can include a calendar with multiple buckets, each of which can store events that fall within a particular slice of time. For example, the multiple buckets can be sorted and arranged by comparing the target service time with a current time. If the difference in time is in the first byte, then the request can be stored in a bucket in the first 256 buckets. The specific bucket can be chosen using the actual value of the target time for executing the request. Furthermore, if the difference in time is in the second byte, then the request can be stored in a bucket in the second 256 buckets.

[0024] When a consumer, e.g. via one of the worker threads A-C 321-323, tries to remove the next request that is configured to be execute the earliest, the system can scan the calendar for the first bucket that is not empty. If this bucket is not one of the first 256 buckets, then the calendar queue can use a loop and promote method to move the requests to the buckets "one level down" toward the first 256 buckets. Eventually, some requests can be promoted to one or more buckets in the first 256 buckets, and the consumer can claim a request and proceed accordingly.

[0025] The above promotion process can involve logarithmic cost, which may have an impact on the overall performance of the system. Additionally, there can be other designs for the calendar queue, the performance of which may be limited to a choice between “O(1) add, O(logN) delete_min,” and “O(logN) add, O(1) delete_min.”

[0026] **Figure 4** shows an illustration of supporting a non-blocking queue in a middleware machine environment. As shown in Figure 4, a plurality of consumers, e.g. consumers A-B 411-412, can concurrently access a priority queue 401 in a middleware machine environment 400. The priority queue 401 can be implemented as a non-blocking queue and can be accessed via a request manager 402.

[0027] The request manager 402, which manages a thread pool 403, can have a separate logic for associating different threads with different requests. For example, the request manager 402 can serialize all thread pool method calls by wrapping the calls to the priority queue 401 in a synchronized statement, or a synchronized block 410, using a lock mechanism.

[0028] Thus, the operations on the priority queue 401 may be limited by the single-threaded design since the serialization is done outside the non-blocking priority queue 401.

Concurrent Priority Queue

[0029] **Figure 5** shows an illustration of supporting a concurrent priority queue in accordance with various embodiments of the invention. As shown in Figure 5, a plurality of consumers, e.g. consumer A-C 511-513 can concurrently access a concurrent priority queue 501 in a middleware machine environment 500.

[0030] The concurrent priority queue 501 can include a calendar, e.g. a calendar ring 502, which is capable of prioritizing and storing incoming requests. The calendar ring 502, the size of which is limited, can be configured to store requests that have a target response time within a preconfigured time limit. Within the calendar ring 502, a request can be stored, or placed, directly in the ring buffer at a position that matches Quality of Service (QoS) of the request, e.g. the target service time.

[0031] Thus, the system can achieve a much cheaper lookup for requests without changing the memory footprint of a calendar queue. Furthermore, the system can reduce the logarithmic complexity of the delete_min operation of the calendar queue to mostly a linear cache efficient search, while keeping the adding of elements to the calendar queue as O(1) operations.

[0032] Additionally, a request with a target service time higher than the preconfigured time limit can be added to a list of outliers, e.g. the outlier list 504. Since the scheduling of these requests may not be time critical, the system permits the slower addition to a sorted list of outliers 504. Furthermore, the concurrent priority queue 501 can use a sequencer, e.g. outliers_seq, to enforce a first-in-first-out (FIFO) order for the outlier list with the same QoS.

[0033] For example, the calendar ring 502 can be configured to store requests with a target response time (or QoS) below 2 seconds, since the requests with the QoS higher than 2 seconds can be considered rare. Furthermore, the requests with the QoS below 2 seconds can be placed in the calendar ring 502 that matches QoS, while the requests with the QoS higher than 2 seconds can be placed into the list of outliers 504.

[0034] Unlike the calendar queue as shown in Figure 4, the request manager 510 does not need to put every call to the calendar queue 501 in a synchronized statement. The synchronized block 506, which supports continuation-passing 507, can be implemented within the scope of the concurrent priority queue 501. The consumers, e.g. consumers A-C 511-513, may not need to access the thread pool 520 from outside the concurrent priority queue 501.

[0035] Using continuation-passing, the system can transform the calendar queue 501 from non-blocking to blocking. The continuation-passing 507 can enable the consumers A-C 511-513 to manage the idle workers, or Threads 530, in the thread pool 520, so that the threads 530, which may be waiting in the thread pool 520, can be reused.

[0036] Additionally, the concurrent priority queue 501 can include a sequencer 503 that enables the concurrent priority queue 501 to detect contention and can use a fast lane 505 to support cooperative concurrency. Thus, the concurrent priority queue 501 can be aware of and handle the contention properly, without a need for the locks to expose knowledge about contention.

Maintain FIFO for the Outlier List

[0037] **Figure 6** shows an illustration of guaranteeing a first-in-first-out (FIFO) order for the outlier list in accordance with various embodiments of the invention. As shown in Figure 6, a priority queue 601 in a middleware machine environment 600 can include a calendar ring 602 and an outlier list 603.

[0038] A plurality of callers, e.g. deserializers A-B 611-612 on different threads, may try to access the priority queue 601 concurrently. The priority queue 601 can use a sequencer 604, which is based on a ticket mechanism, to guarantee the first-in-first-out (FIFO) order for the outlier list 603.

[0039] For example, before a caller, e.g. deserializer A 611, is allowed to add a request A 621 into the outlier list 603, the deserializer A 611 can first send a message to the sequencer 604 requesting a ticket. The sequencer 604 can issue a ticket, e.g. ticket A 631, to the deserializer A 611 if there is no contention.

[0040] Furthermore, another caller, e.g. deserializer B 612, may be trying to add another request, e.g. request B 622, into the outlier list 603. The sequencer 604 may receive another request for a ticket from the deserializer B 612. Since, at this time, the deserializer A 611 is adding the request A 621 into the outlier list 603, the sequencer 604 can block the deserializer B 612.

[0041] After the deserializer A 611 finishes adding the request A 621 into the outlier list 603, the deserializer A 611 can advance the sequencer 604. Then, the sequencer 604 can issue a ticket, e.g. ticket B 632, to the deserializer B 612, which allows the deserializer B 612 to add the request B 622 into the outlier list 603 after the request A 621.

[0042] **Figure 7** illustrates an exemplary interactive chart for adding multiple requests into an outlier list in a concurrent priority queue in accordance with an embodiment of the invention. As shown in Figure 7, a sequencer 710 can coordinate multiple callers or threads, e.g. deserializers A-B 711-722, which may try to access a priority queue concurrently.

[0043] At step 701, the deserializer A 711 can try to obtain a ticket from the sequencer 710. Then, at step 702, the sequencer 710 can check with the ticket mechanism. If this succeeds, then there is no other caller that is trying to access the priority queue. Otherwise, the deserializer A 711 may be blocked until someone releases it.

[0044] Then, at step 703, another caller, the deserializer B 712, can successfully add a request into the outlier list, while the deserializer A 711 is blocked. At step 704, the deserializer B can try to advance the sequencer 710.

[0045] At step 705, after the sequencer 710 receives the message to advance, the sequencer 710 can create and issue a ticket to the deserializer A 711, which is waiting. Consequently, at step 706, the deserializer A 711 receives the ticket, and, at step 707, the deserializer A 711 can proceed to add another request into the outlier list.

[0046] Thus, using the sequencer 710, the FIFO order of the outlier list can be preserved without a need for implementing a synchronization or lock mechanism.

Detect and Handle Contention among Different Consumers

[0047] **Figure 8** shows an illustration of detecting contention among different consumers of a concurrent priority queue in accordance with various embodiments of the invention. As shown in Figure 8, different consumers, e.g. consumers A-B 811-812, can concurrently access a priority queue 801 in the middleware machine environment 800. The priority queue 801 can include a calendar ring 802, an outlier list 803, and a fast lane 804.

[0048] Each consumer can take advantage of a list of stolen requests to further reduce concurrency. For example, consumer A 811 can use a list of stolen requests 807, which can appear as a local list to the consumer A 811. Additionally, each of the consumers A-B 811-812 can send a message to a sequencer 805, requesting for a ticket before it is allowed for accessing the priority queue 801.

[0049] The sequencer 805 can maintain a reader count 806, which is the current count of the total number of readers that have requested for a ticket from the sequencer 805. The sequencer 805 can increase the reader count 806 every time when it receives a ticket request. Furthermore, this reader count 806 can be used to detect contention.

[0050] For example, consumer A 811 can obtain a ticket, e.g. ticket A 831, from the sequencer 805, before it is allowed for accessing the priority queue 801. Then, another consumer B 812 can send a request for a ticket, e.g. ticket B 832, to the sequencer 805. The sequencer 805 can increase the reader count 806 and block the consumer B 812 until consumer A 811 finishes its operation.

[0051] On the other hand, consumer A 811 can detect a contention from consumer B 812, when it detects that the current reader count 806, t, at the sequencer 805 exceeds the ticket number that it holds (t>ticket number). Then, the consumer A 811 can reduce the contention by placing a request into the fast lane 804 and allowing it to be consumed by consumer B 812, which can be referred as a cooperative concurrency strategy.

[0052] Thus, using the sequencer 805, the priority queue can be optimized for handling contention among multiple consumers by allowing consumers to access the fast lane 804 and the list of stolen requests 807 in addition to the calendar ring 802, without a need for a lock or synchronization mechanism.

[0053] **Figure 9** is an exemplary interactive chart for illustrating interaction between a victim and a contender when a contention is detected in accordance with an embodiment of the invention. As shown in Figure 9, a sequencer 920 can be used to detect and handle contentions among different consumers of a concurrent priority queue, e.g. a victim 922 and a contender 921.

[0054] At step 901, the contender 921 can send a message to the sequencer 920 indicating that it is waiting for a ticket. Since the victim 922 is currently accessing the priority queue, at step 902, the sequencer 920 can block the contender 921 and increase the reader count, readers.

[0055] At steps 903-905, the victim 922 can pick up a request from the concurrent priority queue, and check with the sequencer 920 for the current readers. (The steps 903-904 can also be referred to as a non-temporal transition 913, which will be discussed in a later section.)

[0056] Then, at step 906, the victim 922 can compare the obtained reader count, t, with the ticket number, ticket number, that it holds. Since the sequencer 920 have already increased the reader count, the victim 922 can detect the contention when it finds out t > ticket number.

[0057] Subsequently, at steps 907-908, the victim 922 can place the request into the fast lane and update the request count, fastLane_w, in the fast lane, before the victim 922 tries to advance the reader sequencer, at step 909. (The steps 907-908 can also be referred to as a non-temporal transition 914, which will be discussed in a later section.)

[0058] After receiving the message from the victim 922 to advance the reader sequencer, at step 910, the sequencer 920 can proceed to issue a ticket to the contender 921, which releases the the contender 921.

[0059] Eventually, at step 911, the contender 921 can receive the ticket and, at step 912, the contender 921 can proceed to claim a request from the fast lane.

[0060] **Figure 10** is an exemplary interactive chart for illustrating interaction between a victim and a contender when no contention is detected in accordance with an embodiment of the invention. As shown in Figure 10, a victim 1022 may not always be able to detect contention from a contender 1021, due to the concurrent nature of accessing the priority queue.

[0061] At steps 1001-1003, the victim 1022 can pick up a request from the concurrent priority queue, and the victim 1022 can check with the sequencer 1020 for the current reader count, readers. (The steps 1001-1002 can also be referred to as a non-temporal transition 1011, which will be discussed in a later section.)

[0062] Furthermore, at step 1004, the sequencer 1020 may receive a request for a ticket from the contender 1021, and, at step 1005, the sequencer 1020 can increase the reader count, readers.

[0063] At step 1006, the victim 1022 may not be able to detect the contention from a contender 1021, since the sequencer 1020 increased the reader count after the contender 1021 returned the current readers to the victim 1022.

[0064] At step 1007, the victim 1022 can claim the request and advance the the reader sequencer, which in turn, at step 1008, can issue a ticket to the contender 1021.

[0065] Then, at step 1009, the contender 1021 can receive the ticket and, at step 1010, the contender 1021 can proceed to claim a request from the priority queue, e.g. from the fast lane.

[0066] **Figure 11** illustrates an exemplary flow chart for supporting cooperative concurrency in a priority queue in accordance with an embodiment of the invention. As shown in Figure 11, at step 1101, the system allows a plurality of threads to interact with the concurrent priority queue. Then, at step 1102, the system can use a sequencer to detect a plurality threads that contend for one or more requests in the priority queue. Furthermore, at step 1103, the system can reduce the contention among the plurality threads in the priority queue.

Temporal and Non-temporal Transitions

[0067] In accordance with an embodiment of the invention, a concurrent programming model can support different kinds of transitions: such as temporal transitions and non-temporal transitions.

[0068] The non-temporal transition steps, which include the usual program order steps, can be freely reordered by the compiler and processor. For example, Figure 9 includes non-temporal transition steps 913 and 914, and Figure 10 includes non-temporal transition step 1011. Within each of these non-temporal transition steps, it can be ensured that reordering does not break the logic intended by program order.

[0069] On the other hand, there are restrictions on reordering for the temporal transitions, which may also appear in program order. The temporal transitions can include deliberate design choices in concurrent algorithms, and the temporal transitions can be implemented as compiler and processor barriers. For example, a temporal transition in delete_min can indicate that any data loads, e.g. reading fastLane_w, cannot go ahead of data loads, e.g. the load of fastLane_r.

[0070] The use of temporal transitions and non-temporal transitions allows the caller to collect information about the progress of other parts of the concurrent algorithm in order to make consistent mutation of the states.

[0071] In the example as shown in Figure 9, after the victim 922 detects a contender at step 906, the readers can change when more contenders arrives. Even though the victim 922 may not know the exact value of readers, it remains true that the reader count may only increases. Thus, it is safe to place the request in the fast lane ring.

[0072] On the other hand, in the example as shown in Figure 10, after the victim 1022 determines that there is no contender (readers == ticket number) at step 1006, the true value of readers can actually be higher than what is returned previously, since the contender 1021 arrives. It can be assumed that such occurrence is rare, and the cost for determining the exact value of readers is higher than the gain from knowing it. Additionally, the cost of having a wrong guess here can be considered low, since the victim 1022 proceeds almost immediately to advance the reader sequencer at step 1007, which can unblock any contenders fairly quickly.

[0073] Furthermore, in the example as shown in Figure 8, the adding of a request, e.g. request A 821, into the fast lane 804 by consumer A 811 can be implemented as non-temporal transitions, while the claiming of the request A 821 from the fast lane 804 by consumer B 812 can be implemented as temporal transitions.

[0074] In this example, the contender, consumer B 812, may observe the value of fastLane_w before the requests are actually stored to the fast lane ring 804. Here, only one contender is allowed to access the fast lane 804, after the victim, consumer A 811, advances the readers sequencer. It follows that there can be no more contenders accessing the fast lane 804 than the number of requests that can be observed.

[0075] Thus, even though the contenders 811 may observe that the request count for the fast lane 804, fastLane_w, is ahead of the actual filled part of the fast lane 804, the contenders can only access the values using an index, fastLane_r, which is updated only once by each contender. In other words, the victim, consumer A 811, can control the accessibility of requests in fast lane 804 by controlling the maximum number of contenders that may be released, since the value of fastLane_r can reach fastLane_w, only when enough contenders are released by the victim 812.

[0076] In some embodiments, a computer program for implementing one of the above-mentioned methods is provided. In accordance with an embodiment of the invention, the computer program causes a system to perform the steps comprising: allowing a plurality of threads to interact with the concurrent priority queue; detecting, via a sequencer, a plurality of threads that contend for one or more requests in the priority queue, reducing, via the priority queue, the contention among the plurality of threads.

[0077] In one embodiment, there is provided for a system for supporting a concurrent priority queue. The system comprises means for allowing a plurality of threads to interact with the concurrent priority queue, means for detecting, via a sequencer, a plurality of threads that contend for one or more requests in the priority queue, and means for reducing, via the priority queue, the contention among the plurality of threads.

[0078] In one embodiment, the system further comprises means for containing a calendar ring in the priority queue, wherein the calendar ring operates to store various requests with a target response time less than a pre-configured quality of service (QoS).

[0079] In one embodiment, the system further comprises means for specifying the pre-configured quality of service (QoS) to be about two seconds.

[0080] In one embodiment, the system further comprises means for using a bit map updated in synchronization with the calendar ring for faster scanning.

[0081] In one embodiment, the system further comprises means for Associating, with the priority queue, a list of stolen requests that can be used to reduce contention on the calendar ring when workload is intensive, wherein the list of stolen requests contains one or more requests that land on a same calendar entry.

[0082] In one embodiment, the system further comprises means for containing a fast lane in the priority queue that can be used to support cooperative concurrency.

[0083] In one embodiment, the system further comprises means for using, via the sequencer, a ticket mechanism that can order the plurality of threads.

[0084] In one embodiment, the system further comprises means for keeping, via the ticket mechanism, a reader count that is increased everytime when a consumer try to access the priority queue.

[0085] In one embodiment, the system further comprises means for issuing, via the sequencer, a ticket to a contending thread, wherein the ticket allows the contending thread to access the priority queue.

[0086] 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.

[0087] 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.

[0088] 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 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 system for supporting a concurrent priority queue, comprising:

one or more microprocessors;

a priority queue that allows a plurality of threads to interact with the priority queue; and

a sequencer, running on the one or more microprocessors, wherein the sequencer operates to detect a plurality of threads that contend for one or more requests in the priority queue, and

wherein the priority queue operates to reduce the contention among the plurality of threads.

2. The system according to Claim 1, wherein:

the priority queue contains a calendar ring, wherein the calendar ring operates to store various requests with a target response time less than a pre-configured quality of service (QoS).

3. The system according to Claim 2, wherein:

the pre-configured quality of service (QoS) is specified to be about two seconds.

4. The system according to Claim 2 or 3, wherein:

a bit map updated in synchronization with the calendar ring is used for faster scanning.

5. The system according to any of Claims 2 to 4, wherein:

the priority queue contains a list of stolen requests that can be used to reduce contention on the calendar ring when workload is intensive, wherein the list of stolen requests contains one or more requests that land on a same calendar entry.

6. The system according to any of Claims 2 to 5, wherein:

the priority queue contains a fast lane that is used to support cooperative concurrency.

7. The system according to any preceding Claim, wherein:

the sequencer uses a ticket mechanism that can order the plurality of threads.

8. The system according to Claim 7, wherein:

the ticket mechanism keeps a reader count that is increased every time when a consumer tries to access the priority queue.

9. The system according to Claim 7 or 8, wherein:
the sequencer operates to issue a ticket to a contending thread, wherein the ticket allows the contending thread to access the priority queue.

10. The system according to any preceding Claim, wherein:
the sequencer operates to preserve a first-in-first-out (FIFO) order of a outlier list in the priority queue.

11. A method for supporting a concurrent priority queue, comprising:
allowing a plurality of threads to interact with the concurrent priority queue;
detecting, via a sequencer, a plurality of threads that contend for one or more requests in the priority queue,
reducing, via the priority queue, the contention among the plurality of threads.

12. The method according to Claim 11, further comprising:
containing a calendar ring in the priority queue, wherein the calendar ring operates to store various requests with a target response time less than a pre-configured quality of service (QoS).

13. The method according to Claim 12, further comprising:
specifying the pre-configured quality of service (QoS) to be about two seconds.

14. The method according to Claim 12 or 13, further comprising:
using a bit map updated in synchronization with the calendar ring for faster scanning.

15. The method according to any of Claims 12 to 14, further comprising:
associating, with the priority queue, a list of stolen requests that can be used to reduce contention on the calendar ring when workload is intensive, wherein the list of stolen requests contains one or more requests that land on a same calendar entry.

16. The method according to any of Claims 12 to 15, further comprising:
containing a fast lane in the priority queue that is used to support cooperative concurrency.

17. The method according to any of Claims 11 to 16, further comprising:
using, via the sequencer, a ticket mechanism that can order the plurality of threads.

18. The method according to Claim 17, further comprising:

keeping, via the ticket mechanism, a reader count that is increased every time when a consumer tries to access the priority queue.

19. The method according to Claim 17 or 18, further comprising:

issuing, via the sequencer, a ticket to a contending thread, wherein the ticket allows the contending thread to access the priority queue.

20. A computer program comprising machine-readable instructions that when executed cause a system to perform the method according to any of Claims 11 to 19.

21. A computer program product comprising a machine readable storage medium storing the computer program of claim 20.

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

allowing a plurality of threads to interact with the concurrent priority queue;

detecting, via a sequencer, a plurality of threads that contend for one or more requests in the priority queue,

reducing, via the priority queue, the contention among the plurality of threads.

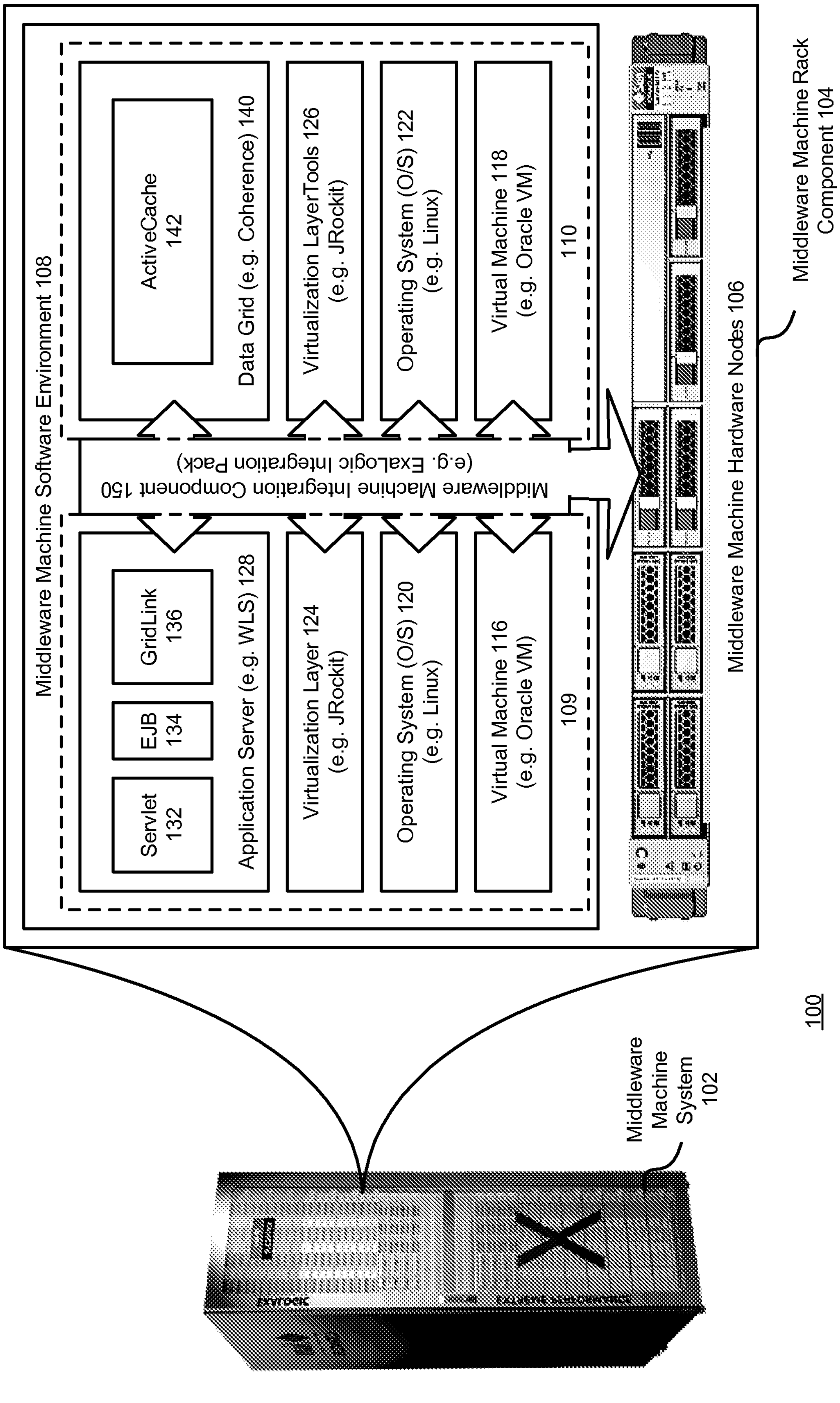
Middleware Machine Software Environment 108
Servlet 132
EJB 134
GridLink 136
Application Server (e.g. WLS) 128
Virtualization Layer 124 (e.g. JRockit)
Operating System (O/S) 120 (e.g. Linux)
Virtual Machine 116 (e.g. Oracle VM)
109
ActiveCache 142
Data Grid (e.g. Coherence) 140
Virtualization LayerTools 126 (e.g. JRockit)
Operating System (O/S) 122 (e.g. Linux)
Virtual Machine 118 (e.g. Oracle VM)
110
Middleware Machine Integration Component 150 (e.g. Exalogic Integration Pack)
Middleware Machine Hardware Nodes 106
Middleware Machine Rack Component 104
Middleware Machine System 102
100

FIGURE 1

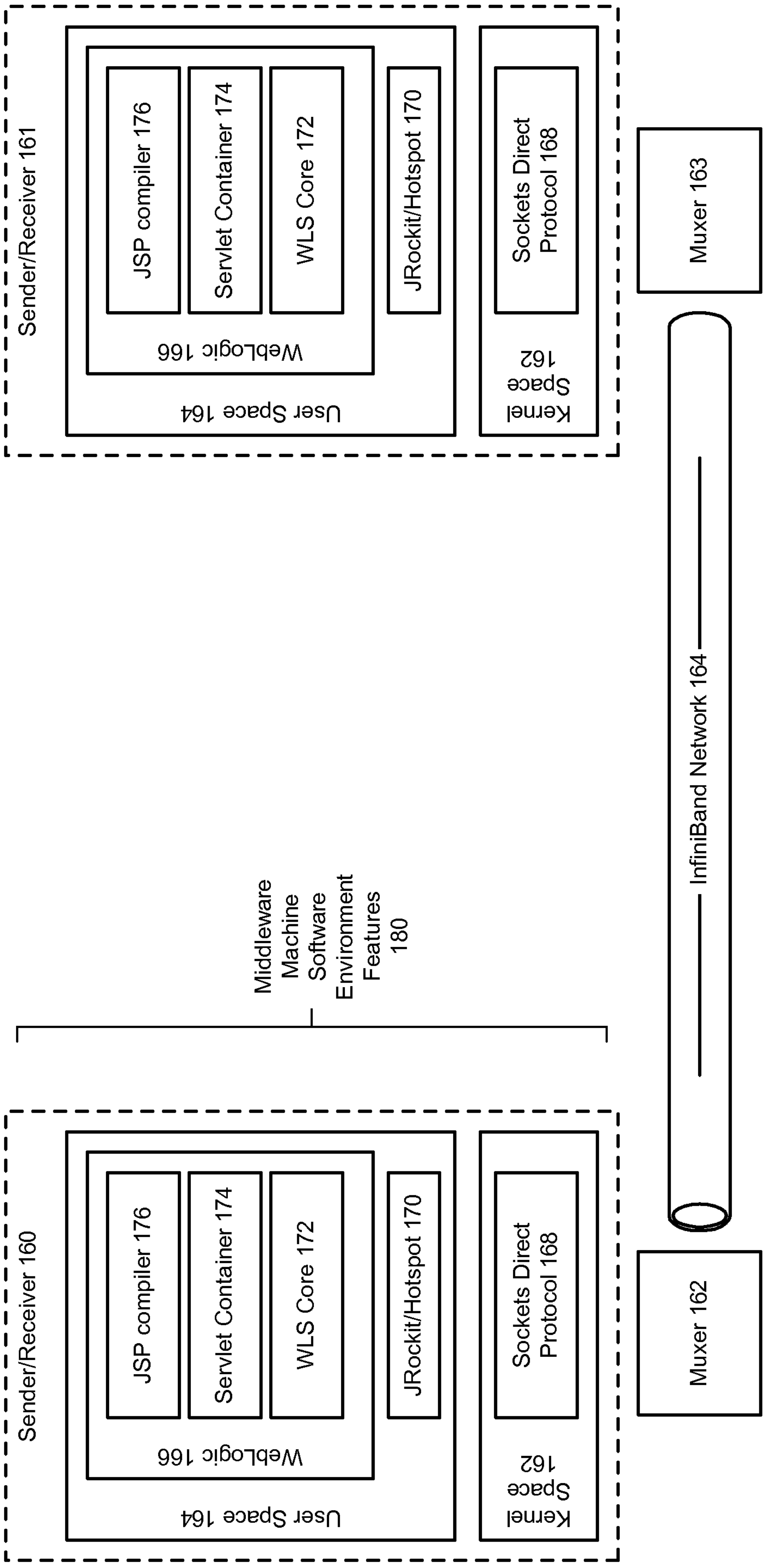
Sender/Receiver 160
JSP compiler 176
Servlet Container 174
WLS Core 172
WebLogic 166
JRockit/Hotspot 170
User Space 164
Sockets Direct Protocol 168
Kernel Space 162
Muxer 162
Middleware Machine Software Environment Features 180
InfiniBand Network 164
Sender/Receiver 161
JSP compiler 176
Servlet Container 174
WLS Core 172
WebLogic 166
JRockit/Hotspot 170
User Space 164
Sockets Direct Protocol 168
Kernel Space 162
Muxer 163

FIGURE 2

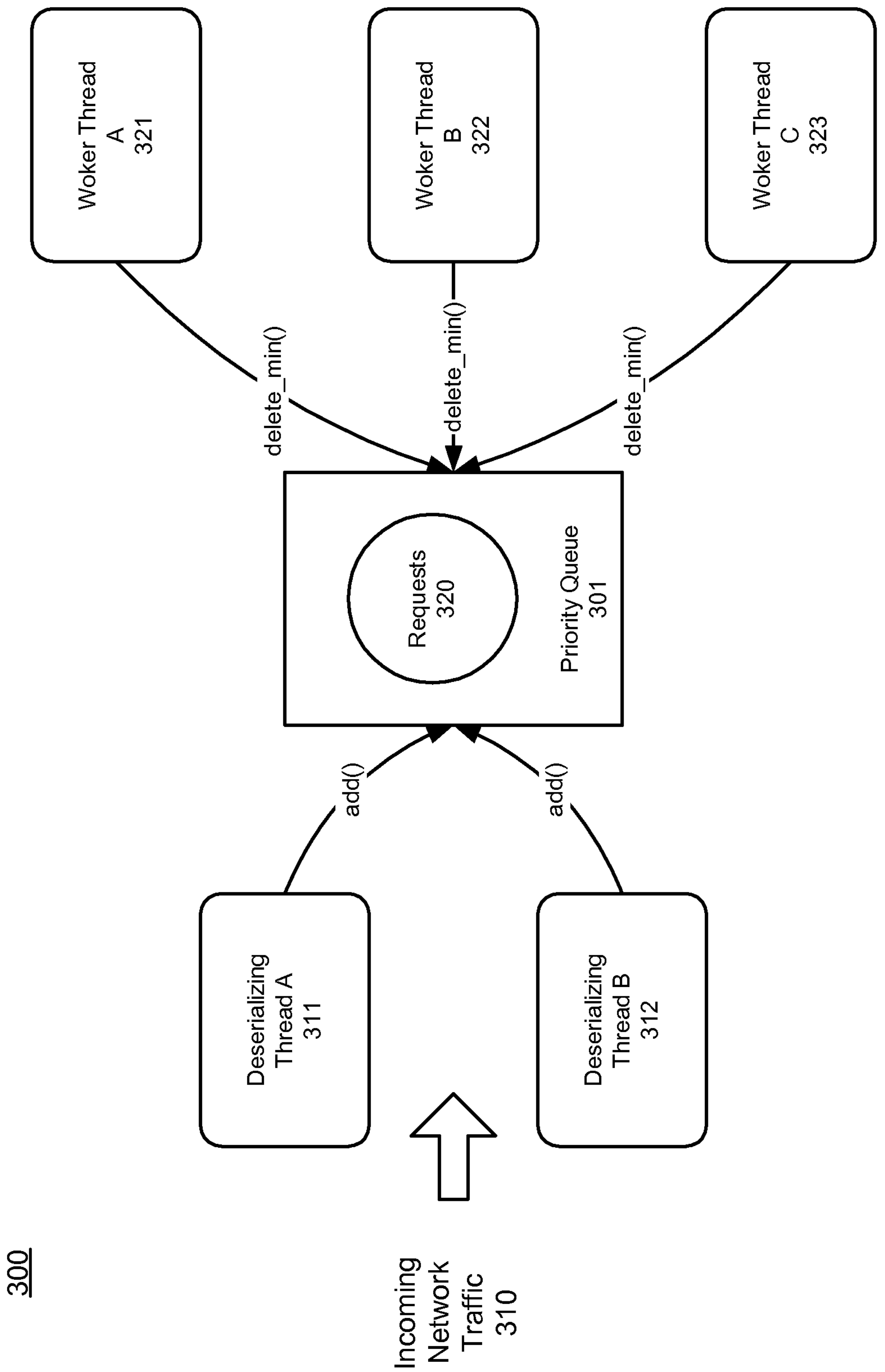
300
Incoming Network Traffic 310
Deserializing Thread A 311
Deserializing Thread B 312
add()
add()
Requests 320
Priority Queue 301
delete_min()
delete_min()
delete_min()
Woker Thread A 321
Woker Thread B 322
Woker Thread C 323

FIGURE 3

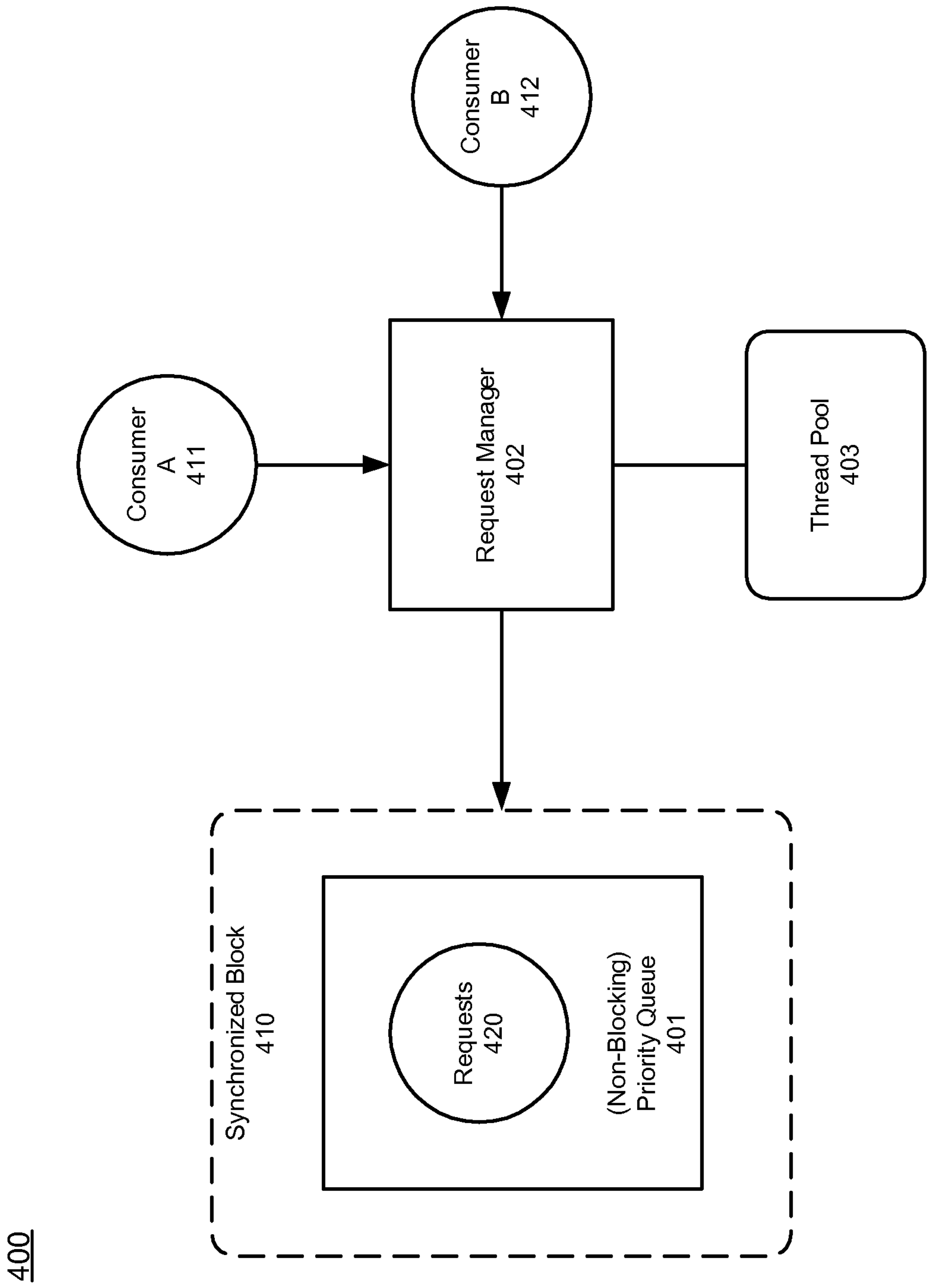
400
Consumer A 411
Consumer B 412
Request Manager 402
Thread Pool 403
Synchronized Block 410
Requests 420
(Non-Blocking) Priority Queue 401

FIGURE 4

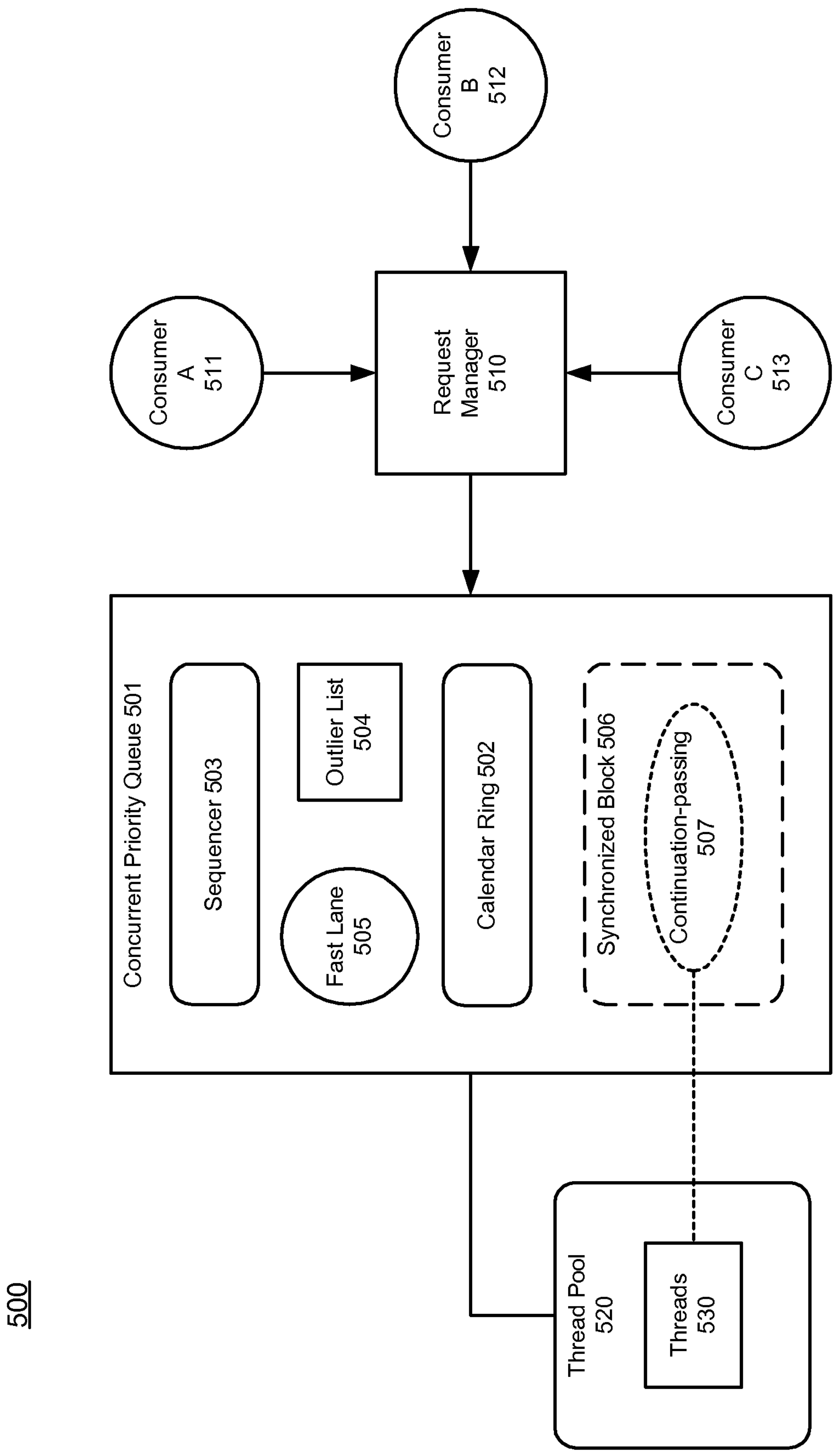
500
Consumer B 512
Consumer A 511
Request Manager 510
Consumer C 513
Concurrent Priority Queue 501
Sequencer 503
Outlier List 504
Fast Lane 505
Calendar Ring 502
Synchronized Block 506
Continuation-passing 507
Thread Pool 520
Threads 530

FIGURE 5

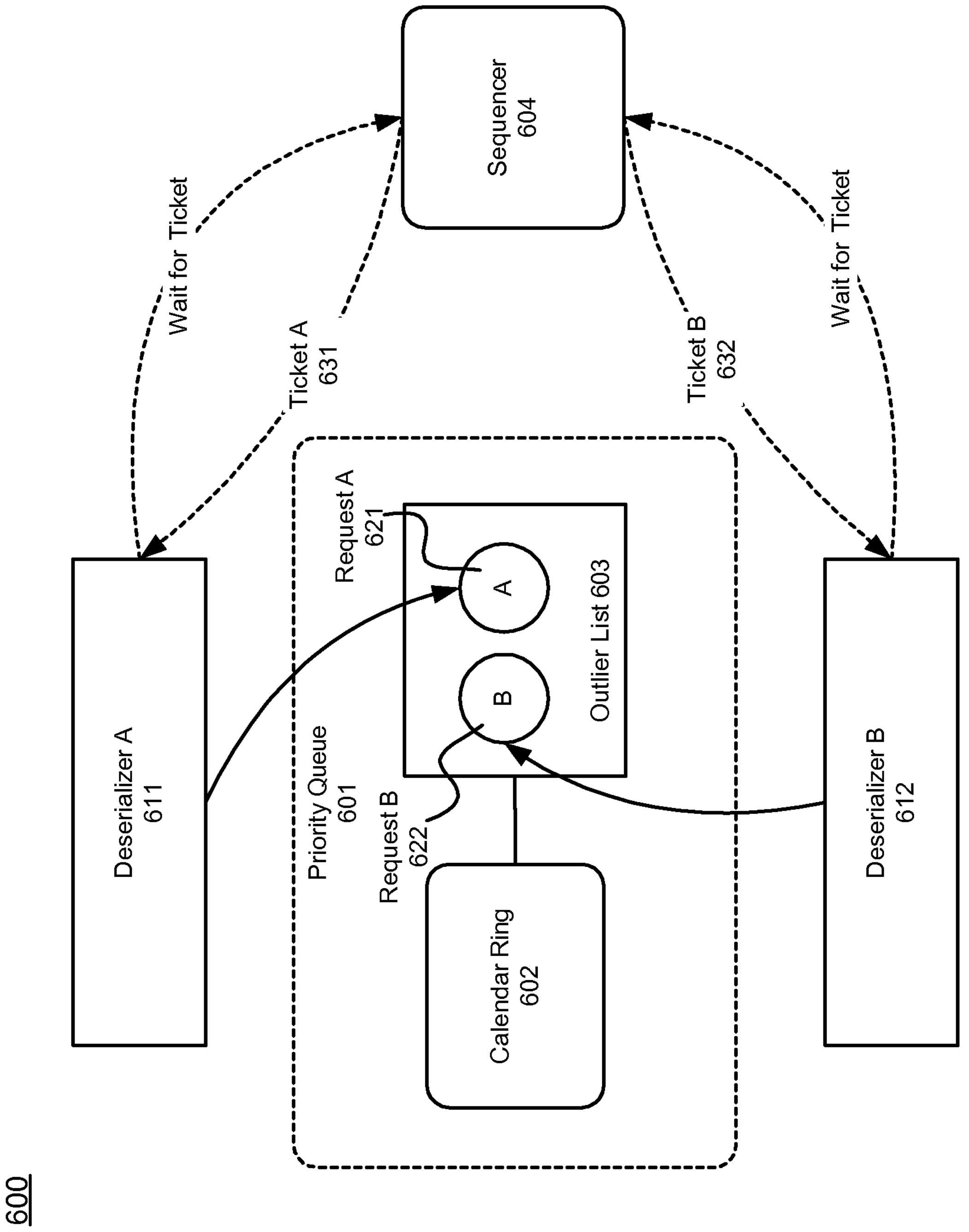
600
Sequencer
604
Wait for Ticket
Wait for Ticket
Ticket A
631
Ticket B
632
Deserializer A
611
Deserializer B
612
Priority Queue
601
Request A
621
Request B
622
A
B
Outlier List 603
Calendar Ring
602

FIGURE 6

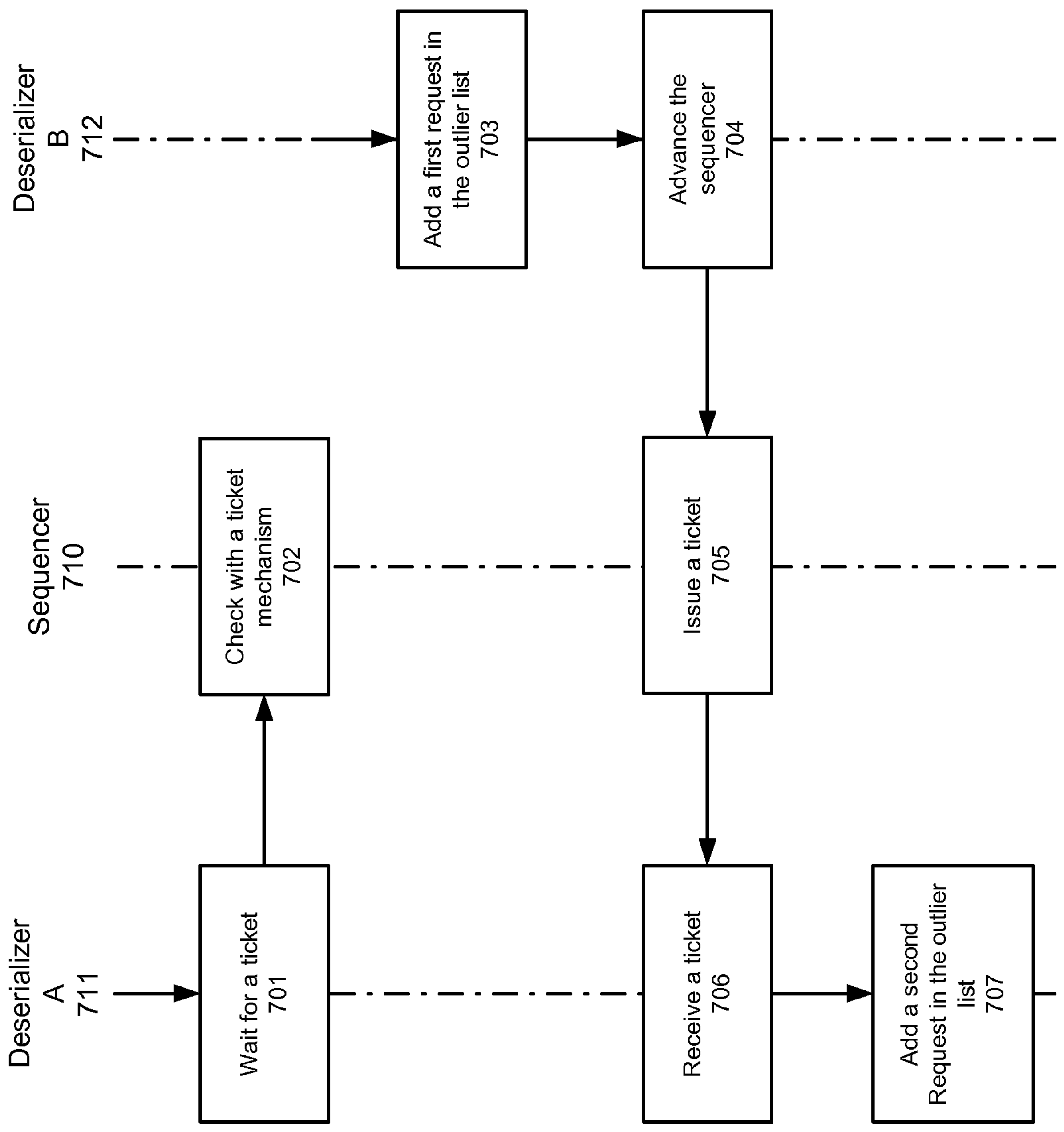
Deserializer A 711
Sequencer 710
Deserializer B 712
Wait for a ticket 701
Check with a ticket mechanism 702
Add a first request in the outlier list 703
Advance the sequencer 704
Issue a ticket 705
Receive a ticket 706
Add a second Request in the outlier list 707

FIGURE 7

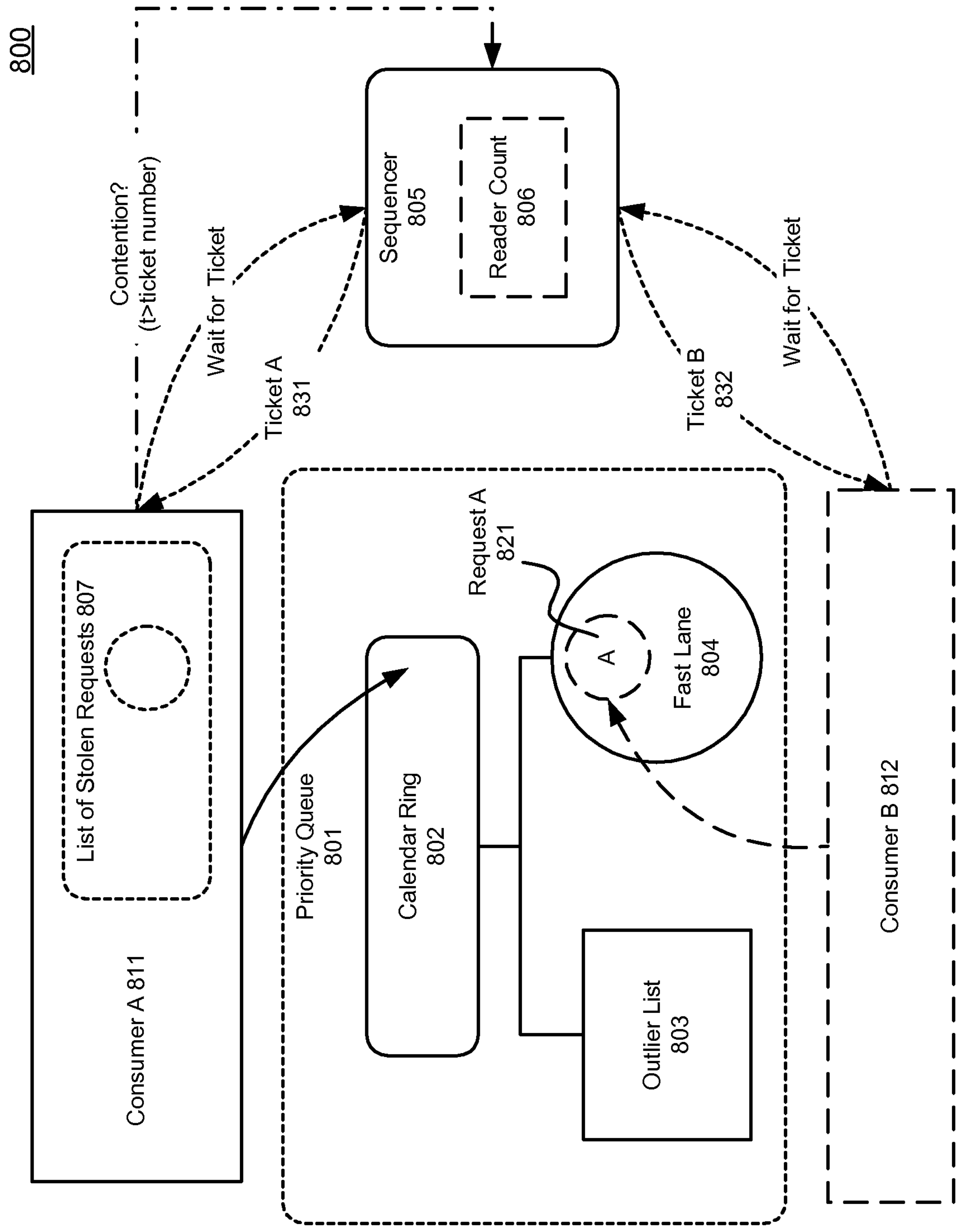
800
Contention? (t>ticket number)
Wait for Ticket
Ticket A 831
Sequencer 805
Reader Count 806
Ticket B 832
Wait for Ticket
Consumer A 811
List of Stolen Requests 807
Priority Queue 801
Calendar Ring 802
Request A 821
A
Fast Lane 804
Outlier List 803
Consumer B 812

FIGURE 8

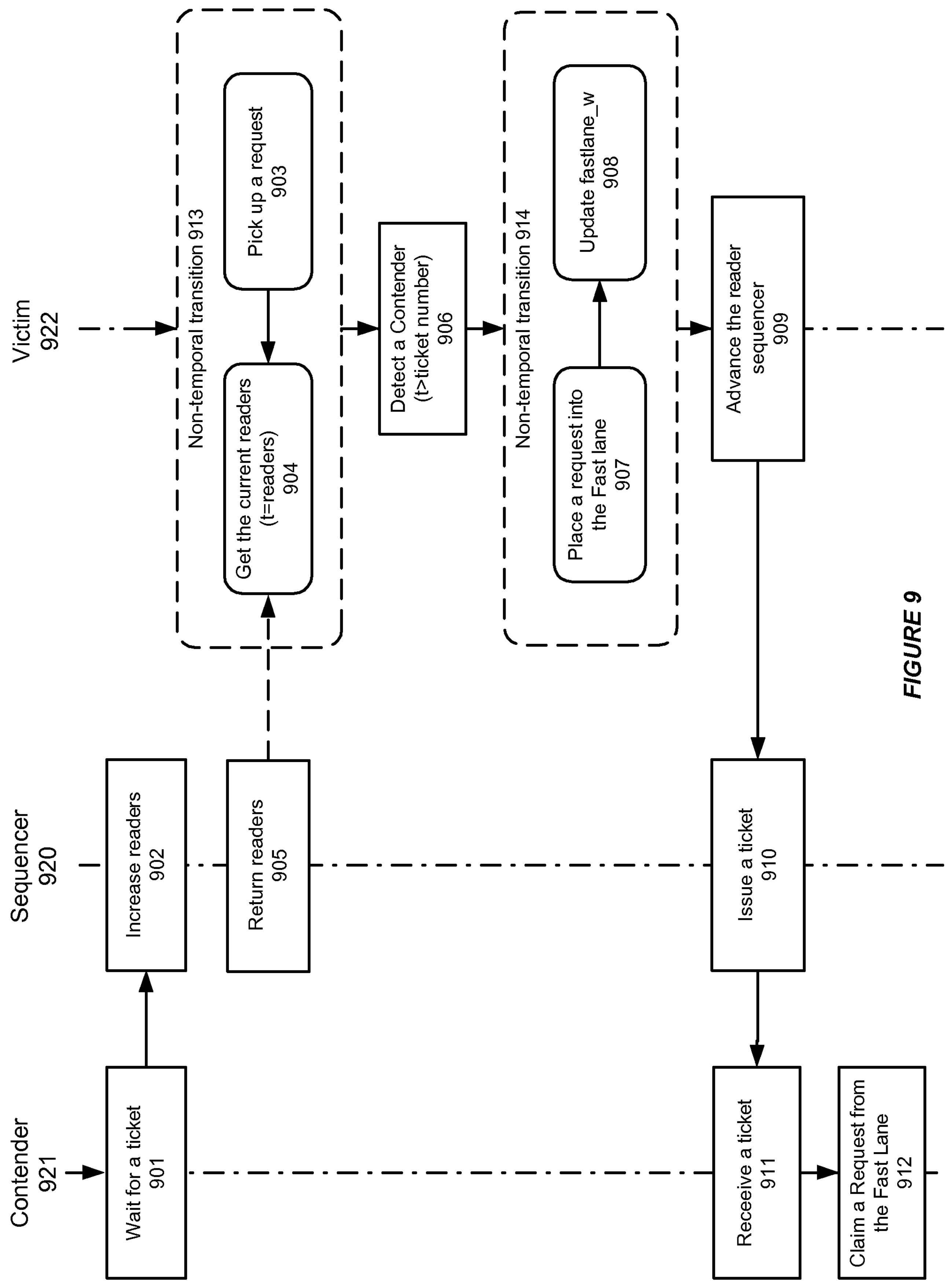
Contender 921
Sequencer 920
Victim 922
Wait for a ticket 901
Increase readers 902
Pick up a request 903
Get the current readers (t=readers) 904
Return readers 905
Detect a Contender (t>ticket number) 906
Place a request into the Fast lane 907
Update fastlane_w 908
Advance the reader sequencer 909
Issue a ticket 910
Receeive a ticket 911
Claim a Request from the Fast Lane 912
Non-temporal transition 913
Non-temporal transition 914

FIGURE 9

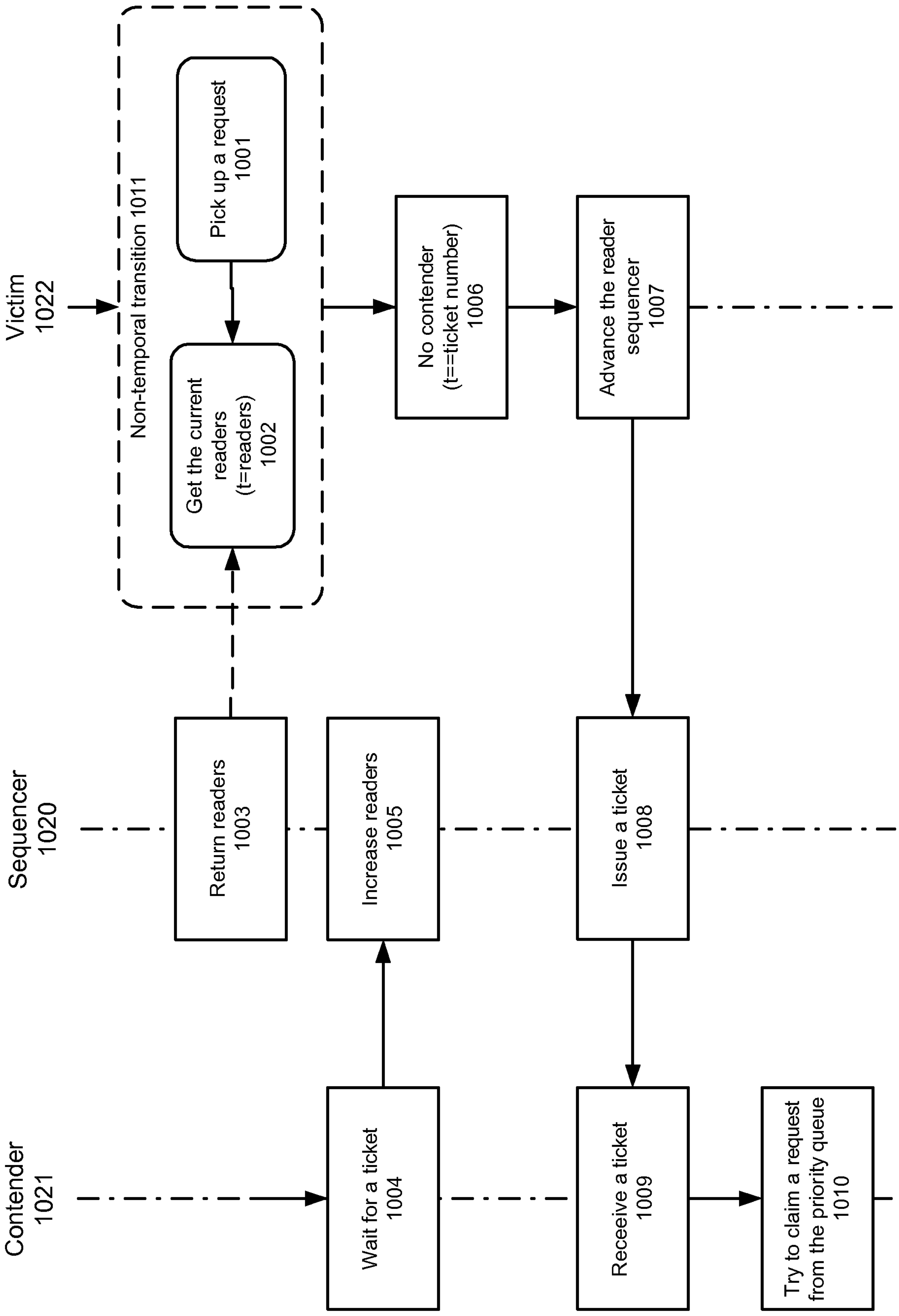
Victim 1022
Non-temporal transition 1011
Pick up a request 1001
Get the current readers (t=readers) 1002
No contender (t==ticket number) 1006
Advance the reader sequencer 1007
Sequencer 1020
Return readers 1003
Increase readers 1005
Issue a ticket 1008
Contender 1021
Wait for a ticket 1004
Receeive a ticket 1009
Try to claim a request from the priority queue 1010

FIGURE 10

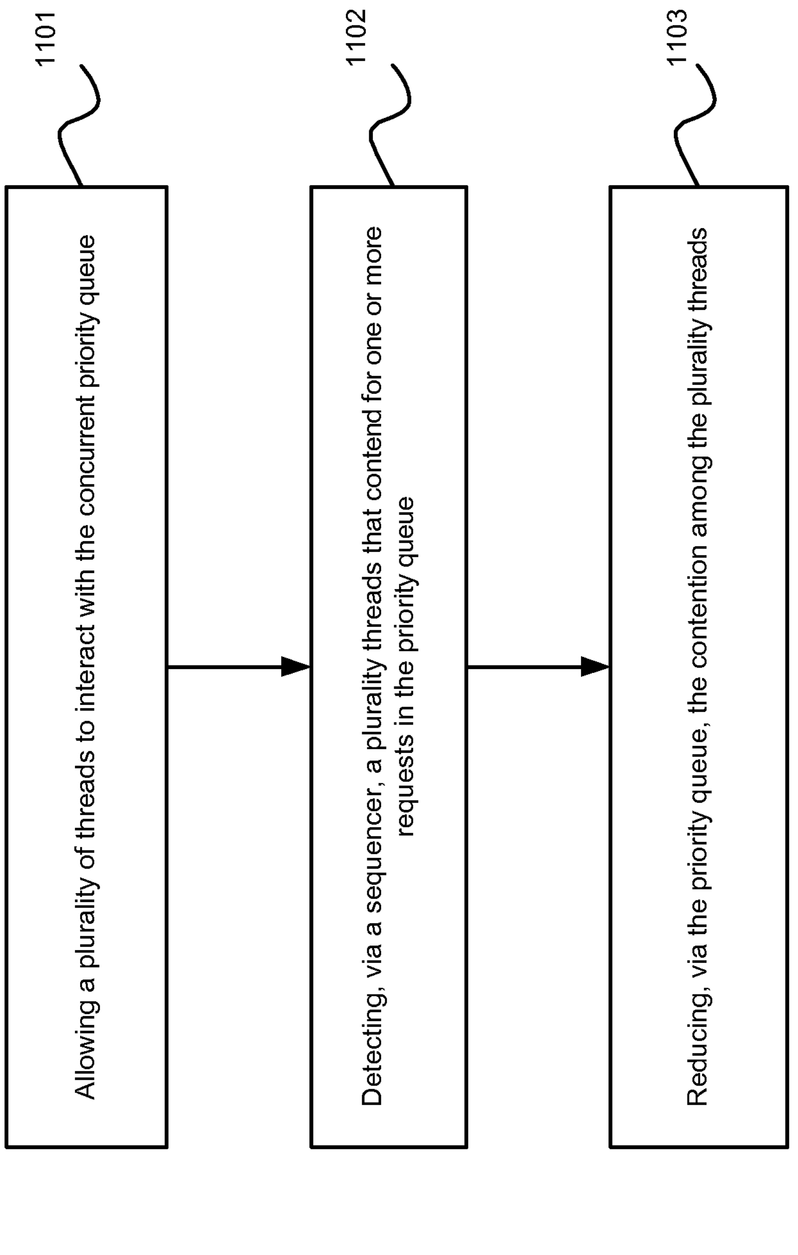
1101
Allowing a plurality of threads to interact with the concurrent priority queue
1102
Detecting, via a sequencer, a plurality threads that contend for one or more requests in the priority queue
1103
Reducing, via the priority queue, the contention among the plurality threads

FIGURE 11

INTERNATIONAL SEARCH REPORT

International application No
PCT/US2013/067108

A. CLASSIFICATION OF SUBJECT MATTER
INV. G06F9/52 G06F9/54
ADD.

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

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)

EPO-Internal, INSPEC, WPI Data

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	US 2007/118601 A1 (PACHECO GARY A [CA] PACHECO GARY ADAM [CA]) 24 May 2007 (2007-05-24) paragraphs [0005] - [0008] paragraphs [0018] - [0057] figures 1-8 -----	1-22
X	US 2005/283577 A1 (SIVARAM RAJEEV [US] ET AL) 22 December 2005 (2005-12-22) paragraphs [0005] - [0009] paragraphs [0022] - [0048] figures 1-9 -----	1-22
A	US 2012/158684 A1 (LOWENSTEIN RAFAEL [IL] ET AL) 21 June 2012 (2012-06-21) paragraphs [0002] - [0008] paragraphs [0016] - [0031] figures 1-4 ----- -/--	1-22

[X] Further documents are listed in the continuation of Box C.

[X] 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 27 February 2014	Date of mailing of the international search report 14/03/2014
Name and mailing address of the ISA/ European Patent Office, P.B. 5818 Patentlaan 2 NL - 2280 HV Rijswijk Tel. (+31-70) 340-2040, Fax: (+31-70) 340-3016	Authorized officer Noll, Joachim

INTERNATIONAL SEARCH REPORT

International application No
PCT/US2013/067108

C(Continuation). DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	US 2011/153992 A1 (SRINIVAS SURESH [US] ET AL) 23 June 2011 (2011-06-23) paragraphs [0013] - [0044] figures 1-10 -----	1-22
A	US 2006/015700 A1 (BURKA PETER W [CA]) 19 January 2006 (2006-01-19) paragraphs [0003] - [0009] paragraphs [0014] - [0045] figures 1-3 -----	1-22

INTERNATIONAL SEARCH REPORT

Information on patent family members

International application No
PCT/US2013/067108

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
US 2007118601 A1	24-05-2007	US 2007118601 A1 WO 2007062510 A1	24-05-2007 07-06-2007
US 2005283577 A1	22-12-2005	CN 1713151 A TW I391818 B US 2005283577 A1	28-12-2005 01-04-2013 22-12-2005
US 2012158684 A1	21-06-2012	NONE	
US 2011153992 A1	23-06-2011	NONE	
US 2006015700 A1	19-01-2006	NONE	