Exhibit 596-14

Invalidity of the ’596 Patent by Michaylov ’684 and Other Prior Art

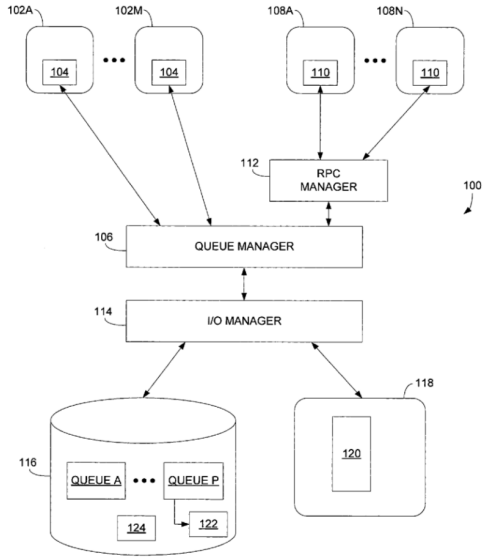
The asserted claims of U.S. Patent No. 8,924,596 are anticipated and/or rendered obvious in view of US7865684B2 (“Michaylov ’684”) alone or in combination with knowledge of one of skill in the art or other prior art.

This claim chart is based on Defendants’ present understanding of the asserted claims, Plaintiffs’ vague and overbroad infringement contentions (and corresponding, implicit claim constructions), and Defendants’ investigation to date. Defendants are not adopting Plaintiffs’ apparent constructions, nor are Defendants admitting to the correctness of any particular construction. Defendants reserve all rights to amend this invalidity claim chart in light of any claim construction developments or any amendments to Plaintiffs’ infringement contentions. Further, as discovery is ongoing and Defendants continue to seek discovery from third parties regarding the references identified in Defendants’ invalidity contentions as well as other potential prior art, Defendants reserve the right to amend or supplement this claim chart in connection with Defendants’ continuing investigation. At a minimum, Defendants intend to revise these invalidity contentions as appropriate based on discovery from third parties as discovery progresses.

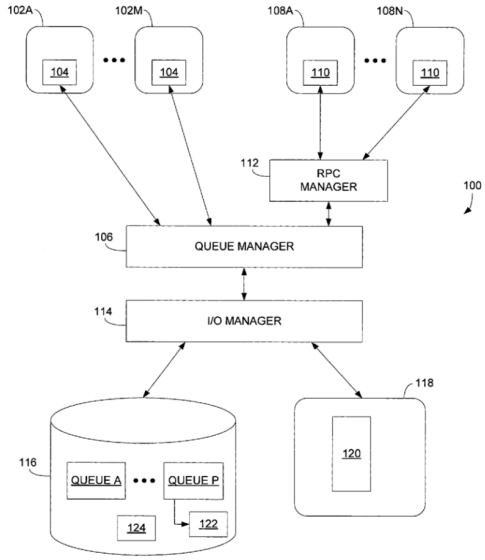
The claim chart below identifies where each limitation of each asserted claim of the ’596 patent can be found in Michaylov ’684. This analysis is based on Plaintiff’s interpretation of the claims as indicated in their Infringement Contentions. The citations provided below are exemplary, rather than exhaustive, and Defendants reserve the right to rely upon any other portion of the cited references.

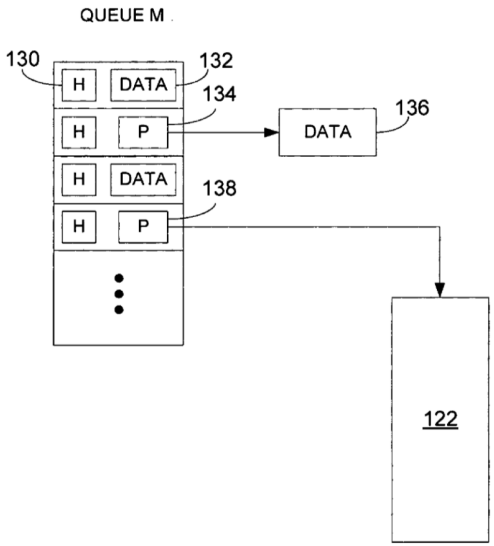
Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
[1pre] A system for dividing and synchronizing a processing task across a plurality of processing elements comprising:	To the extent the preamble is considered a claim limitation, Michaylov ’684 teaches “[a] system for dividing and synchronizing a processing task across a plurality of processing elements,” including at least the following exemplary citations. For example, Michaylov ’684 teaches a system for dividing and synchronizing processing tasks by disclosing, as non–limiting examples, access entities, queue transaction modules, and queue managers that coordinate, divide, and synchronize messages. *See, e.g.:*

IPR2025-00486
Concurrent Ventures LLC, XtreamEdge Inc. EX2035

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	 FIG. 1A Michaylov ’684 at Figure 1A. “In a general aspect, the invention features a method, and corresponding software and system, for writing data to a plurality of queues, each portion of the data being written to a corresponding one of the queues. The method includes, without requiring concurrent locking of more than one queue, determining if a space is available in each queue for writing a corresponding portion of the data, and if available, reserving the spaces in the queues. The method includes writing each portion of the data to a corresponding one of the queues.” Michaylov ’684 at col. 1, ll. 22–30. “FIG. 1A shows a queueing system 100 in which a set of trusted access entities 102A-102M each includes a queue transaction module 104 for interacting directly with a queue manager 106. The queueing system 100 also includes a set of untrusted access entities 108A-108N each including a remote queue transaction module 110 for interacting with the queue manager 106 through a remote procedure call (RPC) manager 112.” Michaylov ’684 at col. 3, ll. 8–15. “The queueing system 100 provides a mechanism for passing messages between the access entities through one or more message queues. The access entities provide interfaces for modules to interact with the queueing system 100. For example, a “publisher” computational module in a distributed computing system can pass

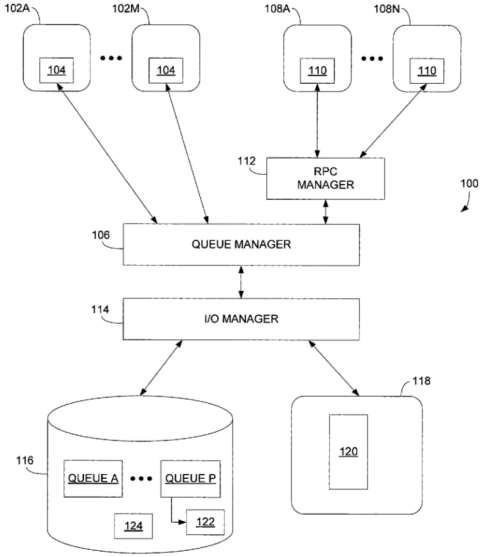
Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	messages including processed data elements to one or more "subscriber" computational modules." Michaylov '684 at col. 3, ll. 16–23.
[1a] an input queue implemented in hardware;	Michaylov '684 teaches "an input queue implemented in hardware," including at least the following exemplary citations. For example, Michaylov '684 teaches an input queue implemented in hardware, by disclosing, as non–limiting examples, multiple messages queues. Plaintiffs have cited the use of an input queue implemented in memory through software as allegedly satisfying limitation [1a]. *See* Am. Compl. at ¶59; '596 Infringement Contentions at 43–58. Michaylov '684 teaches or renders obvious claim limitation [1a] under Plaintiffs' apparent interpretation of the claim. *See, e.g.*, Michaylov '684, Fig. 1b (queues stored in non–volatile memory 116); 3:24–31 ("The queue manager 106 interacts with an input/output (I/O) manager 114 that manages memory storage for a set of message queues with corresponding queue data structures QUEUE_A-QUEUE_P that are each assigned storage space (e.g., a set of disk pages) in a volatile memory store"); 4"13–18 ("The queue manager 106 passes the messages to the I/O manager 114 to write to volatile memory store 118 and eventually to non-volatile memory store 116. Alternative division of functions per formed by"). *See, e.g.:* "A method, and corresponding system and software, is described for writing data to a plurality of queues, each portion of the data being written to a corresponding one of the queues. The method includes, without requiring concurrent locking of more than one queue, determining if a space is available in each queue for writing a corresponding portion of the data, and if available, reserving the spaces in the queues. The method includes writing each portion of the data to a corresponding one of the queues." Michaylov '684 at Abstract.

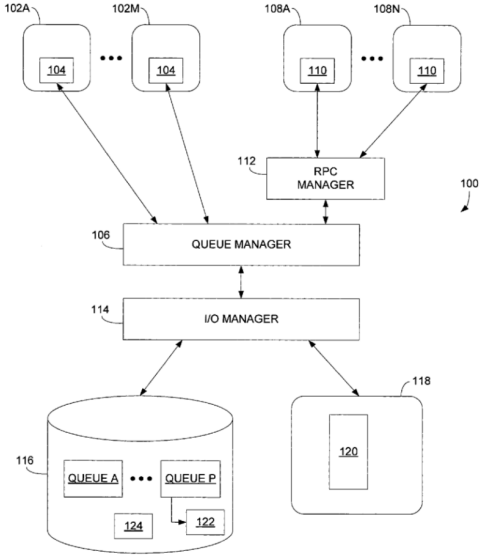
Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	 FIG. 1A Michaylov ’684 at Figure 1A.

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	 FIG. 1B Michaylov ’684 at Figure 1B. “The invention relates to managing message queues. Message queues can be used to provide an asynchronous communication protocol for access entities (e.g., servers, operating systems, software modules, etc.) to exchange messages. Messages placed onto a message queue are stored in a queue data structure until a recipient (e.g., a subscriber to the message queue) retrieves them.” Michaylov ’684 at col. 1, ll. 5–11. “In a general aspect, the invention features a method, and corresponding software and system, for writing data to a plurality of queues, each portion of the data being written to a corresponding one of the queues. The method includes, without requiring concurrent locking of more than one queue, determining if a space is available in each queue for writing a corresponding portion of the data, and if available, reserving the spaces in the queues. The method includes writing each portion of the data to a corresponding one of the queues.” Michaylov ’684 at col. 1, ll. 22–30. “FIG. 1A shows a queueing system 100 in which a set of trusted access entities 102A-102M each includes a queue transaction module 104 for interacting directly with a queue manager 106. The queueing system 100 also includes a set of untrusted access entities 108A-108N each including a remote queue transaction module

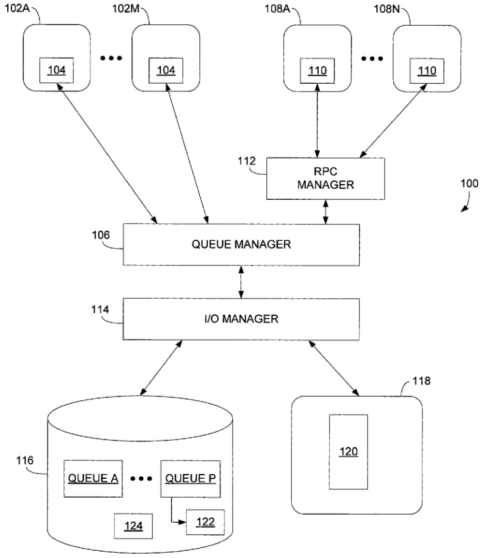
Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	110 for interacting with the queue manager 106 through a remote procedure call (RPC) manager 112." Michaylov '684 at col. 3, ll. 8–15. "The queueing system 100 provides a mechanism for passing messages between the access entities through one or more message queues. The access entities provide interfaces for modules to interact with the queueing system 100. For example, a "publisher" computational module in a distributed computing system can pass messages including processed data elements to one or more "subscriber" computational modules." Michaylov '684 at col. 3, ll. 16–23. "A queue data structure stores records including "message records" (also called simply "messages") that include the message data being distributed, and "control records" that include information used by the queueing system 100 to manage the queue. FIG. 1B shows an exemplary queue data structure QUEUE_M containing a series of message records each including a message header 130 and message data 132. The queue can optionally store message data along with the message header 130, or alternatively can store a pointer 134 along with the message header 130 specifying an address of externally stored message data 136. In a "large transaction indirection technique" described below, a record can optionally include a pointer 138 to a large transaction data structure 122 that stores a sequence of messages" Michaylov '684 at col. 3, ll. 40–53. "The queueing system 100 supports various data distribution models including a publish-subscribe data distribution model. An access entity (trusted or untrusted) acting as a "publisher" to a queue can add one or more messages to the queue (also called a "topic") in a "write transaction." An access entity (trusted or untrusted) acting as a "subscriber" to a queue can read one or more messages from the queue in a "read transaction." Multiple publishers can add messages to the same queue, and multiple subscribers can read the same messages from the same queue. The queue manager 106 removes a message from a queue after all subscribers to the queue have read the message. Alternatively, in a point-to-point data distribution model, multiple access entities can add messages to a queue, but each message is read from the queue by a single access entity. A "compound transaction" includes interaction with more than one queue, as described in more detail below." Michaylov '684 at col. 3, l. 54 – col. 4, l. 3. "The I/O manager 114 maintains messages for any particular write transaction in the order in which they were presented to the queue manager 106, when storing the messages to an ordered queue data structure in non-volatile memory 118. The order of messages stored in the queue data structure is determined, for example, by a linked list of disk pages that are currently part of the queue data structure. Messages for different write transactions may be interleaved in the queue data structure. If a new write transaction is initiated after a

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	previously initiated write transaction is committed, then all messages associated with the new write transaction occur after all of the messages in the previous write transaction in the queue data structure." Michaylov '684 at col. 5, ll. 55–67.
[1b] an output queue implemented in hardware;	Michaylov '684 teaches "an output queue implemented in hardware," including at least the following exemplary citations. For example, Michaylov '684 teaches an output queue implemented in hardware, by disclosing, as non–limiting examples, multiple messages queues. Plaintiffs have cited the use of an input queue implemented in memory through software as allegedly satisfying limitation [1a]. *See* Am. Compl. at ¶59; '596 Infringement Contentions at 43–58. Michaylov '684 teaches or renders obvious claim limitation [1a] under Plaintiffs' apparent interpretation of the claim. *See, e.g.*, Michaylov '684, Fig. 1b (queues stored in non–volatile memory 116); 3:24–31 ("The queue manager 106 interacts with an input/output (I/O) manager 114 that manages memory storage for a set of message queues with corresponding queue data structures QUEUE_A-QUEUE_P that are each assigned storage space (e.g., a set of disk pages) in a volatile memory store"); 4"13–18 ("The queue manager 106 passes the messages to the I/O manager 114 to write to volatile memory store 118 and eventually to non-volatile memory store 116. Alternative division of functions per formed by"). *See, e.g.:*

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	 FIG. 1A Michaylov ’684 at Figure 1A. “In a “compound transaction” an access entity can write to and/or read from more than one queue with all the messages being associated with the same transaction ID. A compound transaction may also be committed or rolled back. When a compound transaction is committed, in a “compound commit” operation, a commit record is added to each queue to which messages are being written in the compound transaction. These “queue commit records” are used to signal that the corresponding committed messages can be passed to a subscriber.” Michaylov ’684 at col. 4, ll. 56–65.
[1c] a first processing element having access to said input queue and said output queue;	Michaylov ’684 teaches “a first processing element having access to said input queue and said output queue,” including at least the following exemplary citations. For example, Michaylov ’684 teaches a first processing element having access to said input queue and said output queue, by disclosing, as non–limiting examples, an access entity communicating with the message queues. The first processing element has access to both the input queue and output queue. *See, e.g.:*

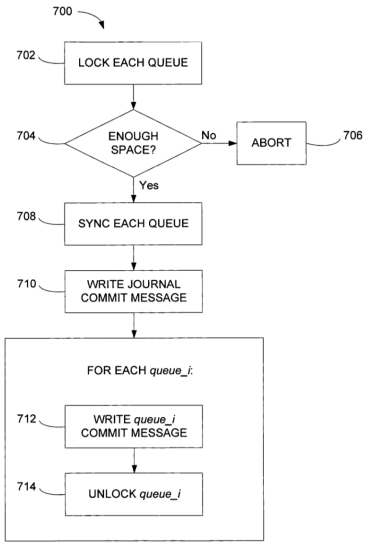
Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	 FIG. 1A Michaylov ’684 at Figure 1A. “FIG. 1A shows a queueing system 100 in which a set of trusted access entities 102A-102M each includes a queue transaction module 104 for interacting directly with a queue manager 106. The queueing system 100 also includes a set of untrusted access entities 108A-108N each including a remote queue transaction module 110 for interacting with the queue manager 106 through a remote procedure call (RPC) manager 112.” Michaylov ’684 at col. 3, ll. 8–15. “The queueing system 100 provides a mechanism for passing messages between the access entities through one or more message queues. The access entities provide interfaces for modules to interact with the queueing system 100. For example, a “publisher” computational module in a distributed computing system can pass messages including processed data elements to one or more “subscriber” computational modules.” Michaylov ’684 at col. 3, ll. 16–23. “The queueing system 100 supports various data distribution models including a publish-subscribe data distribution model. An access entity (trusted or untrusted) acting as a “publisher” to a queue can add one or more messages to the queue (also called a “topic”) in a “write transaction.” An access entity (trusted or untrusted) acting as a “subscriber” to a queue can read one or more messages from the queue in a “read

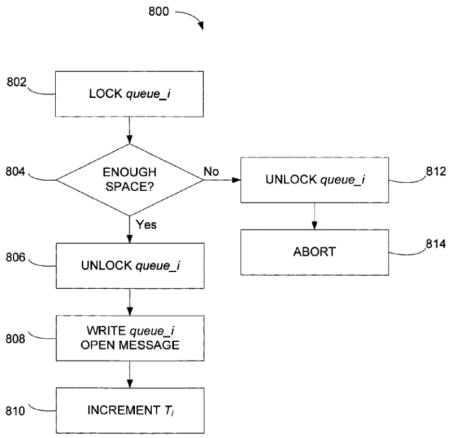
Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	transaction." Multiple publishers can add messages to the same queue, and multiple subscribers can read the same messages from the same queue. The queue manager 106 removes a message from a queue after all subscribers to the queue have read the message. Alternatively, in a point-to-point data distribution model, multiple access entities can add messages to a queue, but each message is read from the queue by a single access entity. A "compound transaction" includes interaction with more than one queue, as described in more detail below." Michaylov '684 at col. 3, l. 54 – col. 4, l. 3. "In a "compound transaction" an access entity can write to and/or read from more than one queue with all the messages being associated with the same transaction ID. A compound transaction may also be committed or rolled back. When a compound transaction is committed, in a "compound commit" operation, a commit record is added to each queue to which messages are being written in the compound transaction. These "queue commit records" are used to signal that the corresponding committed messages can be passed to a subscriber." Michaylov '684 at col. 4, ll. 56–65.
[1d] at least one second processing element in communication with said first processing element;	Michaylov '684 teaches "at least one second processing element in communication with said first processing element," including at least the following exemplary citations. For example, Michaylov '684 teaches at least one second processing element in communication with said first processing element, by disclosing, as non–limiting examples, a queue manager, interacting with the first processing element. A queue manager, the second processing element is in communication with the first processing element. *See, e.g.:*

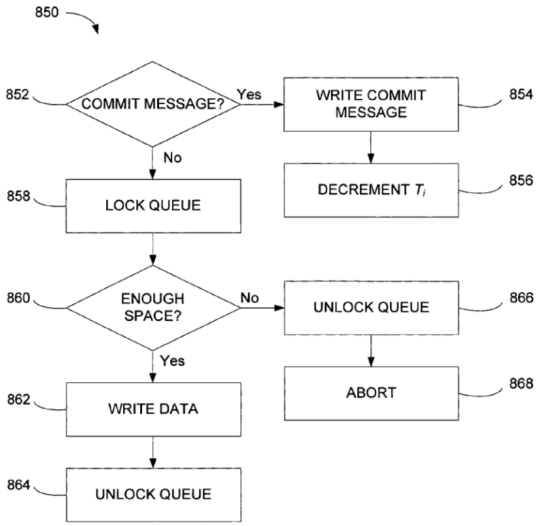
Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	 FIG. 1A Michaylov ’684 at Figure 1A. “FIG. 1A shows a queueing system 100 in which a set of trusted access entities 102A-102M each includes a queue transaction module 104 for interacting directly with a queue manager 106. The queueing system 100 also includes a set of untrusted access entities 108A-108N each including a remote queue transaction module 110 for interacting with the queue manager 106 through a remote procedure call (RPC) manager 112.” Michaylov ’684 at col. 3, ll. 8–15. “The queueing system 100 provides a mechanism for passing messages between the access entities through one or more message queues. The access entities provide interfaces for modules to interact with the queueing system 100. For example, a “publisher” computational module in a distributed computing system can pass messages including processed data elements to one or more “subscriber” computational modules.” Michaylov ’684 at col. 3, ll. 16–23.
[1e] a reservation register implemented in hardware storing a value indicative of available space	Michaylov ’684 teaches “a reservation register implemented in hardware storing a value indicative of available space in said input queue, said reservation register accessible by both said first processing element and said at least one second processing element,” including at least the following exemplary citations.

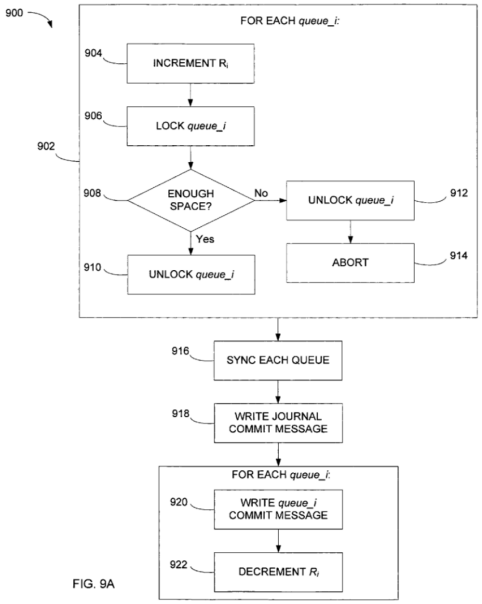
Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
in said input queue, said reservation register accessible by both said first processing element and said at least one second processing element;	For example, Michaylov ’684 teaches a reservation register implemented in hardware storing a value indicative of available space in said input queue, said reservation register accessible by both said first processing element and said at least one second processing element, by disclosing, as non–limiting examples, a counter associated with a queue or a reservation counter, keeping tracking of the number of transactions on the queue. A counter associated with a queue stores a value of indicative space and is accessible by both the first and second processing elements discussed above. Plaintiffs have cited the use of data structures in memory storing a value indicative of available space, e.g., qstate, as allegedly satisfying limitation [1e]. *See* Am. Compl. at ¶62; ’596 Infringement Contentions at 100–137. To the extent that Plaintiffs argue that Michaylov ’684’s queue counter is in memory, Michaylov ’684 discloses or renders obvious claim limitation [1e] under Plaintiffs’ apparent interpretation of the claim. *See, e.g.:* “Reserving the spaces in the queues includes incrementing a corresponding counter for each queue.” Michaylov ’684 at col. 2, ll. 4–5. “FIG. 8A shows a flowchart for an “open operation” 800 that is performed at the beginning of a write transaction. For each queue queue_i, the queue manager 106 keeps track of the number of outstanding transactions T_i for which a commit record has not yet been written. Before a opening new transaction on queue_i and incrementing T_i, the queue manager 106 locks 802 and checks 804 the queue (a single queue for a simple write transaction or each queue to which messages are being added in a compound transaction) to ensure there is space for an “open record” and a commit record. The queue manager 106 compares the current available space with D_o, given by: [Equation] If the available space is larger than or equal to D_o, then the queue manager 106 unlocks 806 queue_i and writes 808 an “open record” and increments 810 the number of outstanding transactions T_i. Otherwise, if the available space is smaller than D_o, then the queue manager 106 unlocks 812 queue_i and aborts 814 the open operation 800.” Michaylov ’684 at col. 10, l. 61 – col. 11, l. 14.
[1f] computer storage storing instructions, which when executed by said at least one second processing element:	Michaylov ’684 teaches computer storage storing instructions executed by said at least one second processing element, including at least the following exemplary citations. For example, Michaylov ’684 teaches computer storage storing instructions that are executed by said at least one second processing element, by disclosing, as non–limiting examples, that the queue managing functions of the second processing element (e.g., a queue manager) may be implemented as a computer program stored

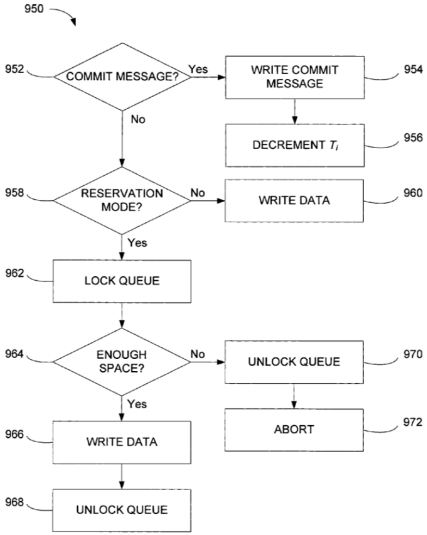
Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	in a computer–readable storage medium, executed by, for example, a processor. The second processing element executes those instructions stored in the computer storage. *See, e.g.:* "The queue management functions described herein can be implemented using software for execution on a computer. For instance, the software forms procedures in one or more computer programs that execute on one or more programmed or programmable computer systems (which may be of various architectures such as distributed, client/server, or grid) each including at least one processor, at least one data storage system (including volatile and non-volatile memory and/or storage elements), at least one input device or port, and at least one output device or port. The software may form one or more modules of a larger program, for example, that provides other services related to the design and configuration of computation graphs. The data structures described herein can be implemented as data structures stored in a computer readable medium or other organized data conforming to a data model stored in a data repository. The software may be provided on a medium, such as a CD-ROM, readable by a general or special purpose programmable computer or delivered (encoded in a propagated signal) over a network to the computer where it is executed. All of the functions may be performed on a special purpose computer, or using special-purpose hardware, such as coprocessors. The software may be implemented in a distributed manner in which different parts of the computation specified by the software are performed by different computers. Each such computer program is preferably stored on or downloaded to a storage media or device (e.g., solid state memory or media, or magnetic or optical media) readable by a general or special purpose programmable computer, for configuring and operating the computer when the storage media or device is read by the computer system to perform the procedures described herein. The inventive system may also be considered to be implemented as a computer-readable storage medium, configured with a computer program, where the storage medium so configured causes a computer system to operate in a specific and predefined manner to perform the functions described herein." Michaylov '684 at col. 13, ll. 9–45.
[1g] accesses said reservation register and reads said stored value;	Michaylov '684 teaches "accesses said reservation register and reads said stored value," including at least the following exemplary citations. For example, Michaylov '684 teaches that the second processing element executes instructions that causes it to access said reservation register and reads said stored value, by disclosing, as non–limiting examples, the second processing element (e.g., a queue manager) checking the queue counter. The second processing element accesses and reads the stored value in the reservation register. *See, e.g.:*

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	"A method, and corresponding system and software, is described for writing data to a plurality of queues, each portion of the data being written to a corresponding one of the queues. The method includes, without requiring concurrent locking of more than one queue, determining if a space is available in each queue for writing a corresponding portion of the data, and if available, reserving the spaces in the queues. The method includes writing each portion of the data to a corresponding one of the queues." Michaylov '684 at Abstract.  FIG. 7 Michaylov '684 at Figure 7.

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	 FIG. 8A Michaylov ’684 at Figure 8A.

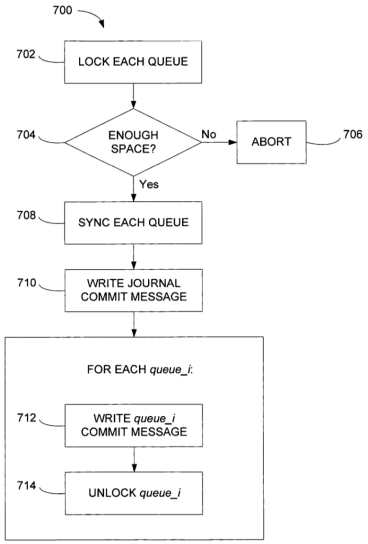
Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	 FIG. 8C Michaylov ’684 at Figure 8C.

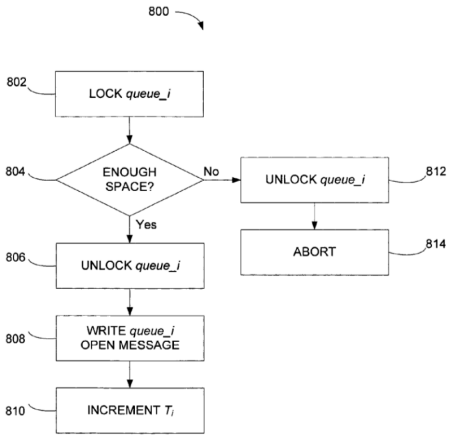
Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	 Michaylov ’684 at Figure 9A.

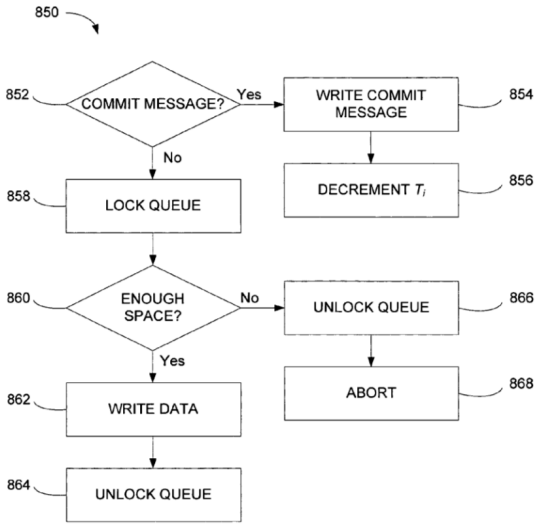
Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	 FIG. 9B Michaylov ’684 at Figure 9B. “Reserving the spaces in the queues includes incrementing a corresponding counter for each queue.” Michaylov ’684 at col. 2, ll. 4–5. “After the publisher supplies the queue manager 106 all the messages to be added in a write transaction, the publisher may request that the queue manager 106 “commit” or “roll back” the write transaction. In order to commit a write transaction, the queue manager 106 adds a “commit record” to the corresponding queue data structure in non-volatile memory. The commit record indicates that the messages of a committed write transaction (“committed messages”) can be passed to a subscriber. Before a write transaction is committed, the associated messages are made durable by ensuring that they are synchronized from volatile memory to non-volatile memory (if they have not already been synchronized).” Michaylov ’684 at col. 4, ll. 19–30. “In one approach to managing queue commit records for a compound commit operation, the queue manager 106 ensures that enough space for a commit record is available in each of the queues by concurrently locking each queue while writing the commit records to prevent space from being used up during the commit operation. In second and third approaches, to perform the compound commit operation more efficiently, the queue manager 106 uses a method for guaranteeing successful writing of a commit record to each of multiple

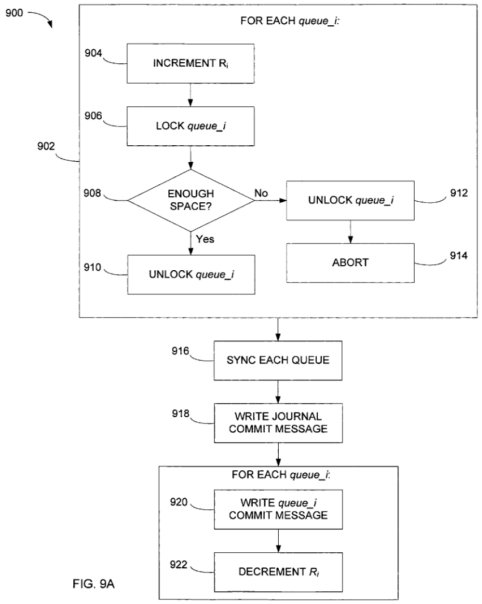
Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	queues without requiring concurrent locking of more than one queue. Each of these three approaches is described in more detail below." Michaylov '684 at col. 9, ll. 46–57. "After locking 702 each queue, the queue manager 106 checks 704 the storage space available in each queue. In the example of FIG. 6, the queue data structure QUEUE_A has a block of storage 600 available and the queue data structure QUEUE_B has a block of storage 602 available. The queue manager 106 compares the amount of space available in each queue with the amount of space that would be used to write a commit record for the compound transaction being committed. If any of the queues does not have enough available space reserved for a commit record, then the queue manager 106 aborts 706 the compound commit operation. The queue manager 106 may attempt the compound commit operation at a later time and/or attempt to acquire more storage space for one or more of the queues." Michaylov '684 at col. 10, ll. 9–22. "FIG. 8A shows a flowchart for an "open operation" 800 that is performed at the beginning of a write transaction. For each queue queue_i, the queue manager 106 keeps track of the number of outstanding transactions T_i for which a commit record has not yet been written. Before a opening new transaction on queue_i and incrementing T_i, the queue manager 106 locks 802 and checks 804 the queue (a single queue for a simple write transaction or each queue to which messages are being added in a compound transaction) to ensure there is space for an "open record" and a commit record. The queue manager 106 compares the current available space with D_o, given by: [Equation] If the available space is larger than or equal to D_o, then the queue manager 106 unlocks 806 queue_i and writes 808 an "open record" and increments 810 the number of outstanding transactions T_i. Otherwise, if the available space is smaller than D_o, then the queue manager 106 unlocks 812 queue_i and aborts 814 the open operation 800." Michaylov '684 at col. 10, l. 61 – col. 11, l. 14. "Referring to FIG. 8C, the queue manager 106 uses the write operation 850 for data to be written to a queue data structure. The queue manager 106 first determines 852 whether the data to be written is a commit record. If so, the queue manager 106 writes the commit record 854 and decrements 856 the number of outstanding transactions T_i. If not, the queue manager 106 locks 858 the queue and checks 860 the storage space available in the queue. The queue manager 106 compares the current available space with D_w, given by: [Equation] where size_of (data) returns the size of data in the appropriate units (e.g., bytes). If the available space is larger than or equal to D_w, then the queue manager 106 writes 862 the data and unlocks 864 the queue. Otherwise, if the available space is smaller than D_w, then the queue manager 106 unlocks 866 the queue and aborts 868 the write operation 850. In this approach, only a single queue, the queue to which data is being written, is locked while storage space is checked. 3.3 Third Approach" Michaylov '684 at col. 11, ll. 37–58.

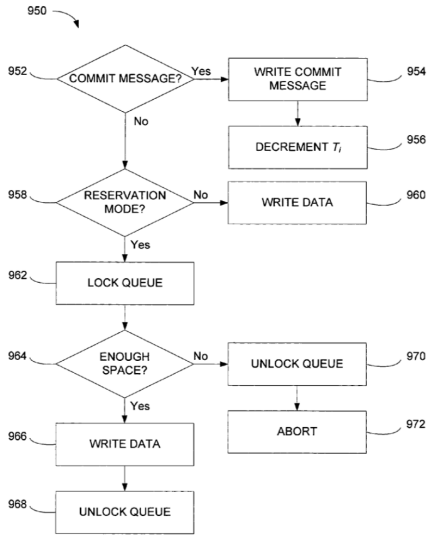
Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	"After incrementing the reservation mode counter, in the loop 902, the queue manager 106 locks 906 queue_i and checks 908 the storage space available in queue_i. The queue manager 106 compares the current available space with D_c, given by: [Equation] If the available space is larger than or equal to D_c, then the queue manager 106 unlocks 910 queue_i and continues. Otherwise, if the available space is smaller than D_c, then the queue manager 106 unlocks 912 queue_i and aborts 914 the commit record writing operation 900." Michaylov '684 at col. 12, ll. 25–36.
[1h] determines when said read value indicates available space in said input queue for said first processing element to issue a command;	Michaylov '684 teaches "determines when said read value indicates available space in said input queue for said first processing element to issue a command," including at least the following exemplary citations. For example, Michaylov '684 teaches that the second processing element executes instructions that causes it to determine when said read value indicates available space in said input queue for said first processing element to issue a command, by disclosing, as non–limiting examples, the second processing element (e.g., a queue manager) comparing the space needed by a transaction to the available space and determining when is available space. The second processing element determines if the stored value indicates available space in the input queue for the first processing element to issue a command. *See, e.g.:* "A method, and corresponding system and software, is described for writing data to a plurality of queues, each portion of the data being written to a corresponding one of the queues. The method includes, without requiring concurrent locking of more than one queue, determining if a space is available in each queue for writing a corresponding portion of the data, and if available, reserving the spaces in the queues. The method includes writing each portion of the data to a corresponding one of the queues." Michaylov '684 at Abstract.

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	 FIG. 7 Michaylov ’684 at Figure 7.

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	 FIG. 8A Michaylov ’684 at Figure 8A.

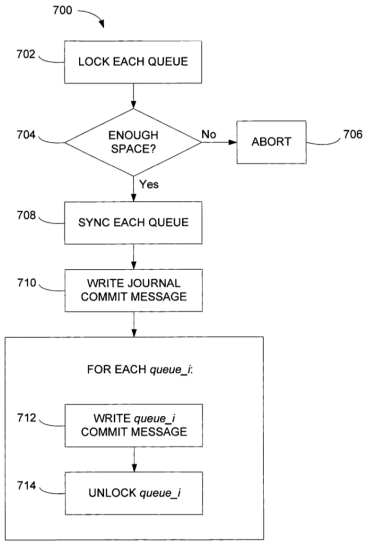
Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	 FIG. 8C Michaylov ’684 at Figure 8C.

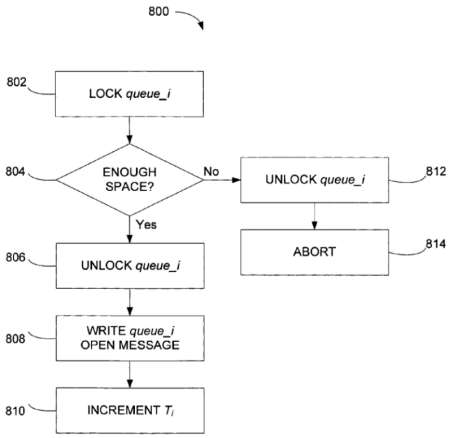
Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	 Michaylov ’684 at Figure 9A.

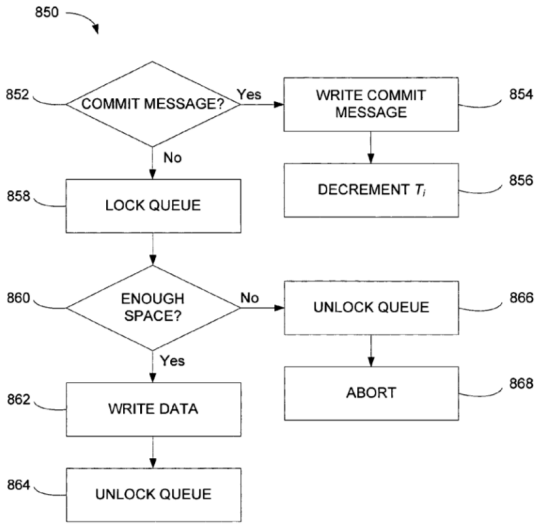
Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	 FIG. 9B Michaylov ’684 at Figure 9B. “In a general aspect, the invention features a method, and corresponding software and system, for writing data to a plurality of queues, each portion of the data being written to a corresponding one of the queues. The method includes, without requiring concurrent locking of more than one queue, determining if a space is available in each queue for writing a corresponding portion of the data, and if available, reserving the spaces in the queues. The method includes writing each portion of the data to a corresponding one of the queues.” Michaylov ’684 at col. 1, ll. 22–30. “Determining if the space is available in each queue and if available reserving the spaces includes, for each of multiple of the queues: locking the queue; determining if the space is available in the queue; if available reserving the space; and unlocking the queue.” Michaylov ’684 at col. 1, ll. 39–43. “Reserving the spaces in the queues includes incrementing a corresponding counter for each queue.” Michaylov ’684 at col. 2, ll. 4–5. “After the publisher supplies the queue manager 106 all the messages to be added in a write transaction, the publisher may request that the queue manager 106 “commit” or “roll back” the write transaction. In order to commit a write transaction, the queue manager 106 adds a “commit record” to the corresponding queue data

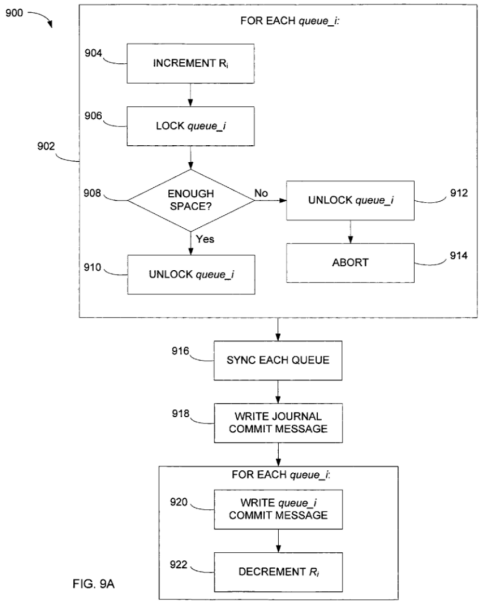
Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	structure in non-volatile memory. The commit record indicates that the messages of a committed write transaction ("committed messages") can be passed to a subscriber. Before a write transaction is committed, the associated messages are made durable by ensuring that they are synchronized from volatile memory to non-volatile memory (if they have not already been synchronized)." Michaylov '684 at col. 4, ll. 19–30. "In one approach to managing queue commit records for a compound commit operation, the queue manager 106 ensures that enough space for a commit record is available in each of the queues by concurrently locking each queue while writing the commit records to prevent space from being used up during the commit operation. In second and third approaches, to perform the compound commit operation more efficiently, the queue manager 106 uses a method for guaranteeing successful writing of a commit record to each of multiple queues without requiring concurrent locking of more than one queue. Each of these three approaches is described in more detail below." Michaylov '684 at col. 9, ll. 46–57. "After locking 702 each queue, the queue manager 106 checks 704 the storage space available in each queue. In the example of FIG. 6, the queue data structure QUEUE_A has a block of storage 600 available and the queue data structure QUEUE_B has a block of storage 602 available. The queue manager 106 compares the amount of space available in each queue with the amount of space that would be used to write a commit record for the compound transaction being committed. If any of the queues does not have enough available space reserved for a commit record, then the queue manager 106 aborts 706 the compound commit operation. The queue manager 106 may attempt the compound commit operation at a later time and/or attempt to acquire more storage space for one or more of the queues." Michaylov '684 at col. 10, ll. 9–22. "FIG. 8A shows a flowchart for an "open operation" 800 that is performed at the beginning of a write transaction. For each queue queue_i, the queue manager 106 keeps track of the number of outstanding transactions T_i for which a commit record has not yet been written. Before a opening new transaction on queue_i and incrementing T_i, the queue manager 106 locks 802 and checks 804 the queue (a single queue for a simple write transaction or each queue to which messages are being added in a compound transaction) to ensure there is space for an "open record" and a commit record. The queue manager 106 compares the current available space with D_o, given by: [Equation] If the available space is larger than or equal to D_o, then the queue manager 106 unlocks 806 queue_i and writes 808 an "open record" and increments 810 the number of outstanding transactions T_i. Otherwise, if the available space is smaller than D_o, then the queue manager 106 unlocks 812 queue_i and aborts 814 the open operation 800." Michaylov '684 at col. 10, l. 61 – col. 11, l. 14. "Referring to FIG. 8C, the queue manager 106 uses the write operation 850 for data to be written to a queue data structure. The queue manager 106 first determines 852 whether the data to be written is a commit record.

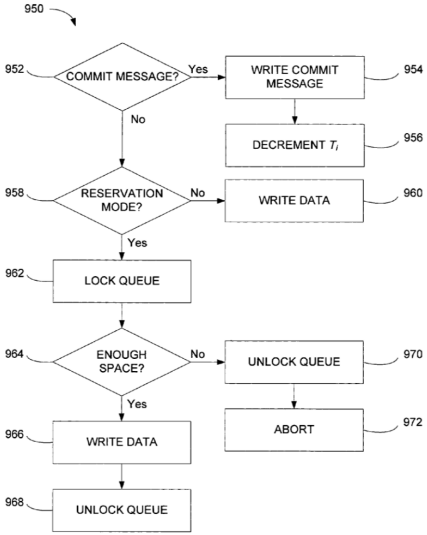
Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	If so, the queue manager 106 writes the commit record 854 and decrements 856 the number of outstanding transactions T_i. If not, the queue manager 106 locks 858 the queue and checks 860 the storage space available in the queue. The queue manager 106 compares the current available space with D_w, given by: [Equation] where size_of (data) returns the size of data in the appropriate units (e.g., bytes). If the available space is larger than or equal to D_w, then the queue manager 106 writes 862 the data and unlocks 864 the queue. Otherwise, if the available space is smaller than D_w, then the queue manager 106 unlocks 866 the queue and aborts 868 the write operation 850. In this approach, only a single queue, the queue to which data is being written, is locked while storage space is checked. 3.3 Third Approach" Michaylov '684 at col. 11, ll. 37–58. "After incrementing the reservation mode counter, in the loop 902, the queue manager 106 locks 906 queue_i and checks 908 the storage space available in queue_i. The queue manager 106 compares the current available space with D_c, given by: [Equation] If the available space is larger than or equal to D_c, then the queue manager 106 unlocks 910 queue_i and continues. Otherwise, if the available space is smaller than D_c, then the queue manager 106 unlocks 912 queue_i and aborts 914 the commit record writing operation 900." Michaylov '684 at col. 12, ll. 25–36.
[1i] notifies said first processing element to issue said command to said input queue; and	Michaylov '684 teaches "notifies said first processing element to issue said command to said input queue," including at least the following exemplary citations. For example, Michaylov '684 teaches that the second processing element executes instructions that causes it to notify said first processing element to issue said command to said input queue, by disclosing, as non–limiting examples, the second processing element (e.g., a queue manager) writing an open record. The second processing element notifies the first processing element to issue the command to the input queue. *See, e.g.:*

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	 FIG. 7 Michaylov ’684 at Figure 7.

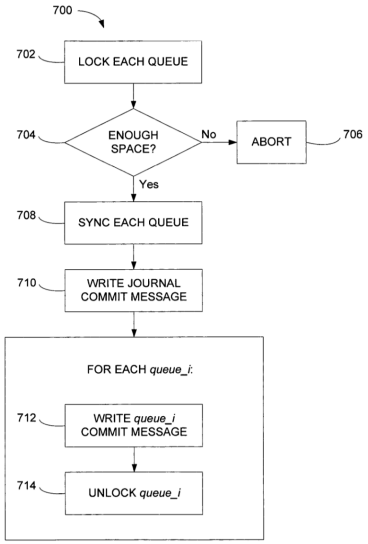
Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	 FIG. 8A Michaylov ’684 at Figure 8A.

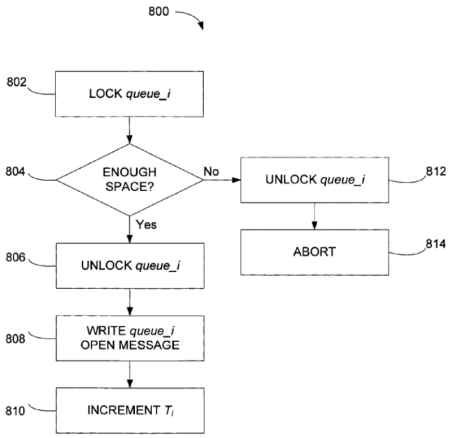
Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	 FIG. 8C Michaylov ’684 at Figure 8C.

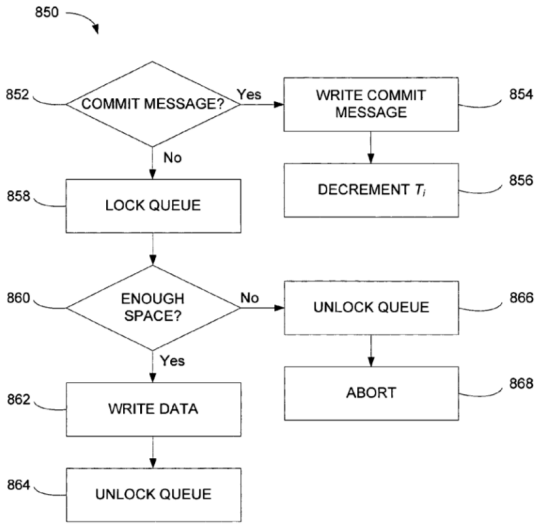
Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	 Michaylov ’684 at Figure 9A.

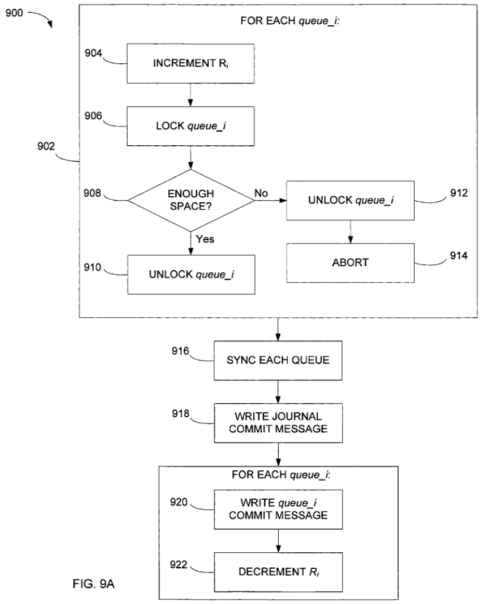
Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	 FIG. 9B Michaylov ’684 at Figure 9B. “After the publisher supplies the queue manager 106 all the messages to be added in a write transaction, the publisher may request that the queue manager 106 “commit” or “roll back” the write transaction. In order to commit a write transaction, the queue manager 106 adds a “commit record” to the corresponding queue data structure in non-volatile memory. The commit record indicates that the messages of a committed write transaction (“committed messages”) can be passed to a subscriber. Before a write transaction is committed, the associated messages are made durable by ensuring that they are synchronized from volatile memory to non-volatile memory (if they have not already been synchronized).” Michaylov ’684 at col. 4, ll. 19–30. “If the queues do have enough storage space reserved for commit records, then queue manager 106 synchronizes 708 each queue to ensure that any messages stored in volatile memory have been moved to non-volatile memory. After each queue has been synchronized, the queue manager 106 writes 710 a commit record to the journal data structure 126. After writing the journal commit record, the queue manager 106 writes 712 a commit record to each queue. After writing the commit record to a queue, the queue manager 106 unlocks 714 that queue.” Michaylov ’684 at col. 10, ll. 23–32.

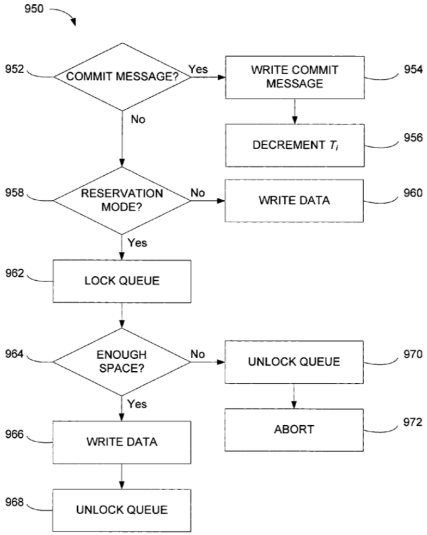
Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	"Referring to FIG. 8C, the queue manager 106 uses the write operation 850 for data to be written to a queue data structure. The queue manager 106 first determines 852 whether the data to be written is a commit record. If so, the queue manager 106 writes the commit record 854 and decrements 856 the number of outstanding transactions T_i. If not, the queue manager 106 locks 858 the queue and checks 860 the storage space available in the queue. The queue manager 106 compares the current available space with D_w, given by: [Equation] where size_of (data) returns the size of data in the appropriate units (e.g., bytes). If the available space is larger than or equal to D_w, then the queue manager 106 writes 862 the data and unlocks 864 the queue. Otherwise, if the available space is smaller than D_w, then the queue manager 106 unlocks 866 the queue and aborts 868 the write operation 850. In this approach, only a single queue, the queue to which data is being written, is locked while storage space is checked. 3.3 Third Approach" Michaylov '684 at col. 11, ll. 37–58.
[1j] wherein said first processing element receives notification from said at least one second processing element regarding issuing the command, issues said command to said input queue, and receives a response corresponding to said command from said output queue.	Michaylov '684 teaches "wherein said first processing element receives notification from said at least one second processing element regarding issuing the command, issues said command to said input queue, and receives a response corresponding to said command from said output queue," including at least the following exemplary citations. For example, Michaylov '684 teaches wherein said first processing element receives notification from said at least one second processing element regarding issuing the command, issues said command to said input queue, and receives a response corresponding to said command from said output queue, by disclosing, as non–limiting examples, the first processing element (e.g., an access entity) receiving a notification from the queue manager (e.g., a transaction ID), issuing the command (e.g., passing messages to the queue), and receiving a corresponding response (e.g., abort or queueing of the transaction) or results. The first processing element receives the notification, issues the command, and receives a corresponding response from the output queue. *See, e.g.:*

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	 FIG. 7 Michaylov ’684 at Figure 7.

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	 FIG. 8A Michaylov ’684 at Figure 8A.

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	 FIG. 8C Michaylov ’684 at Figure 8C.

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	 Michaylov ’684 at Figure 9A.

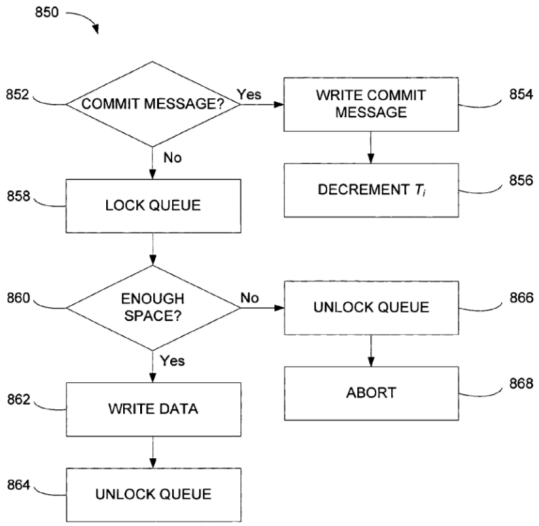
Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	 FIG. 9B Michaylov ’684 at Figure 9B. “The invention relates to managing message queues. Message queues can be used to provide an asynchronous communication protocol for access entities (e.g., servers, operating systems, software modules, etc.) to exchange messages. Messages placed onto a message queue are stored in a queue data structure until a recipient (e.g., a subscriber to the message queue) retrieves them.” Michaylov ’684 at col. 1, ll. 5–11. “In a general aspect, the invention features a method, and corresponding software and system, for writing data to a plurality of queues, each portion of the data being written to a corresponding one of the queues. The method includes, without requiring concurrent locking of more than one queue, determining if a space is available in each queue for writing a corresponding portion of the data, and if available, reserving the spaces in the queues. The method includes writing each portion of the data to a corresponding one of the queues.” Michaylov ’684 at col. 1, ll. 22–30. “After the publisher supplies the queue manager 106 all the messages to be added in a write transaction, the publisher may request that the queue manager 106 “commit” or “roll back” the write transaction. In order to commit a write transaction, the queue manager 106 adds a “commit record” to the corresponding queue data

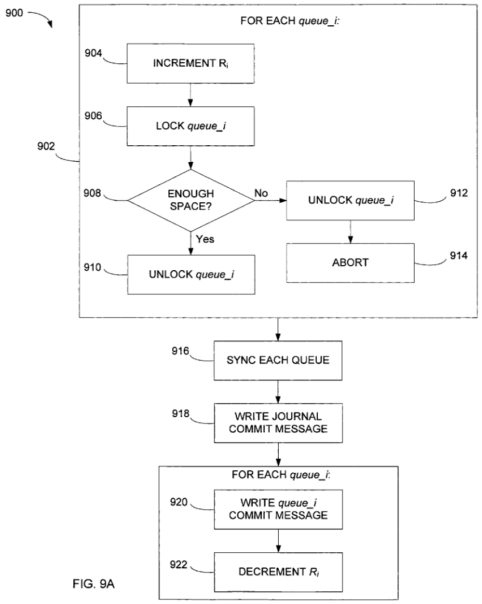
Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	structure in non-volatile memory. The commit record indicates that the messages of a committed write transaction ("committed messages") can be passed to a subscriber. Before a write transaction is committed, the associated messages are made durable by ensuring that they are synchronized from volatile memory to non-volatile memory (if they have not already been synchronized)." Michaylov '684 at col. 4, ll. 19–30. "In a "compound transaction" an access entity can write to and/or read from more than one queue with all the messages being associated with the same transaction ID. A compound transaction may also be committed or rolled back. When a compound transaction is committed, in a "compound commit" operation, a commit record is added to each queue to which messages are being written in the compound transaction. These "queue commit records" are used to signal that the corresponding committed messages can be passed to a subscriber." Michaylov '684 at col. 4, ll. 56–65. "After locking 702 each queue, the queue manager 106 checks 704 the storage space available in each queue. In the example of FIG. 6, the queue data structure QUEUE_A has a block of storage 600 available and the queue data structure QUEUE_B has a block of storage 602 available. The queue manager 106 compares the amount of space available in each queue with the amount of space that would be used to write a commit record for the compound transaction being committed. If any of the queues does not have enough available space reserved for a commit record, then the queue manager 106 aborts 706 the compound commit operation. The queue manager 106 may attempt the compound commit operation at a later time and/or attempt to acquire more storage space for one or more of the queues." Michaylov '684 at col. 10, ll. 9–22. "If the queues do have enough storage space reserved for commit records, then queue manager 106 synchronizes 708 each queue to ensure that any messages stored in volatile memory have been moved to non-volatile memory. After each queue has been synchronized, the queue manager 106 writes 710 a commit record to the journal data structure 126. After writing the journal commit record, the queue manager 106 writes 712 a commit record to each queue. After writing the commit record to a queue, the queue manager 106 unlocks 714 that queue." Michaylov '684 at col. 10, ll. 23–32. "Referring to FIG. 8C, the queue manager 106 uses the write operation 850 for data to be written to a queue data structure. The queue manager 106 first determines 852 whether the data to be written is a commit record. If so, the queue manager 106 writes the commit record 854 and decrements 856 the number of outstanding transactions T_i. If not, the queue manager 106 locks 858 the queue and checks 860 the storage space available in the queue. The queue manager 106 compares the current available space with D_w, given by: [Equation] where size_of (data) returns the size of data in the appropriate units (e.g., bytes). If the available space is larger than or equal to D_w, then the queue manager 106 writes 862 the data and unlocks 864 the queue.

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	Otherwise, if the available space is smaller than D_w, then the queue manager 106 unlocks 866 the queue and aborts 868 the write operation 850. In this approach, only a single queue, the queue to which data is being written, is locked while storage space is checked. 3.3 Third Approach" Michaylov '684 at col. 11, ll. 37–58.
2. The system of claim 1, wherein said input queue is a first-in-first-out (FIFO) buffer.	Michaylov '684 teaches "[t]he system of claim 1, wherein said input queue is a first-in-first-out (FIFO) buffer," including at least the following exemplary citations. For example, Michaylov '684 teaches wherein said input queue is a first-in-first-out (FIFO) buffer, by disclosing, as non–limiting examples, the message queues storing outstanding transactions in the order they are presented to the queues, and/or the message queues maintaining FIFO sub-lists. *See, e.g.:* "To initiate a read transaction, a subscriber obtains a transaction ID and receives the next unread message or messages from the queue manager 106. The I/O manager 114 handles retrieving the messages from the appropriate queue data structure and the queue manager 106 passes them to the subscriber. Messages may or may not be passed back in the same order in which they were written to the queue data structure since only committed messages are passed to a subscriber and messages that have been committed may be interleaved with messages that have not yet been committed. The queue manager 106 determines which messages in a queue to pass to the subscriber by building up a "read data structure," as described in more detail below." Michaylov '684 at col. 4, ll. 43–55. "The I/O manager 114 maintains messages for any particular write transaction in the order in which they were presented to the queue manager 106, when storing the messages to an ordered queue data structure in non-volatile memory 118. The order of messages stored in the queue data structure is determined, for example, by a linked list of disk pages that are currently part of the queue data structure. Messages for different write transactions may be interleaved in the queue data structure. If a new write transaction is initiated after a previously initiated write transaction is committed, then all messages associated with the new write transaction occur after all of the messages in the previous write transaction in the queue data structure." Michaylov '684 at col. 5, ll. 55–67. "A subscriber can initiate a read transaction in which the subscriber requests one or more messages from a queue. The messages received by the subscriber may come from one write transaction, from a subset of a write transaction, or from more than one write transaction. As described in more detail below, the queue manager 106 passes messages to a subscriber from committed write transactions. Messages that were written in the same write transaction are provided to a subscriber in the order of the write transaction. Messages from different write transactions are provided to a subscriber in the order in which the write transactions were

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	committed. The same messages read by different subscribers to a queue are seen in the same order by those subscribers." Michaylov '684 at col. 6, ll. 1–13. "FIG. 4 shows an exemplary read data structure 400 corresponding to QUEUE_A. In this example, the read data structure 400 includes an ordered list of first-in/first-out (FIFO) sub-lists 401-404. Each time the queue manager 106 encounters a message corresponding to a new write transaction, a new FIFO sub-list is added to the read data structure 400 identified by the transaction ID for that write transaction. The queue manager 106 adds to each FIFO sub-list a pointer to each message associated with the corresponding write transaction in the order in which they are scanned (i.e., the order in which they were added to the queue)." Michaylov '684 at col. 7, l. 66 – col. 8, l. 9. "In scanning the queue data structure QUEUE_A shown in FIG. 4, the queue manager 106 generates a first FIFO sub-list 401 for write transaction T1 with pointers to messages M1 and M2. The queue manager 106 does not start passing messages back to the corresponding subscriber (in response to a read transaction) until confirming that the messages have been committed (i.e., until after scanning to a commit record for the associated transaction). After reaching the commit record for T1, the FIFO sub-list 401 for T1 is complete and the queue manager 106 temporarily stops building up the read data structure 400, retrieves the next message based on a pointer in the FIFO sub-list 401, and passes the message to the subscriber when the subscriber asks for a new message. After all the messages in the completed FIFO sub-list 401 have been passed to the subscriber, the queue manager 106 starts scanning QUEUE_A again to continue building up the read data structure 400 until the next FIFO sub-list 402 is complete. The queue manager alternates between handing messages to the subscriber from a completed FIFO sub-list and scanning the queue data structure to build up the read data structure. In this example, the messages M1 and M2 for T4 are passed to the subscriber before the messages M1-M3 for T3 since the commit record for T4 occurs before the commit record for T3. In alternative implementations, the queue manager can transition from handing messages to a subscriber to building up the read data structure before all the messages in a completed FIFO sub-list have been passed to the subscriber." Michaylov '684 at col. 8, ll. 10–36. "Each FIFO sub-list is maintained until after the corresponding subscriber receives all the messages in that FIFO sub-list, or until the queue manager 106 determines the corresponding write transaction will not be committed (e.g., after reading a roll back record). After a read transaction is committed, a commit record is written to the journal data structure 124 indicating which messages have been read by the subscriber. After the queue manager 106 finishes building up the read data structure 400, the read data structure 400 is retained

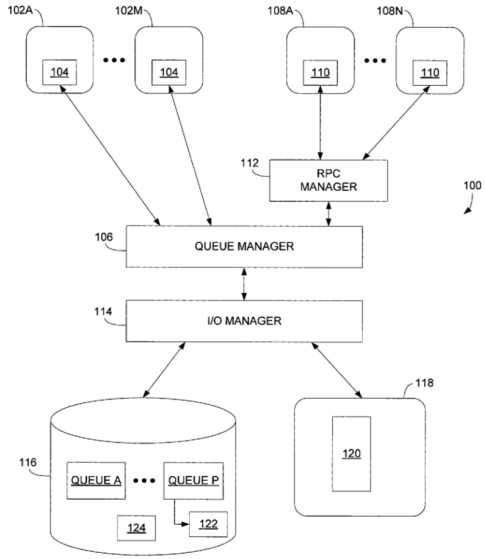
Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	for the next read transaction from the same queue for the same subscriber." Michaylov '684 at col. 8, ll. 37–47.
3. The system of claim 1, wherein said system is part of a storage area network (SAN) controller.	
4. The system of claim 1, wherein said system is part of a network attached storage (NAS) controller.	
5. The system of claim 1, wherein said stored value is decremented by a predefined value prior to returning said stored value to said at least one second processing element.	Michaylov '684 teaches "[t]he system of claim 1, wherein said stored value is decremented by a predefined value prior to returning said stored value to said at least one second processing element," including at least the following exemplary citations. For example, Michaylov '684 teaches wherein said stored value is decremented by a predefined value prior to returning said stored value to said at least one second processing element by disclosing, as non–limiting examples, the queue manager reserves space for a transaction by incrementing the value stored in the reservation counter prior to determining if there is enough space, and/or incrementing the value stored in the counter associated with the queue in relation to a transaction. A queue manager decrements the stored value by a predefined value. *See, e.g.:*

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	 FIG. 8C Michaylov ’684 at Figure 8C.

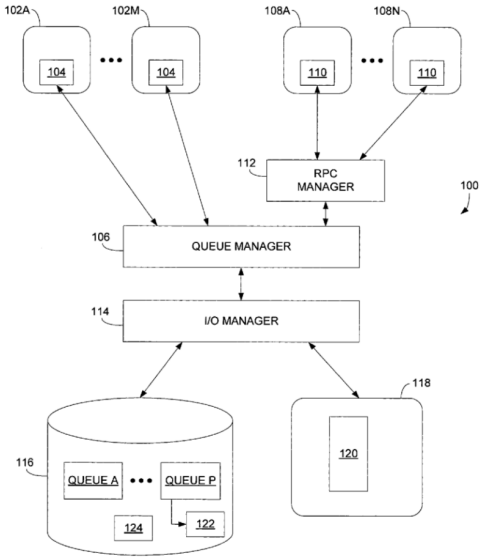
Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	 FIG. 9A Michaylov ’684 at Figure 9A. “Reserving the spaces in the queues includes incrementing a corresponding counter for each queue.” Michaylov ’684 at col. 2, ll. 4–5. “FIG. 8A shows a flowchart for an “open operation” 800 that is performed at the beginning of a write transaction. For each queue queue_i, the queue manager 106 keeps track of the number of outstanding transactions T_i for which a commit record has not yet been written. Before a opening new transaction on queue_i and incrementing T_i, the queue manager 106 locks 802 and checks 804 the queue (a single queue for a simple write transaction or each queue to which messages are being added in a compound transaction) to ensure there is space for an “open record” and a commit record. The queue manager 106 compares the current available space with D_o, given by: [Equation] If the available space is larger than or equal to D_o, then the queue manager 106 unlocks 806 queue_i and writes 808 an “open record” and increments 810 the number of outstanding transactions T_i. Otherwise, if the available space is smaller than D_o, then the queue manager 106 unlocks 812 queue_i and aborts 814 the open operation 800.” Michaylov ’684 at col. 10, l. 61 – col. 11, l. 14. “FIGS. 8B and 8C show a flowchart for a commit record writing operation 840 and an associated write operation 850, respectively. The queue manager 106 (or the I/O manager 114) uses the commit record writing

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	operation 840 for a commit operation, and uses the write operation 850 (potentially concurrently) to write any data other than commit records to a queue. In this approach, the commit record writing operation 840 does not need to check for available space in the queues since the write operation 850 includes this check before writing any data to a queue data structure. The queue manager 106 ensures that each queue queue_i reserves enough space for a commit record for each of the T_i outstanding transactions for that queue, as described in more detail below. Therefore, the queue manager 106 can safely assume that space has been reserved for each commit record when performing the commit record writing operation 840." Michaylov '684 at col. 11, ll. 15–30. "Referring to FIG. 8C, the queue manager 106 uses the write operation 850 for data to be written to a queue data structure. The queue manager 106 first determines 852 whether the data to be written is a commit record. If so, the queue manager 106 writes the commit record 854 and decrements 856 the number of outstanding transactions T_i. If not, the queue manager 106 locks 858 the queue and checks 860 the storage space available in the queue. The queue manager 106 compares the current available space with D_w, given by: [Equation] where size_of (data) returns the size of data in the appropriate units (e.g., bytes). If the available space is larger than or equal to D_w, then the queue manager 106 writes 862 the data and unlocks 864 the queue. Otherwise, if the available space is smaller than D_w, then the queue manager 106 unlocks 866 the queue and aborts 868 the write operation 850. In this approach, only a single queue, the queue to which data is being written, is locked while storage space is checked. 3.3 Third Approach" Michaylov '684 at col. 11, ll. 37–58. "Referring to FIG. 9A, in the commit record writing operation 900, the queue manager 106 first executes a loop 902 (or equivalent control structure) once for each queue queue_i included in the commit record writing operation 900 (i.e., where i=1 . . . number of queues in the operation 900). The loop 902 turns on reservation mode for queue_i in a way that can account for the possibility of multiple compound commit operations being performed concurrently. In this example, the loop 902 increments 904 a reservation mode counter R_i for queue_i. The reservation mode counter starts at an initial value, e.g., R_i=0, in which queue_i is not in reservation mode. When R_i>0, queue_i is in reservation mode. This allows the reservation mode state of a queue to be responsive to the number of times the queue has been placed into reservation mode." Michaylov '684 at col. 12, ll. 10–24. "After incrementing the reservation mode counter, in the loop 902, the queue manager 106 locks 906 queue_i and checks 908 the storage space available in queue_i. The queue manager 106 compares the current available space with D_c, given by: [Equation] If the available space is larger than or equal to D_c, then the queue manager 106 unlocks 910 queue_i and continues. Otherwise, if the available space is smaller than D_c,

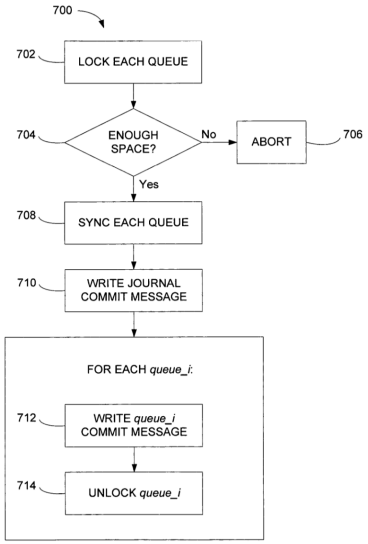
Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	then the queue manager 106 unlocks 912 queue_i and aborts 914 the commit record writing operation 900." Michaylov '684 at col. 12, ll. 25–36. "Referring to FIG. 9B, the queue manager 106 uses the write operation 950 for data to be written to a queue data structure. The queue manager 106 first determines 952 whether the data to be written is a commit record. If so, the queue manager 106 writes the commit record 954 and decrements 956 the number of outstanding transactions T_i. If not, the queue manager 106 determines 958 whether the queue is in reservation mode (e.g., by determining whether $R_i>0$). If the queue is not in reservation mode, then the queue manager 106 writes 960 the data. If the queue is in reservation mode, then the queue manager 106 locks 962 the queue and checks 964 the storage space available in the queue. The queue manager 106 compares the current available space with D_w as defined above for the second approach. If the available space is larger than or equal to D_w, then the queue manager 106 writes 966 the data and unlocks 968 the queue. Otherwise, if the available space is smaller than D_w, then the queue manager 106 unlocks 970 the queue and aborts 972 the write operation 950." Michaylov '684 at col. 12, ll. 44–61.
6. The system of claim 1, wherein at least one third processing element executes said command and notifies said output queue to issue said response.	Michaylov '684 teaches "[t]he system of claim 1, wherein at least one third processing element executes said command and notifies said output queue to issue said response," including at least the following exemplary citations. For example, Michaylov '684 teaches wherein at least one third processing element executes said command and notifies said output queue to issue said response, by disclosing, as non–limiting examples, the third processing element (e.g., another access entity) process data elements and messages and passing the results to another access entity. An access entity executes the command and notifies the output queue to issue a response. *See, e.g.:*

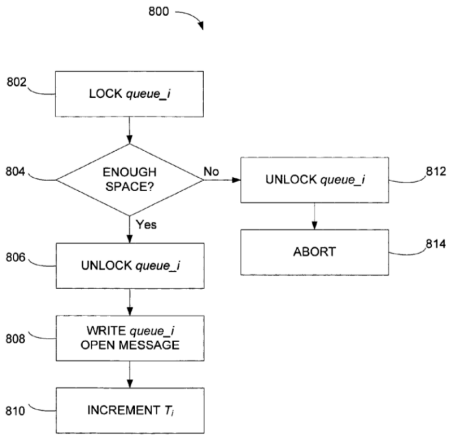
Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	 FIG. 1A Michaylov ’684 at Figure 1A. “FIG. 1A shows a queueing system 100 in which a set of trusted access entities 102A-102M each includes a queue transaction module 104 for interacting directly with a queue manager 106. The queueing system 100 also includes a set of untrusted access entities 108A-108N each including a remote queue transaction module 110 for interacting with the queue manager 106 through a remote procedure call (RPC) manager 112.” Michaylov ’684 at col. 3, ll. 8–15. “The queueing system 100 provides a mechanism for passing messages between the access entities through one or more message queues. The access entities provide interfaces for modules to interact with the queueing system 100. For example, a “publisher” computational module in a distributed computing system can pass messages including processed data elements to one or more “subscriber” computational modules.” Michaylov ’684 at col. 3, ll. 16–23. “The queueing system 100 supports various data distribution models including a publish-subscribe data distribution model. An access entity (trusted or untrusted) acting as a “publisher” to a queue can add one or more messages to the queue (also called a “topic”) in a “write transaction.” An access entity (trusted or untrusted) acting as a “subscriber” to a queue can read one or more messages from the queue in a “read

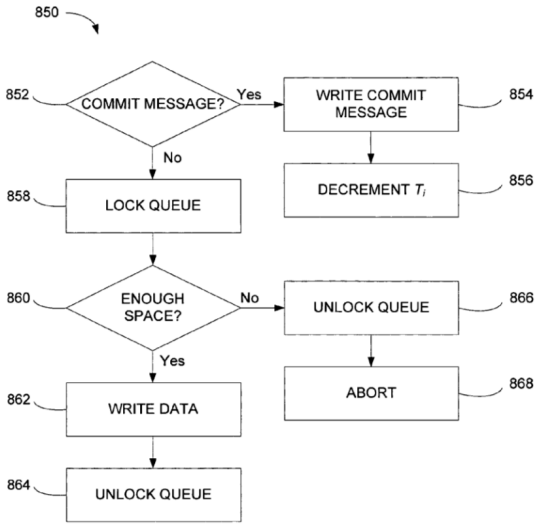
Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	transaction." Multiple publishers can add messages to the same queue, and multiple subscribers can read the same messages from the same queue. The queue manager 106 removes a message from a queue after all subscribers to the queue have read the message. Alternatively, in a point-to-point data distribution model, multiple access entities can add messages to a queue, but each message is read from the queue by a single access entity. A "compound transaction" includes interaction with more than one queue, as described in more detail below." Michaylov '684 at col. 3, l. 54 – col. 4, l. 3. "To initiate a read transaction, a subscriber obtains a transaction ID and receives the next unread message or messages from the queue manager 106. The I/O manager 114 handles retrieving the messages from the appropriate queue data structure and the queue manager 106 passes them to the subscriber. Messages may or may not be passed back in the same order in which they were written to the queue data structure since only committed messages are passed to a subscriber and messages that have been committed may be interleaved with messages that have not yet been committed. The queue manager 106 determines which messages in a queue to pass to the subscriber by building up a "read data structure," as described in more detail below." Michaylov '684 at col. 4, ll. 43–55.
[7pre] A system for dividing and synchronizing a processing task across a plurality of processing elements comprising:	Michaylov '684 teaches "[a] system for dividing and synchronizing a processing task across a plurality of processing elements," including at least the following exemplary citations. *See* above at row [1pre].
[7a] an input queue implemented in hardware;	Michaylov '684 teaches "an input queue implemented in hardware," including at least the following exemplary citations. *See* above at row [1a].
[7b] an output queue implemented in hardware;	Michaylov '684 teaches "an output queue implemented in hardware," including at least the following exemplary citations. *See* above at row [1b].
[7c] a first processing element having access to said input queue and said output queue;	Michaylov '684 teaches "a first processing element having access to said input queue and said output queue," including at least the following exemplary citations.

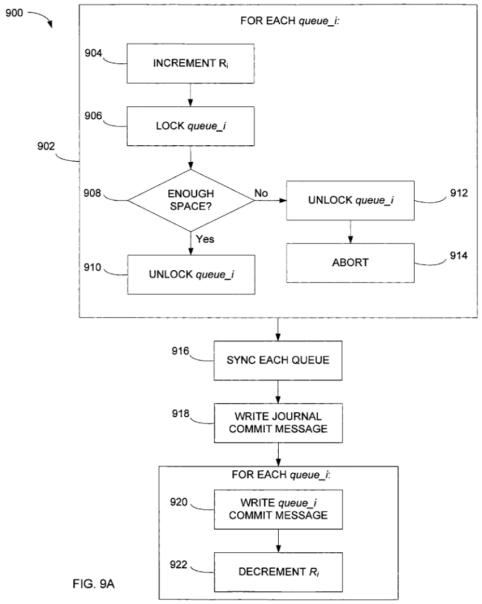
Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	See above at row [1c].
[7d] at least one second processing element in communication with said input queue;	Michaylov ’684 teaches “at least one second processing element in communication with said input queue,” including at least the following exemplary citations. For example, Michaylov ’684 teaches at least one second processing element in communication with said input queue, by disclosing, as non–limiting examples, a queue manager interacting with the message queues. A queue manager, the second processing element is in communication with the input queue. *See, e.g.:*  FIG. 1A Michaylov ’684 at Figure 1A. “FIG. 1A shows a queueing system 100 in which a set of trusted access entities 102A-102M each includes a queue transaction module 104 for interacting directly with a queue manager 106. The queueing system 100 also includes a set of untrusted access entities 108A-108N each including a remote queue transaction module 110 for interacting with the queue manager 106 through a remote procedure call (RPC) manager 112.” Michaylov ’684 at col. 3, ll. 8–15.

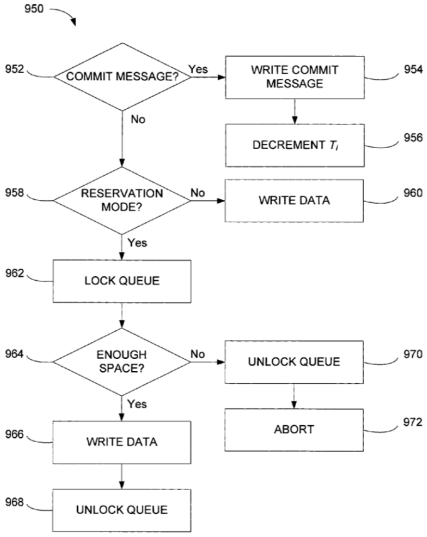
Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	"The queueing system 100 provides a mechanism for passing messages between the access entities through one or more message queues. The access entities provide interfaces for modules to interact with the queueing system 100. For example, a "publisher" computational module in a distributed computing system can pass messages including processed data elements to one or more "subscriber" computational modules." Michaylov '684 at col. 3, ll. 16–23. "The queue manager 106 interacts with an input/output (I/O) manager 114 that manages memory storage for a set of message queues with corresponding queue data structures QUEUE_A-QUEUE_P that are each assigned storage space (e.g., a set of disk pages) in a volatile memory store 118 (such as a semiconductor random access memory (RAM)), which is a temporary working storage that provides relatively fast access for reading and writing data. The I/O manager 114 also manages a non-volatile memory store 116 (such as a magnetic disk system), which is a permanent storage that provides relatively more persistence of data and which may provide relatively slower access for reading and writing than the volatile memory store. Optionally, there is a single I/O manager handling the I/O for all the queues, or there are multiple I/O managers running in parallel, each handling the I/O for a subset of the queues." Michaylov '684 at col. 3, ll. 24–39. "The queueing system 100 supports various data distribution models including a publish-subscribe data distribution model. An access entity (trusted or untrusted) acting as a "publisher" to a queue can add one or more messages to the queue (also called a "topic") in a "write transaction." An access entity (trusted or untrusted) acting as a "subscriber" to a queue can read one or more messages from the queue in a "read transaction." Multiple publishers can add messages to the same queue, and multiple subscribers can read the same messages from the same queue. The queue manager 106 removes a message from a queue after all subscribers to the queue have read the message. Alternatively, in a point-to-point data distribution model, multiple access entities can add messages to a queue, but each message is read from the queue by a single access entity. A "compound transaction" includes interaction with more than one queue, as described in more detail below." Michaylov '684 at col. 3, l. 54 – col. 4, l. 3.
[7e] a reservation register implemented in hardware storing a value indicative of available space in said input queue, said reservation register accessible by both said first processing element	Michaylov '684 teaches "a reservation register implemented in hardware storing a value indicative of available space in said input queue, said reservation register accessible by both said first processing element and said at least one second processing element," including at least the following exemplary citations. *See* above at row [1e].

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
and said at least one second processing element;	
[7f] computer storage storing instructions, which when executed by said at least one second processing element:	Michaylov ’684 teaches computer storage storing instructions executed by said at least one second processing element, including at least the following exemplary citations. *See* above at row [1f].
[7g] receives a command from said first processing element;	Michaylov ’684 teaches “receives a command from said first processing element,” including at least the following exemplary citations. For example, Michaylov ’684 teaches that the second processing element executes instructions that causes it to receive a command from said first processing element, by disclosing, as non–limiting examples, a queue manager receiving messages from an access entity. A queue manager, the second processing element receives a command from the first processing element. *See, e.g.:*  FIG. 7 Michaylov ’684 at Figure 7.

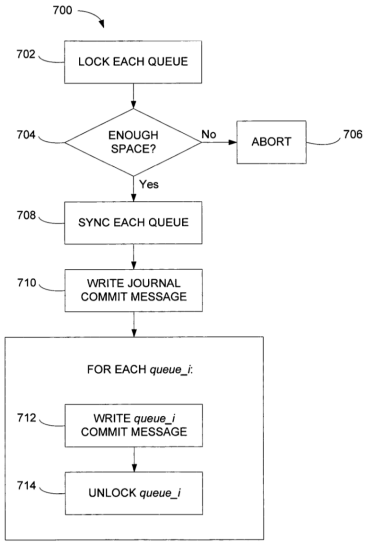
Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	 FIG. 8A Michaylov ’684 at Figure 8A.

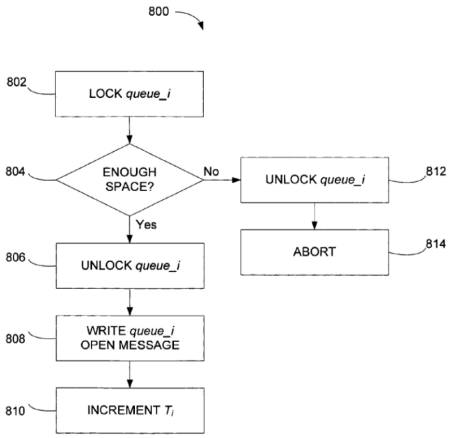
Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	 FIG. 8C Michaylov ’684 at Figure 8C.

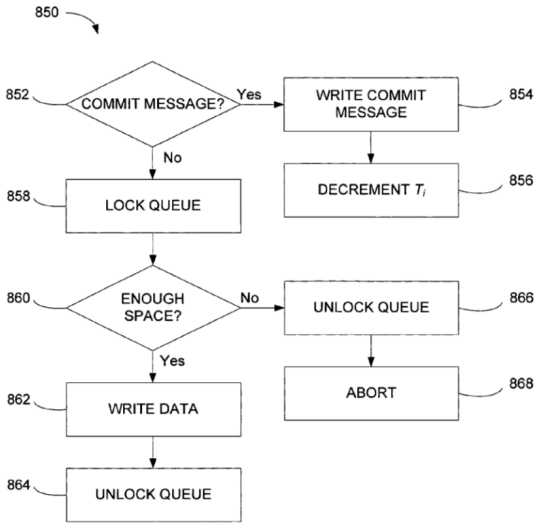
Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	 Michaylov ’684 at Figure 9A.

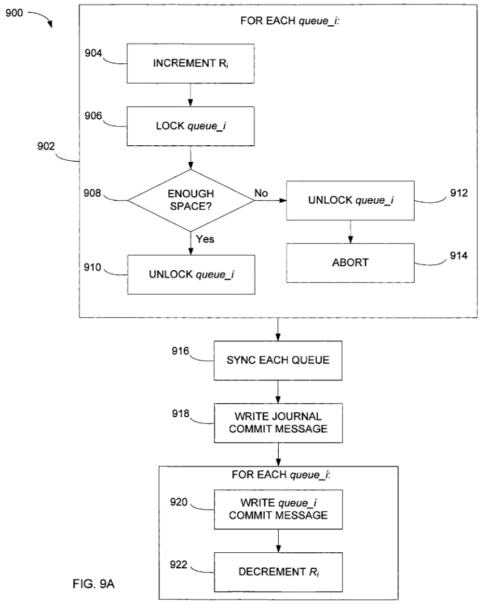
Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	 FIG. 9B Michaylov ’684 at Figure 9B. “FIG. 1A shows a queueing system 100 in which a set of trusted access entities 102A-102M each includes a queue transaction module 104 for interacting directly with a queue manager 106. The queueing system 100 also includes a set of untrusted access entities 108A-108N each including a remote queue transaction module 110 for interacting with the queue manager 106 through a remote procedure call (RPC) manager 112.” Michaylov ’684 at col. 3, ll. 8–15. “To initiate a write transaction, a publisher obtains a transaction identifier (ID) from the queue manager 106 and passes the message or messages to be added to a queue in the write transaction to the queue manager 106. The added messages are associated with the transaction ID of the write transaction in which they were added to the queue. The queue manager 106 passes the messages to the I/O manager 114 to write to volatile memory store 118 and eventually to non-volatile memory store 116. Alternative division of functions performed by the queue manager 106 and the I/O manager 114 can be used.” Michaylov ’684 at col. 4, ll. 8–18. “After the publisher supplies the queue manager 106 all the messages to be added in a write transaction, the publisher may request that the queue manager 106 “commit” or “roll back” the write transaction. In order to commit a write transaction, the queue manager 106 adds a “commit record” to the corresponding queue data

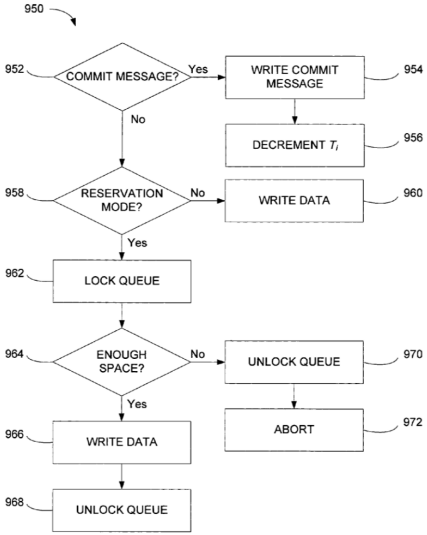
Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	structure in non-volatile memory. The commit record indicates that the messages of a committed write transaction ("committed messages") can be passed to a subscriber. Before a write transaction is committed, the associated messages are made durable by ensuring that they are synchronized from volatile memory to non-volatile memory (if they have not already been synchronized)." Michaylov '684 at col. 4, ll. 19–30. "After locking 702 each queue, the queue manager 106 checks 704 the storage space available in each queue. In the example of FIG. 6, the queue data structure QUEUE_A has a block of storage 600 available and the queue data structure QUEUE_B has a block of storage 602 available. The queue manager 106 compares the amount of space available in each queue with the amount of space that would be used to write a commit record for the compound transaction being committed. If any of the queues does not have enough available space reserved for a commit record, then the queue manager 106 aborts 706 the compound commit operation. The queue manager 106 may attempt the compound commit operation at a later time and/or attempt to acquire more storage space for one or more of the queues." Michaylov '684 at col. 10, ll. 9–22.
[7h] accesses said reservation register and reads said stored value;	Michaylov '684 teaches "accesses said reservation register and reads said stored value," including at least the following exemplary citations. *See* above at row [1g].
[7i] determines when said read value indicates available space in said input queue to issue said command;	Michaylov '684 teaches "determines when said read value indicates available space in said input queue to issue said command," including at least the following exemplary citations. *See* above at row [1h].
[7j] issues said command to said input queue on behalf of said first processing element; and	Michaylov '684 teaches "issues said command to said input queue on behalf of said first processing element," including at least the following exemplary citations. For example, Michaylov '684 teaches that the second processing element executes instructions that causes it to issue said command to said input queue on behalf of said first processing element, by disclosing, as non–limiting examples, the second processing element (e.g., a queue manager) passing the received messages to the queue. The second processing element issues the command to the input queue on behalf of the first processing element.

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	 FIG. 7 Michaylov ’684 at Figure 7.

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	 FIG. 8A Michaylov ’684 at Figure 8A.

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	 FIG. 8C Michaylov ’684 at Figure 8C.

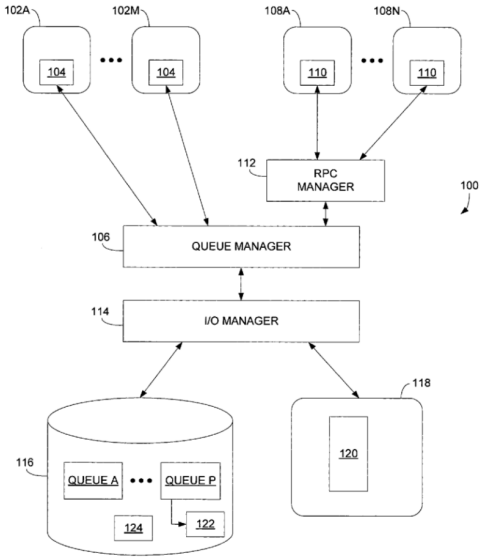
Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	 Michaylov ’684 at Figure 9A.

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	 FIG. 9B Michaylov ’684 at Figure 9B. “To initiate a write transaction, a publisher obtains a transaction identifier (ID) from the queue manager 106 and passes the message or messages to be added to a queue in the write transaction to the queue manager 106. The added messages are associated with the transaction ID of the write transaction in which they were added to the queue. The queue manager 106 passes the messages to the I/O manager 114 to write to volatile memory store 118 and eventually to non-volatile memory store 116. Alternative division of functions performed by the queue manager 106 and the I/O manager 114 can be used.” Michaylov ’684 at col. 4, ll. 8–18. “After the publisher supplies the queue manager 106 all the messages to be added in a write transaction, the publisher may request that the queue manager 106 “commit” or “roll back” the write transaction. In order to commit a write transaction, the queue manager 106 adds a “commit record” to the corresponding queue data structure in non-volatile memory. The commit record indicates that the messages of a committed write transaction (“committed messages”) can be passed to a subscriber. Before a write transaction is committed, the associated messages are made durable by ensuring that they are synchronized from volatile memory to non-volatile memory (if they have not already been synchronized).” Michaylov ’684 at col. 4, ll. 19–30.

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	"If the queues do have enough storage space reserved for commit records, then queue manager 106 synchronizes 708 each queue to ensure that any messages stored in volatile memory have been moved to non-volatile memory. After each queue has been synchronized, the queue manager 106 writes 710 a commit record to the journal data structure 126. After writing the journal commit record, the queue manager 106 writes 712 a commit record to each queue. After writing the commit record to a queue, the queue manager 106 unlocks 714 that queue." Michaylov '684 at col. 10, ll. 23–32. "Referring to FIG. 8B, in the commit record writing operation 840, the queue manager 106 first synchronizes 842 each queue. After synchronizing each queue, the queue manager 106 writes 844 the journal commit record. After writing the journal commit record, the queue manager 106 writes 846 a commit record to each queue." Michaylov '684 at col. 11, ll. 31–36. "Referring to FIG. 8C, the queue manager 106 uses the write operation 850 for data to be written to a queue data structure. The queue manager 106 first determines 852 whether the data to be written is a commit record. If so, the queue manager 106 writes the commit record 854 and decrements 856 the number of outstanding transactions T_i. If not, the queue manager 106 locks 858 the queue and checks 860 the storage space available in the queue. The queue manager 106 compares the current available space with D_w, given by: [Equation] where size_of (data) returns the size of data in the appropriate units (e.g., bytes). If the available space is larger than or equal to D_w, then the queue manager 106 writes 862 the data and unlocks 864 the queue. Otherwise, if the available space is smaller than D_w, then the queue manager 106 unlocks 866 the queue and aborts 868 the write operation 850. In this approach, only a single queue, the queue to which data is being written, is locked while storage space is checked. 3.3 Third Approach" Michaylov '684 at col. 11, ll. 37–58. "After the loop 902, the queue manager 106 synchronizes 916 each queue. After synchronizing each queue, the queue manager 106 writes 918 the journal commit record. After writing the journal commit record, the queue manager 106 writes 920 a commit record to each queue. After writing a commit record to a queue, the queue manager 106 decrements 922 the reservation mode counter for that queue." Michaylov '684 at col. 12, ll. 37–43.
[7k] wherein said first processing element receives a response corresponding to said command from said output queue.	Michaylov '684 teaches "wherein said first processing element receives a response corresponding to said command from said output queue," including at least the following exemplary citations. For example, Michaylov '684 teaches wherein said first processing element receives a response corresponding to said command from said output queue, by disclosing, as non–limiting examples, the first processing element (e.g., an access entity) receiving a response (e.g., abort or queueing of the transaction) or

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	results. The first processing entity receives a response from the output queue corresponding to the command. *See, e.g.:* "In a "compound transaction" an access entity can write to and/or read from more than one queue with all the messages being associated with the same transaction ID. A compound transaction may also be committed or rolled back. When a compound transaction is committed, in a "compound commit" operation, a commit record is added to each queue to which messages are being written in the compound transaction. These "queue commit records" are used to signal that the corresponding committed messages can be passed to a subscriber." Michaylov '684 at col. 4, ll. 56–65. "After locking 702 each queue, the queue manager 106 checks 704 the storage space available in each queue. In the example of FIG. 6, the queue data structure QUEUE_A has a block of storage 600 available and the queue data structure QUEUE_B has a block of storage 602 available. The queue manager 106 compares the amount of space available in each queue with the amount of space that would be used to write a commit record for the compound transaction being committed. If any of the queues does not have enough available space reserved for a commit record, then the queue manager 106 aborts 706 the compound commit operation. The queue manager 106 may attempt the compound commit operation at a later time and/or attempt to acquire more storage space for one or more of the queues." Michaylov '684 at col. 10, ll. 9–22.
8. The system of claim 7, wherein said input queue is a first-in-first-out (FIFO) buffer.	Michaylov '684 teaches "[t]he system of claim 7, wherein said input queue is a first-in-first-out (FIFO) buffer," including at least the following exemplary citations. *See* above at row 2.
9. The system of claim 7, wherein said system is part of a storage area network (SAN) controller.	
10. The system of claim 7, wherein said system is part of a network attached storage (NAS) controller.	

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
11. The system of claim 7, wherein said stored value is decremented by a predefined value prior to returning said stored value to said at least one second processing element.	Michaylov '684 teaches "[t]he system of claim 7, wherein said stored value is decremented by a predefined value prior to returning said stored value to said at least one second processing element," including at least the following exemplary citations. *See* above at row 5.
12. The system of claim 7, wherein at least one third processing element executes said command and notifies said output queue to issue said response.	Michaylov '684 teaches "[t]he system of claim 7, wherein at least one third processing element executes said command and notifies said output queue to issue said response," including at least the following exemplary citations. *See* above at row 6.
[13pre] A system for dividing and synchronizing a processing task across a plurality of processing elements comprising:	Michaylov '684 teaches "[a] system for dividing and synchronizing a processing task across a plurality of processing elements," including at least the following exemplary citations. *See* above at row [1pre].
[13a] an input queue implemented in hardware;	Michaylov '684 teaches "an input queue implemented in hardware," including at least the following exemplary citations. *See* above at row [1a].
[13b] an output queue implemented in hardware;	Michaylov '684 teaches "an output queue implemented in hardware," including at least the following exemplary citations. *See* above at row [1b].
[13c] a first processing element having access to said input queue and said output queue;	Michaylov '684 teaches "a first processing element having access to said input queue and said output queue," including at least the following exemplary citations. *See* above at row [1c].

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
[13d] at least one second processing element in communication with said input queue and said output queue;	Michaylov ’684 teaches “at least one second processing element in communication with said input queue and said output queue,” including at least the following exemplary citations. For example, Michaylov ’684 teaches at least one second processing element in communication with said input queue and said output queue, by disclosing, as non–limiting examples, a second processing element (e.g., a queue manager) interacting with the message queues. A queue manager, the second processing element is in communication with the input and output queue. *See, e.g.:*  FIG. 1A Michaylov ’684 at Figure 1A. “FIG. 1A shows a queueing system 100 in which a set of trusted access entities 102A-102M each includes a queue transaction module 104 for interacting directly with a queue manager 106. The queueing system 100 also includes a set of untrusted access entities 108A-108N each including a remote queue transaction module 110 for interacting with the queue manager 106 through a remote procedure call (RPC) manager 112.” Michaylov ’684 at col. 3, ll. 8–15. “The queueing system 100 provides a mechanism for passing messages between the access entities through one or more message queues. The access entities provide interfaces for modules to interact with the queueing system 100. For example, a “publisher” computational module in a distributed computing system can pass

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	messages including processed data elements to one or more "subscriber" computational modules." Michaylov '684 at col. 3, ll. 16–23. "The queue manager 106 interacts with an input/output (I/O) manager 114 that manages memory storage for a set of message queues with corresponding queue data structures QUEUE_A-QUEUE_P that are each assigned storage space (e.g., a set of disk pages) in a volatile memory store 118 (such as a semiconductor random access memory (RAM)), which is a temporary working storage that provides relatively fast access for reading and writing data. The I/O manager 114 also manages a non-volatile memory store 116 (such as a magnetic disk system), which is a permanent storage that provides relatively more persistence of data and which may provide relatively slower access for reading and writing than the volatile memory store. Optionally, there is a single I/O manager handling the I/O for all the queues, or there are multiple I/O managers running in parallel, each handling the I/O for a subset of the queues." Michaylov '684 at col. 3, ll. 24–39. "The queueing system 100 supports various data distribution models including a publish-subscribe data distribution model. An access entity (trusted or untrusted) acting as a "publisher" to a queue can add one or more messages to the queue (also called a "topic") in a "write transaction." An access entity (trusted or untrusted) acting as a "subscriber" to a queue can read one or more messages from the queue in a "read transaction." Multiple publishers can add messages to the same queue, and multiple subscribers can read the same messages from the same queue. The queue manager 106 removes a message from a queue after all subscribers to the queue have read the message. Alternatively, in a point-to-point data distribution model, multiple access entities can add messages to a queue, but each message is read from the queue by a single access entity. A "compound transaction" includes interaction with more than one queue, as described in more detail below." Michaylov '684 at col. 3, l. 54 – col. 4, l. 3.
[13e] a reservation register implemented in hardware storing a value indicative of available space in said input queue, said reservation register accessible by both said first processing element and said at least one second processing element;	Michaylov '684 teaches "a reservation register implemented in hardware storing a value indicative of available space in said input queue, said reservation register accessible by both said first processing element and said at least one second processing element," including at least the following exemplary citations. *See* above at row [1e].

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
[13f] computer storage storing instructions, which when executed by said at least one second processing element:	Michaylov '684 teaches computer storage storing instructions that are executed by said at least one second processing element including at least the following exemplary citations. *See* above at row [1f].
[13g] receives a command from said first processing element;	Michaylov '684 teaches "receives a command from said first processing element," including at least the following exemplary citations. *See* above at row [7g].
[13h] accesses said reservation register and reads said stored value;	Michaylov '684 teaches "accesses said reservation register and reads said stored value," including at least the following exemplary citations. *See* above at row [1g].
[13i] determines when said read value indicates available space in said input queue to issue said command;	Michaylov '684 teaches "determines when said read value indicates available space in said input queue to issue said command," including at least the following exemplary citations. *See* above at row [1h].
[13j] issues said command to said input queue on behalf of said first processing element; and	Michaylov '684 teaches "issues said command to said input queue on behalf of said first processing element," including at least the following exemplary citations. *See* above at row [7j].
[13k] receives a response corresponding to said command from said output queue.	Michaylov '684 teaches "receives a response corresponding to said command from said output queue," including at least the following exemplary citations. For example, Michaylov '684 teaches that the second processing element executes instructions that causes it to receive a response corresponding to said command from said output queue, by disclosing, as non–limiting examples, the second processing element (e.g., a queue manager) receiving the results from the message queues. The second processing element receives a corresponding response from the output queue. *See, e.g.:* "To initiate a read transaction, a subscriber obtains a transaction ID and receives the next unread message or messages from the queue manager 106. The I/O manager 114 handles retrieving the messages from the

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	appropriate queue data structure and the queue manager 106 passes them to the subscriber. Messages may or may not be passed back in the same order in which they were written to the queue data structure since only committed messages are passed to a subscriber and messages that have been committed may be interleaved with messages that have not yet been committed. The queue manager 106 determines which messages in a queue to pass to the subscriber by building up a "read data structure," as described in more detail below." Michaylov '684 at col. 4, ll. 43–55. "After locking 702 each queue, the queue manager 106 checks 704 the storage space available in each queue. In the example of FIG. 6, the queue data structure QUEUE_A has a block of storage 600 available and the queue data structure QUEUE_B has a block of storage 602 available. The queue manager 106 compares the amount of space available in each queue with the amount of space that would be used to write a commit record for the compound transaction being committed. If any of the queues does not have enough available space reserved for a commit record, then the queue manager 106 aborts 706 the compound commit operation. The queue manager 106 may attempt the compound commit operation at a later time and/or attempt to acquire more storage space for one or more of the queues." Michaylov '684 at col. 10, ll. 9–22.
14. The system of claim 13, wherein said input queue is a first-in-first-out (FIFO) buffer.	Michaylov '684 teaches "[t]he system of claim 13, wherein said input queue is a first-in-first-out (FIFO) buffer," including at least the following exemplary citations. *See* above at row 2.
15. The system of claim 13, wherein said system is either part of a storage area network (SAN) controller.	
16. The system of claim 13, wherein said system is part of a network attached storage (NAS) controller.	
17. The system of claim 13, wherein said stored value is decremented by a predefined value	

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
prior to returning said stored value to said at least one second processing element.	
18. The system of claim 13, wherein at least one third processing element executes said command and notifies said output queue to issue said response.	Michaylov '684 teaches "[t]he system of claim 13, wherein at least one third processing element executes said command and notifies said output queue to issue said response," including at least the following exemplary citations. *See* above at row 6.