DISTRIBUTED LOCK PROCESSING METHOD, DEVICE, STORAGE MEDIUM, AND SYSTEM

[0001] The present disclosure claims priority to Chinese Patent Application No. 202410095739.9, filed with the China National Intellectual Property Administration on January 23, 2024, and entitled "DISTRIBUTED LOCK PROCESSING METHOD, DEVICE, STORAGE MEDIUM, AND SYSTEM", which is incorporated herein by reference in its entirety.

TECHNICAL FIELD

[0002] The present disclosure relates to the field of cloud computing technologies and, in particular, to a distributed lock processing method, a device, a storage medium, and a system.

BACKGROUND

[0003] ZooKeeper is an open-source distributed coordination service, and has been widely applied in many large-scale distributed systems. For example, a ZooKeeper service system can be applied in an application scenario of a distributed lock service. The ZooKeeper service system includes a plurality of nodes (znodes), and can tolerate simultaneous failures of some nodes. For example, a system consisting of five nodes can tolerate simultaneous failures of two nodes.

[0004] Some software-level faults may cause the entire ZooKeeper service system to be unavailable. For example, there are the following situations: (1) A leader node hangs (i.e., suddenly gets stuck during running and stops running subsequent codes), but a heartbeat mechanism between the leader node and a follower node still runs normally. In this case, the follower node considers that the leader node is still normal and does not trigger a new round of leader node election. Consequently, the service system remains persistently unavailable in this case. (2) It is assumed that the service system includes three nodes, and it is assumed that a machine on which one follower node is located becomes unavailable due to a hardware failure, and the leader node Leader falls into deadlock upon exiting (for example, a transaction ID is exhausted, or a heartbeat between the leader node and the follower node expires). In this case, two of the three nodes have software failures, causing the entire service system to be

unavailable.

[0005] Therefore, in an application scenario using the ZooKeeper service system, for example, the application scenario of the distributed lock service, high availability of the distributed lock service needs to be ensured.

SUMMARY

[0006] Embodiments of the present disclosure provide a distributed lock processing method, a device, a storage medium, and a system, to improve high availability of a distributed lock service.

[0007] According to a first aspect, an embodiment of the present disclosure provides a distributed lock processing system. The system includes:

a first service cluster and a second service cluster that provide a distributed lock service, and a client using the distributed lock service, where

the client is configured to: sequentially acquire a first session lease from the first service cluster and acquire a second session lease from the second service cluster; send a first request for lock creation to the first service cluster by using the first session lease and send a second request for lock creation to the second service cluster by using the second session lease; and upon receiving lock creation success messages respectively sent by the first service cluster and the second service cluster, determine that a lock is acquired, where the first session lease and the second session lease are respectively bound to set validity periods;

the first service cluster is configured to: in response to the first request, create the lock for the client when it is determined that no other client occupies the lock, and send a lock creation success message to the client; and

the second service cluster is configured to: in response to the second request, create the lock for the client when it is determined that no other client occupies the lock, and send a lock creation success message to the client.

[0008] According to a second aspect, an embodiment of the present disclosure provides a distributed lock processing method, applied to a client using a distributed lock service. The method includes:

acquiring a first session lease from a first service cluster, and sending a first request for lock creation to the first service cluster by using the first session lease;

acquiring a second session lease from a second service cluster, and sending a second request for lock creation to the second service cluster by using the second session lease, where

both the first service cluster and the second service cluster provide the distributed lock service, and the first session lease and the second session lease are respectively bound to set validity periods; and

determining that the lock is acquired by the client upon receiving a lock creation success message sent by the first service cluster and a lock creation success message sent by the second service cluster.

[0009] According to a third aspect, an embodiment of the present disclosure provides a distributed lock processing apparatus, used in a client using a distributed lock service. The apparatus includes:

a first creation module, configured to: acquire a first session lease from a first service cluster, and send a first request for lock creation to the first service cluster by using the first session lease;

a second creation module, configured to: acquire a second session lease from a second service cluster, and send a second request for lock creation to the second service cluster by using the second session lease, where both the first service cluster and the second service cluster provide the distributed lock service, and the first session lease and the second session lease are respectively bound to set validity periods; and

a determining module, configured to determine that the lock is acquired by the client upon receipt of a lock creation success message sent by the first service cluster and a lock creation success message sent by the second service cluster.

[0010] According to a fourth aspect, an embodiment of the present disclosure provides an electronic device, including a memory, a processor, and a communication interface, where the memory stores executable codes, and the executable codes, when executed by the processor, causes the processor to implement at least the distributed lock processing method according to the second aspect.

[0011] According to a fifth aspect, an embodiment of the present disclosure provides a non-transitory machine-readable storage medium, where the non-transitory machine-readable storage medium stores executable codes, and the executable codes, when executed by a processor of an electronic device, causes the processor to implement at least the distributed lock processing method according to the second aspect.

[0012] According to a sixth aspect, an embodiment of the present disclosure provides a computer program product, including a computer program, where the computer program, when executed by a processor, implements at least the distributed lock processing method according to the second aspect.

[0013] The distributed lock processing system provided in this embodiment of the present disclosure includes at least the first service cluster and the second service cluster that provide the distributed lock service, and further includes the client using the distributed lock service. A lock creation (in other words, lock contention) process for a client is as follows: The client sequentially acquires the first session lease from the first service cluster and acquires the second session lease from the second service cluster, and sends the first request for lock creation to the first service cluster by using the first session lease and sends the second request for lock creation to the second service cluster by using the second session lease. In response to the first request, the first service cluster creates the lock for the client when it is determined that no other client occupies the lock, and sends the lock creation success message to the client. Similarly, in response to the second request, the second service cluster creates the lock for the client when it is determined that no other client occupies the lock, and sends the lock creation success message to the client. Only when receiving the two lock creation success messages, the client determines that the client successfully acquires the lock. It can be learned that, each service cluster maintains a lock, and only when the client sequentially acquires locks from two service clusters, the client can determine that the client successfully acquires the lock. In other words, if the client fails to create a lock in a service cluster, which means that another client occupies the lock provided by the service cluster at this time, the client finally fails to acquire the lock. Therefore, higher robustness of a lock is achieved by increasing complexity of a lock contention phase, so that high availability of a distributed lock service continues to be maintained when a single service cluster is entirely unavailable.

BRIEF DESCRIPTION OF DRAWINGS

[0014] To describe the technical solutions in the embodiments of the present disclosure more clearly, the following briefly introduces the accompanying drawings required for describing the embodiments. Apparently, the accompanying drawings described below show some embodiments of the present disclosure, and a person of ordinary skill in the art may still derive other drawings from these accompanying drawings without creative efforts.

[0015] FIG. 1 is a schematic diagram of a lock contention process for a conventional distributed lock according to an embodiment of the present disclosure.

[0016] FIG. 2 is a schematic diagram of a dual-lock mechanism according to an embodiment of the present disclosure.

[0017] FIG. 3 is a schematic diagram of a hardware execution environment for a distributed

lock processing system according to an embodiment of the present disclosure.

[0018] FIG. 4 is a schematic diagram of a cloud computing environment for a distributed lock processing system according to an embodiment of the present disclosure.

[0019] FIG. 5 is a schematic diagram of an execution process for a distributed lock processing method according to an embodiment of the present disclosure.

[0020] FIG. 6 is a schematic diagram in which a client observes a server that fails to be a leader in a dual-lock mechanism according to an embodiment of the present disclosure.

[0021] FIG. 7 is a schematic diagram of a session lease timing design in a primary-secondary lock mechanism according to an embodiment of the present disclosure.

[0022] FIG. 8 is a schematic diagram of a delayed heartbeat response scenario according to an embodiment of the present disclosure.

[0023] FIG. 9 is a flowchart of a distributed lock processing method according to an embodiment of the present disclosure.

[0024] FIG. 10 is a schematic structural diagram of a distributed lock processing apparatus according to an embodiment of the present disclosure.

[0025] FIG. 11 is a schematic structural diagram of an electronic device according to an embodiment of the present disclosure.

DESCRIPTION OF EMBODIMENTS

[0026] To make the objectives, technical solutions, and advantages of the embodiments of the present disclosure clearer, the following clearly and completely describes the technical solutions in the embodiments of the present disclosure with reference to the accompanying drawings in the embodiments of the present disclosure. Apparently, the described embodiments are some embodiments of the present disclosure rather than all of the embodiments. All other embodiments obtained by a person of ordinary skill in the art based on the embodiments of the present disclosure without creative efforts shall fall within the protection scope of the present disclosure.

[0027] It should be noted that, user information (including, but not limited to, user equipment information, user personal information, and the like) and data (including, but not limited to, data for analysis, stored data, displayed data, and the like) involved in the embodiments of the present disclosure are information and data that are authorized by a user or that are sufficiently authorized by all parties. Related data needs to be collected, used, and processed in accordance with related laws and regulations and standards of related regions, and

a corresponding operation entry is provided for a user to select to grant or deny authorization.

[0028] Some implementations of the present disclosure are described in detail below with reference to the accompanying drawings. The following embodiments and features in the embodiments may be mutually combined when there is no conflict between the various embodiments. In addition, a sequence of steps in the following method embodiments is merely an example, and is not strictly limited.

[0029] Terms or concepts involved in the embodiments of the present disclosure are first explained and described:

[0030] A distributed lock is a mechanism that is in a distributed system and that is used for mutually exclusive access to shared resources. The mechanism ensures that only one client can obtain the ownership of a lock at any time point, thereby ensuring correctness of concurrent operations.

[0031] A session lease is a session protocol between a client and a distributed lock service. The client creates a session lease when contending for a lock, and a validity period of the session lease determines a validity period of the lock.

[0032] An ephemeral node is a node that is in a ZooKeeper service cluster and that is used to represent the distributed lock. A lifecycle of the ephemeral node is bound to a session lease. When a session of a client holding a lock expires or terminates, the ephemeral node is automatically deleted.

[0033] Availability of the distributed lock: When an entire cluster providing the distributed lock service is unavailable, the session lease quickly expires, causing the client that has acquired the lock to enter a lock loss logic and re-contend for a lock. Because the cluster is unavailable, a lock contention operation continuously fails, and consequently, the client contending for the lock is in an unavailable state. That the entire cluster is unavailable may be that a majority of nodes in the cluster are unavailable. For example, a ZooKeeper service cluster includes three ZooKeeper nodes (znodes), and when at least two of the ZooKeeper nodes are unavailable, the entire ZooKeeper cluster is unavailable.

[0034] Service discovery refers to automatically discovering and identifying an available service instance in the distributed system, so that another service or client can interact with the service instance.

[0035] Currently, providing the distributed lock service based on, for example, a ZooKeeper distributed cluster, has been widely used. In this case, three or five nodes are usually deployed in a common ZooKeeper cluster. When a majority of the nodes are unavailable, the entire ZooKeeper cluster is unavailable, and consequently can no longer provide the distributed

lock service normally. A distributed lock processing solution provided in an embodiment of the present disclosure aims to improve high availability of a distributed lock service.

[0036] The following first briefly describes, with reference to FIG. 1, a lock contention (in other words, lock creation) process for a conventional distributed lock and a related maintenance process.

[0037] The distributed lock has two basic attributes: ownership and a validity period. As shown in FIG. 1, a ZooKeeper distributed lock is used as an example. It is assumed that client 1 and client 2 contend for a same lock in a same ZooKeeper cluster (ZooKeeper cluster 1 in FIG. 1), and each client creates a session lease, where a session lease of client 1 is represented as *lease1*, and a session lease of client 2 is represented as *lease2*. A lock is an ephemeral node, and a lifecycle of the lock is bound to a session lease. For ease of understanding, the lock is referred to as *abc* in this embodiment.

[0038] It is assumed that the lock creation process is as follows: 1. Client 1 first establishes a communication connection (for example, a TCP (Transmission Control Protocol) connection) to ZooKeeper cluster 1. ZooKeeper cluster 1 creates a piece of identification information of a session layer for client 1, that is, the session lease (*lease1*), binds the session lease to a corresponding validity period, and feeds back the session lease to client 1. Then, client 1 and ZooKeeper cluster 1 perform heartbeat maintenance based on a heartbeat mechanism. 2. Client 1 sends a request for lock creation to ZooKeeper cluster 1 by using *lease1* as an owner identifier. Assuming that in this case, ZooKeeper cluster 1 finds that lock *abc* is not occupied by another client, ZooKeeper cluster 1 creates an ephemeral node corresponding to lock *abc*, and associates the ephemeral node with the session lease *lease1* of client 1, which indicates that client 1 successfully acquires the lock in this case, where a validity period is the validity period corresponding to *lease1*. Then, ZooKeeper cluster 1 feeds back a lock creation success message to client 1, and client 1 determines that client 1 successfully acquires the lock.

[0039] In addition, when client 1 performs the foregoing lock creation process, it is assumed that client 2 also performs a lock creation process. As shown in FIG. 1: 1. Client 2 first establishes a TCP connection to ZooKeeper cluster 1. ZooKeeper cluster 1 creates a piece of identification information of a session layer for client 2, that is, the session lease (*lease2*), binds the session lease to a corresponding validity period, and feeds back the session lease to client 2. Then, client 2 and ZooKeeper cluster 1 perform heartbeat maintenance based on the heartbeat mechanism. 2. Client 2 sends a request for lock creation to ZooKeeper cluster 1 by using *lease2* as an owner identifier. In this case, because ZooKeeper cluster 1 finds that lock *abc* is being occupied by client 1, ZooKeeper cluster 1 feeds back a lock creation failure

message to client 2, and client 2 determines that client 2 fails to acquire the lock.

[0040] It can be learned that only one client can acquire the lock at one time point, to satisfy a mutual exclusion requirement of the distributed lock. In addition, after a session heartbeat of a client holding a lock stops, a session lease of the client expires within a preset time period, the lock occupied by the client is also automatically released accordingly, and another surviving client successfully acquires the lock. This ensures liveness of the lock.

[0041] For example, in this embodiment shown in FIG. 1, when a heartbeat between client 1 and ZooKeeper cluster 1 stops at a time point after client 1 successfully acquires the lock (for example, client 1 cannot send a heartbeat packet to the ZooKeeper cluster at a set heartbeat sending interval), a time point at which the last response message of the heartbeat packet is received is used as a timing start point, and when the timing reaches the validity period corresponding to *lease1*, *lease1* of client 1 expires, and lock *abc* is actively released. In this case, ZooKeeper cluster 1 deletes *lease1* and the associated lock *abc*. As shown in FIG. 1: 3. It is assumed that a heartbeat between client 2 and ZooKeeper cluster 1 remains normal, and client 2 keeps attempting to create a lock by using its own *lease2*. Therefore, when ZooKeeper cluster 1 receives the request for lock creation from client 2, because no other client occupies lock *abc* at this time, lock *abc* is allocated to client 2 for use. Specifically, ZooKeeper cluster 1 creates an ephemeral node for lock *abc* and associates the ephemeral node with *lease2* of client 2, where a validity period is the validity period of *lease2*.

[0042] In the foregoing process, a lease renewal problem is actually involved. For ease of description, client 1 is used as an example. After establishing the TCP connection to ZooKeeper cluster 1 and being allocated with *lease1*, client 1 can send the heartbeat packet to ZooKeeper cluster 1 at a set time interval, and ZooKeeper cluster 1 feeds back a corresponding response message after receiving the heartbeat packet, so that client 1 and ZooKeeper cluster 1 determine that the communication connection between client 1 and ZooKeeper cluster 1 is normal. Assuming that client 1 is notified by ZooKeeper cluster 1 that the validity period of *lease1* is 20 seconds, client 1 starts timing of the validity period after receiving the lock creation success message fed back by ZooKeeper cluster 1. In this case, it is assumed that the timing start point is a time point *T1*. When client 1 sends a next heartbeat packet to ZooKeeper cluster 1, and receives a corresponding response message (where it is assumed that this time point is *T2*), the timing start point is reset to *T2*. In this case, it is equivalent to renewing *lease1*. Assuming that client 1 subsequently sends no heartbeat packet or receives no corresponding response message, timing starts from T2, and it is determined that *lease1* expires after 20 seconds.

[0043] In the example shown in FIG. 1, the ownership and the validity period of the

distributed lock are both reflected by using the session lease, in other words, the ownership and the validity period are coupled together. Consequently, an availability problem is generated. If ZooKeeper cluster 1 providing the distributed lock service is entirely unavailable, the session lease quickly expires, and the client acquiring the ownership of the lock quickly enters a lock loss logic, which triggers lock re-contention. However, because ZooKeeper cluster 1 is unavailable, lock contention of each client continuously fails, causing each client to remain in a persistent unavailable state.

[0044] "Lock contention" is a low-frequency operation. Therefore, a natural idea is whether it is possible, in this scenario, to ensure that maintenance of a validity period of an existing distributed lock is not affected while only an entirely new lock contention operation is affected. In this way, a failure blast radius of a cluster providing the distributed lock service can be effectively controlled. Based on this idea, an idea provided in this embodiment of the present disclosure is to increase difficulty in lock contention, to further relax a threshold for maintaining a validity period of a lock. In other words, robustness of the lock is achieved by increasing complexity of lock contention.

[0045] Based on this, this embodiment of the present disclosure provides a highly-available distributed lock processing solution that is based on a primary-secondary lock renewal mechanism, to continue to maintain high availability of a lock service when a single ZooKeeper cluster is entirely unavailable. A primary-secondary lock design is introduced based on the ZooKeeper cluster. In this way, higher robustness of the lock is achieved by increasing complexity of a lock contention phase. The ZooKeeper cluster is merely an example of a distributed system, and shall not be construed as a limitation thereto.

[0046] As shown in FIG. 2, in general, a quantity of service clusters providing a distributed lock service is increased from 1 to at least 2, for example, ZooKeeper cluster 1 and ZooKeeper cluster 2 in FIG. 2. Each distributed lock is split into two locks: a primary lock and a secondary lock, and the two locks are respectively maintained in two independent ZooKeeper clusters. Actually, the primary lock and the secondary lock are merely intended to distinguish between locks located in two clusters, and names thereof may be the same, for example, are both referred to as lock *abc*. In this case, to contend for ownership of the lock, a client needs to sequentially acquire ownership of both the primary lock and the secondary lock. Mutual exclusion of the distributed lock is embodied in a fact that at most one client can occupy the two locks in sequence at the same time. A validity period of the distributed lock is a larger one of validity periods of session leases respectively corresponding to the two locks. In other words, as long as either lock is still within a validity period, the client can consider that the lock is not lost,

and can continue to occupy the lock for operation. On the contrary, a prerequisite for the client to contend for the distributed lock becomes more stringent: The validity periods of both locks expire.

[0047] The following describes a distributed lock processing solution provided in this embodiment of the present disclosure.

[0048] FIG. 3 is a schematic diagram of a hardware execution environment for a distributed lock processing system according to an embodiment of the present disclosure. As shown in FIG. 3, the hardware execution environment for the distributed lock processing system may include a client device 301, a first service cluster 302, and a second service cluster 303. The client device 301 is communicatively connected to the first service cluster 302 and the second service cluster 303.

[0049] The first service cluster 302 and the second service cluster 303 are different service clusters providing a same distributed lock service. Each service cluster may include a plurality of nodes. For example, the first service cluster 302 includes node *a1*, node *a2*, and node *a3*, and the second service cluster 303 includes node *b1*, node *b2*, and node *b3*. During actual application, the two service clusters may be specifically implemented as a first ZooKeeper cluster and a second ZooKeeper cluster. Each ZooKeeper cluster includes three ZooKeeper nodes, and the three nodes externally provide the distributed lock service. In this embodiment of the present disclosure, a leader-follower role of each node in each ZooKeeper cluster may not be distinguished.

[0050] During actual application, the first service cluster 302 and the second service cluster 303 may be two service clusters formed by cloud-based cloud servers, and different nodes in a same service cluster may be deployed on different cloud servers.

[0051] The client device 301 refers to a client using the distributed lock service provided by the first service cluster 302 and the second service cluster 303, and generally refers to any client using the distributed lock service. During actual application, the client device 301 is usually a server providing an application/content service, for example, a server providing a service such as e-commerce, audio, video, or artificial intelligence.

[0052] During actual application, nodes included in the first service cluster 302 and the second service cluster 303 may be cloud servers maintained by a cloud service provider, which are referred to as compute nodes. A cloud computing environment shown in FIG. 4 may include several distributed compute nodes (401-1, 401-2, ... shown in FIG. 4), and each compute node has processing resources such as computing and storage resources. In the cloud computing environment, a plurality of compute nodes may be organized to provide a service. Certainly,

one compute node may also provide one or more services, for example, service A, service B, service C, and service D shown in FIG. 4. Service B is, for example, a distributed lock service. In other words, a service other than the distributed lock service may be included in a compute node. In the cloud computing environment, the service may be provided by externally providing a service interface, and the client device calls the service interface to use a corresponding service. The service interface is implemented in forms such as a software development kit (SDK) and an application programming interface (API).

[0053] During operation, executing a request from the client device may require calling of one or more services in the cloud computing environment, and executing one or more functions of one service may require calling of one or more functions of another service. As shown in FIG. 4, after receiving the request sent by the client device, service A may call service B, and service B may request service D to perform one or more functions.

[0054] Each of the first service cluster 302 and the second service cluster 303 may include a plurality of compute nodes providing the distributed lock service. FIG. 4 merely shows a situation in which one compute node 401-1 provides the distributed lock service. Actually, a plurality of compute nodes providing the distributed lock service may form a service cluster, and a plurality of other compute nodes providing the distributed lock service form another service cluster.

[0055] The distributed lock service is deployed based on various virtualization technologies supported by the cloud computing environment, for example, virtual machine-based or container-based virtualization technologies. The container-based virtualization technology is used as an example. Several containers corresponding to one service may be assembled into one pod. For example, service B (the distributed lock service) shown in FIG. 4 may be configured with one or more pods, and each pod may include a proxy and one or more containers. The one or more containers in the pod are configured to process a request related to one or more corresponding functions of the service, and the proxy in the pod is configured to control network functions related to the service, such as routing and load balancing.

[0056] Based on the distributed lock processing system shown in FIG. 3, a process in which the client device 301 contends for a lock, in other words, creates the lock, by using the first service cluster 302 and the second service cluster 303 is as follows.

[0057] The client sequentially acquires a first session lease from the first service cluster and acquires a second session lease from the second service cluster, sends a first request for lock creation to the first service cluster by using the first session lease and sends a second request for lock creation to the second service cluster by using the second session lease, and

upon receiving lock creation success messages respectively sent by the first service cluster and the second service cluster, determines that a lock is acquired, where the first session lease and the second session lease are respectively bound to set validity periods.

[0058] In response to the first request, the first service cluster creates the lock for the client when it is determined that no other client occupies the lock, and sends a lock creation success message to the client.

[0059] In response to the second request, the second service cluster creates the lock for the client when it is determined that no other client occupies the lock, and sends a lock creation success message to the client.

[0060] A lock loss process is as follows: The client determines to discard the lock, if it is determined, based on the validity periods of the first session lease and the second session lease, that a target session lease expires, where the target session lease is a session lease that expires later between the first session lease and the second session lease.

[0061] The validity period of the first session lease is the same as that of the second session lease.

[0062] Lock contention and lock loss processes are described with reference to FIG. 5 by using an example.

[0063] The lock contention process is as follows: 1. Client 1 establishes a TCP connection to ZooKeeper cluster 1. ZooKeeper cluster 1 creates a first session lease (*lease1*) for client 1, binds the first session lease to a corresponding validity period, and feeds back the first session lease to client 1. Then, client 1 and ZooKeeper cluster 1 perform heartbeat maintenance based on a heartbeat mechanism. Client 1 sends a first request for lock creation to ZooKeeper cluster 1 by using *lease1* as an owner identifier. Assuming that in this case, ZooKeeper cluster 1 finds that a local lock *abc* is not occupied by another client, ZooKeeper cluster 1 creates an ephemeral node corresponding to lock *abc*, and associates the ephemeral node with the session lease *lease1* of client 1, where a validity period is a validity period corresponding to *lease1*. Then, ZooKeeper cluster 1 feeds back a lock creation success message to client 1.

[0064] Then, 2. client 1 establishes a TCP connection to ZooKeeper cluster 2. ZooKeeper cluster 2 creates a second session lease (*lease2*) for client 1, binds the second session lease to a corresponding validity period, and feeds back the second session lease to client 1. Then, client 1 and ZooKeeper cluster 2 perform heartbeat maintenance based on the heartbeat mechanism. Client 1 sends a second request for lock creation to ZooKeeper cluster 2 by using *lease2* as an owner identifier. In this case, because ZooKeeper cluster 2 finds that a local lock abc is not occupied by another client, ZooKeeper cluster 2 feeds back a lock creation success message to

client 1.

[0065] Because client 1 sequentially receives the lock creation success messages from both ZooKeeper cluster 1 and ZooKeeper cluster 2, client 1 determines that client 1 successfully acquires the lock.

[0066] Then, as shown in FIG. 5, 3. when a heartbeat between client 1 and ZooKeeper cluster 1 stops/expires at a time point after client 1 successfully acquires the lock, when the timing reaches the validity period of *lease1*, ZooKeeper cluster 1 determines that client 1 loses the lock and deletes a record of client 1 occupying the local lock *abc*. However, it is assumed that in this case, a heartbeat between client 1 and ZooKeeper cluster 2 remains normal, in other words, the local lock *abc* of ZooKeeper cluster 2 is still occupied by client 1. In this case, because *lease2* has not expired, client 1 still occupies lock *abc*. At the same time, 4. assuming that client 2 establishes a TCP connection to ZooKeeper cluster 1, ZooKeeper cluster 1 creates a third session lease (*lease3*) for client 2, binds the third session lease to a corresponding validity period, and feeds back the third session lease to client 2. Then, client 2 and ZooKeeper cluster 1 perform heartbeat maintenance based on the heartbeat mechanism. Client 2 sends a request for lock creation to ZooKeeper cluster 1 by using *lease3* as an owner identifier. In this case, because ZooKeeper cluster 1 finds that the local lock *abc* is not occupied by another client, ZooKeeper cluster 1 creates an ephemeral node corresponding to lock *abc*, associates the ephemeral node with the session lease *lease3* of client 2, and feeds back a lock creation success message to client 2. Then, client 2 establishes a TCP connection to ZooKeeper cluster 2. ZooKeeper cluster 2 creates a fourth session lease (*lease4*) for client 2, binds the fourth session lease to a corresponding validity period, and feeds back the fourth session lease to client 2. Then, client 2 and ZooKeeper cluster 2 perform heartbeat maintenance based on the heartbeat mechanism. 5. Client 2 sends a request for lock creation to ZooKeeper cluster 2 by using *lease4* as an owner identifier. In this case, ZooKeeper cluster 2 finds that the local lock *abc* is still occupied by client 1, and therefore, determines that client 2 fails to acquire the lock, and feeds back a lock contention failure message to client 2.

[0067] It may be understood that, in the foregoing example, if the heartbeat between client 1 and ZooKeeper cluster 2 also expires, causing timing of the validity period corresponding to *lease2* to end, client 1 determines that the occupied lock *abc* is discarded.

[0068] The foregoing heartbeat expiration/stopping refers to a situation in which a communication connection anomaly is detected between a client and a ZooKeeper cluster based on the heartbeat mechanism. The heartbeat expiration refers to, for example, that the client does not receive a response message fed back by the ZooKeeper cluster within a set time

period after sending a heartbeat packet. The heartbeat stopping refers to, for example, that the client does not send a heartbeat packet after a set time period.

[0069] It can be learned that in a dual-lock mechanism of a primary lock and a secondary lock (where for example, in the foregoing example, one of the locks *abc* locally maintained by the two ZooKeeper clusters is the primary lock and the other one is the secondary lock), a condition for determining successful lock contention is simultaneous acquisition of both locks, which is equivalent to concurrently holding validity periods of the two locks. Therefore, the primary lock and the secondary lock are actually equivalent. Switching an acquisition sequence of the primary lock and the secondary lock does not affect mutual exclusion of the lock.

[0070] Based on the foregoing primary-secondary lock mechanism, high availability of a distributed lock service can be achieved. For a client, when either the primary lock or the secondary lock expires, a session lease for the other lock is continuously renewed (as long as the heartbeat is normal), preventing the lock from being lost and allowing the client to continue operating. However, if there is only one set of distributed lock service, when a corresponding service cluster is unavailable, the client discards the only lock (for example, a situation of the embodiment shown in FIG. 1). The client needs to instantly contend for a lock again. Moreover, because the only service cluster is unavailable, the client continuously fails to acquire the lock, leaving the client in an operational standstill.

[0071] It should be noted that in this embodiment of the present disclosure, a processing process of a distributed lock is merely described at a service cluster granularity. Actually, the foregoing ZooKeeper cluster 1 is used as an example. ZooKeeper cluster 1 may include a plurality of nodes. When the client establishes the TCP connection to ZooKeeper cluster 1, the client actually establishes a TCP connection to a node in ZooKeeper cluster 1. Successfully creating a lock by the client means successfully creating a lock on the node. For clients using the distributed lock service, nodes in one service cluster may be set to be equivalent and do not need to be distinguished.

[0072] The foregoing embodiment describes a solution in which an acquisition sequence of the primary lock and the secondary lock does not need to be distinguished (in other words, a sequence of acquiring locks from two service clusters may be set by default, to be specific, it may be set to first acquire a lock from a first service cluster and then acquire a lock from a second service cluster, and it may also be set to first acquire the lock from the second service cluster and then acquire the lock from the first service cluster). The solution may be applicable to a scenario in which different clients using the distributed lock service have equal roles. For example, clients are storage nodes having a same role in a distributed storage system. However,

in an application scenario of leader election and service discovery, the acquisition sequence of the primary lock and the secondary lock needs to be distinguished.

[0073] In the application scenario of leader election and service discovery, a plurality of clients using the distributed lock service determine leader and follower roles through lock contention. A client successfully acquiring a lock is a leader client, and other clients are backup clients. Moreover, the leader client successfully acquiring the lock registers a service address (for example, an IP (internet protocol) address) of the leader client with a service cluster providing the distributed lock service, to facilitate visitors to discover the service address that the visitors need to access. In this case, the client using the distributed lock service may be represented as a server (that is, an application server such as an e-commerce server or an audio/video server), and a party accessing the application server may be represented as a client (that is, an application client). Therefore, leader election refers to determining a leader server and a backup server among a plurality of servers.

[0074] In the scenario of leader election and service discovery, after a server completes leader contention through a distributed lock, the server needs to register a service address of the server with the service cluster (such as a ZooKeeper cluster) providing the distributed lock service, to facilitate access by the client. Therefore, in an optional embodiment, the server may write the service address of the server into the service cluster while contending for the lock. In this way, lock contention and service address registration are completed simultaneously. In this application scenario, if the foregoing dual-lock mechanism is used and the service address is registered through the primary lock, a situation in which the client detects a service address registered by a backup server that fails in leader contention, but the backup server actually does not start to operate yet may exist in some scenarios.

[0075] The following describes the foregoing situation with reference to FIG. 6 by using an example. In FIG. 6, it is assumed that server1 successfully acquires a lock from ZooKeeper cluster 1 and a lock from ZooKeeper cluster 2 sequentially through steps 1 and 2 shown in the figure (where for a specific process, refer to related descriptions in the embodiment shown in FIG. 5, and details are not described herein again), and it is assumed that the lock acquired from ZooKeeper cluster 1 is referred to as a primary lock, and the lock acquired from ZooKeeper cluster 2 is referred to as a secondary lock. Actually, the primary lock and the secondary lock are a same lock, for example lock *abc* in the foregoing example, stored in the two service clusters. In addition, when server1 acquires the primary lock, server1 registers a service address of server1 with ZooKeeper cluster 1. Therefore, as shown in FIG. 6, a client can find the service address of server1 through ZooKeeper cluster 1.

[0076] Then, server1 starts to operate after acquiring the primary and secondary locks. After a period of time, as shown in FIG. 6, assuming that a heartbeat between server1 and ZooKeeper cluster 1 expires, in other words, heartbeat maintenance for the primary lock expires, server2 acquires the primary lock from ZooKeeper cluster 1 in this case, and registers a service address of server2 with ZooKeeper cluster 1. In this case, because server1 still maintains a session lease of the secondary lock, that is, the secondary lock remains within a validity period, server2 fails to acquire the secondary lock from ZooKeeper cluster 2 (where for details, refer to related descriptions in the embodiment shown in FIG. 5, and details are not described herein again). In other words, server1 is still occupying the lock at this time, and what the client observes, from ZooKeeper cluster 1, should be the service address of currently operating server1. However, because server2 successfully acquires the primary lock from ZooKeeper cluster 1 and registers the service address of server2 with ZooKeeper cluster 1, ZooKeeper cluster 1 considers that server2 is a leader server at this time, and replaces the previously registered service address of server1 with the service address of server2. In this case, what the client observes, from ZooKeeper cluster 1, is the service address of server2. This thus generates a problem of leader election inconsistency: The leader server that should be elected is server1, but the service address is actually registered by backup server2.

[0077] To resolve the foregoing problem of leader election inconsistency in the scenario of leader election and service discovery, compared with the foregoing embodiment (in which the first service cluster and the second service cluster do not distinguish between the primary lock and the secondary lock: by default, the first service cluster is set as a primary service cluster and provides the primary lock, and the second service cluster is set as a secondary service cluster and provides the secondary lock, and by default, the server first acquires the primary lock and then acquires the secondary lock, and registers the service address with the primary service cluster, namely, the first service cluster, after successfully acquiring the primary lock), this embodiment of the present disclosure provides the following method for resolving leader election inconsistency: A default acquisition sequence of the primary lock and the secondary lock is changed, so that a server contends for the primary lock after holding ownership of the secondary lock.

[0078] Specifically, the second service cluster is configured as a service cluster providing a service address registration service (different from the foregoing embodiment, a lock provided by the second service cluster is referred to as the primary lock herein), and the first service cluster is configured as a service cluster providing the secondary lock. Actually, because the primary lock and the secondary lock are merely nominally distinct instances of a same lock

located in different service clusters, and may alternatively be collectively referred to as a first lock and a second lock, the solution is essentially as follows: The server first contends for a lock (the secondary lock) provided by the first service cluster that does not provide the service address registration service; then, the server contends for the lock (the primary lock) provided by the second service cluster that provides the service address registration service, and registers the service address of the server with the second service cluster after successfully acquiring the primary lock.

[0079] For each server, by default, a service cluster providing a service address registration function is configured as the second service cluster. Therefore, a client (namely, the server) using a distributed lock service registers a service address with the second service cluster when receiving a lock creation success message sent by the second service cluster, so that an access object (the client) of the server acquires the service address of the server from the second service cluster.

[0080] In the foregoing method for resolving the problem of leader election inconsistency, a situation shown in FIG. 6 in which server1 and server2 contend for the lock is still used as an example. Server1 first successfully acquires the lock from ZooKeeper cluster 1, and then contends for the lock from ZooKeeper cluster 2. If server1 also successfully acquires the lock from ZooKeeper cluster 2, server1 registers its own service address with ZooKeeper cluster 2. In this case, the client observes the service address of server1 through ZooKeeper cluster 2. Then, it is assumed that server1 releases the lock acquired from ZooKeeper cluster 1 (for example, due to heartbeat stopping or heartbeat expiration), but still holds the lock acquired from ZooKeeper cluster 2. In this case, server2 successfully acquires the lock from ZooKeeper cluster 1, but when server2 attempts to contend for the lock from ZooKeeper cluster 2, server2 fails to acquire the primary lock because server1 occupies the lock. Consequently, the client still observes, from ZooKeeper cluster 2, the service address of server1 that still holds the lock.

[0081] It should be noted that if the secondary lock is acquired first, to ensure that the session lease of the secondary lock remains valid when the primary lock is successfully acquired, it may be ensured, before attempting to contend for the primary lock, that a remaining validity period of the secondary lock is greater than time required for acquiring the primary lock, so that the server can wait for return of a message of successfully acquiring the primary lock. In an extreme situation, if after receiving the message of successfully acquiring the primary lock fed back by the second service cluster, the server finds that the session lease of the secondary lock expires, the server needs to actively release the primary lock and enter a logic for contending for the secondary lock. In this way, it can be ensured that the client obtains

a service address of a real leader server through parsing.

[0082] The foregoing embodiment provides a situation in which a distributed lock processing system includes only two service clusters: the first service cluster and the second service cluster that provide the distributed lock service. In practice, to further improve high availability of the distributed lock service, a quantity of the service clusters providing the distributed lock service may further be increased from 2 to a larger number, for example, 3 or 5.

[0083] In other words, in addition to the first service cluster and the second service cluster, the distributed lock processing system further includes at least one other service cluster providing the distributed lock service. In this case, the client using the distributed lock service determines, only when receiving lock creation success messages sent by at least a target quantity of service clusters, that the client acquires the lock. The target quantity is greater than half of a quantity of service clusters.

[0084] In this case, security and liveness of the expanded distributed lock are mainly embodied as follows:

[0085] Security of the distributed lock: After a client acquires lock resources of a majority of service clusters, it is considered that the lock is successfully acquired, and if renewal of the majority expires, it is considered that the lock is lost. The majority refers to more than half of all service clusters. For example, if there are three service clusters in total, the majority refers to at least two of the three service clusters. For example, if there are five service clusters in total, the majority refers to at least three of the five service clusters.

[0086] Liveness of the distributed lock: (1) A deadlock does not occur. Lock resources of a service cluster automatically expires and is recycled. Each client contends for the lock resources of each service cluster in a fixed order. If the acquisition fails, the client skips to a next service cluster to acquire the lock resources. After each failed lock contention, the client needs back-off time longer than time required for successfully acquiring a majority of locks, so that a client that has successfully acquired the lock resources preferentially contends for subsequent lock resources. (2) Downtime of a single service cluster does not affect availability of the entire distributed lock.

[0087] It is mentioned in the foregoing embodiment that a session lease serves as an identifier for the client using the distributed lock service at a session layer. In addition, the session lease is further bound to a validity period. When the validity period expires, an occupied lock is released. This relates to a design of a timing mechanism for the validity period of the session lease.

[0088] In a conventional distributed lock solution, for example, a RedLock algorithm in Redis, a timing method based on absolute time such as a physical clock is used, for example, network time protocol (NTP) time. The physical clock drift may occur. To be specific, the physical clock drift between the service cluster providing the distributed lock service and the client using the distributed lock service may occur, and the occurrence of clock drift may violate mutual exclusion of the lock.

[0089] First, it should be noted that the service cluster and the client both need to time a validity period of a same session lease. For example, if a session lease corresponding to a client has a validity period of 20 seconds, the client and the service cluster both need to time the validity period of the session lease. Therefore, for example, when a physical clock of the service cluster drifts forward (to a time point in the future, which means that time runs excessively fast), the service cluster considers that the validity period of the local session lease expires prematurely (that is, the lock prematurely expires). However, at this time, the validity period of the session lease of the client does not expire, which means that the client considers that the client still holds the lock. In this case, another client contends for the lock from the service cluster again. Because the service cluster considers that the session lease of the previous client expires, the service cluster allocates the lock to the another client. Consequently, the lock is simultaneously held by two clients, breaking mutual exclusion of the lock. For another example, the physical clock of the client drifts backward (to a time point in the past, which means that time runs slowly). In this case, the client considers that the client still holds the lock (the validity period of the local session lease does not expire), but the service cluster determines that the client loses the lock (the validity period of the local session lease expires). Similarly, this leads to the foregoing situation in which a plurality of clients simultaneously holds the lock, breaking mutual exclusion of the lock. In both of the foregoing examples, the session lease of the client expires later than the session lease of the service cluster, resulting in violation of mutual exclusion of the lock.

[0090] To overcome impact of the timing method of the physical clock on mutual exclusion performance of the lock, in a primary-secondary lock design provided in this embodiment of the present disclosure, the timing mechanism for the validity period of the session lease does not rely on the physical clock, but relies on a phenomenon of clock frequency consistency (within a short period of time). In other words, any two machines can be considered to experience same elapsed time over a same duration. Based on this premise, in the foregoing primary-secondary lock mechanism, the session leases of both the client and the service cluster utilize monotonic relative time instead of absolute time. In this way, accuracy of session lease

maintenance can be ensured, and it can be ensured that the session lease of the client expires earlier than the session lease of the service cluster. If the client considers that the session lease maintained locally expires, but the session lease of the service cluster does not expire, the client releases the occupied lock, and the service cluster considers that the client still occupies the lock and rejects a lock creation request (that is, a lock contention request) of another client. The lock is occupied by only one client at one time point, and mutual exclusion of the lock is not violated.

[0091] During specific implementation, both the service cluster and the client may use a CPU (Central Processing Unit) tick (CPU tick) as a relative time timing method to time the validity period of the session lease. In practice, MonotonicTime (monotonic time) may be determined based on the CPU tick, and timing of the validity period of the session lease is implemented by counting MonotonicTime. For a method of determining MonotonicTime based on the CPU tick, refer to existing related technologies for implementation.

[0092] Continuing the foregoing embodiment, for the client, the client times the validity period of the first session lease based on the CPU tick by using a time point at which a lock creation success message sent by the first service cluster is received as a first timing start point, and times the validity period of the second session lease based on the CPU tick by using a time point at which a lock creation success message sent by the second service cluster is received as a second timing start point.

[0093] For the first service cluster and the second service cluster, the first service cluster resets a third timing start point of the validity period of the first session lease maintained locally to a time point at which an initial heartbeat packet sent by the client is received, times, based on the CPU tick, the validity period of the first session lease maintained locally, and deletes the lock and the first session lease when it is determined that the first session lease locally maintained expires. Similarly, the second service cluster resets a fourth timing start point of the validity period of the second session lease maintained locally to a time point at which an initial heartbeat packet sent by the client is received, times, based on the CPU tick, the validity period of the second session lease maintained locally, and deletes the lock and the second session lease when it is determined that the second session lease maintained locally expires. It may be understood that, after receiving the lock creation success message fed back by the first service cluster, the client sends the initial heartbeat packet (referring to a heartbeat packet initially sent to the first service cluster) to the first service cluster, and when receiving the initial heartbeat packet, the first service cluster restarts local timing of the validity period of the first session lease. Similarly, after receiving the lock creation success message fed back by the second

service cluster, the client sends the initial heartbeat packet (referring to a heartbeat packet initially sent to the second service cluster) to the second service cluster, and when receiving the initial heartbeat packet, the second service cluster restarts local timing of the validity period of the second session lease.

[0094] Timing start points of the validity periods of the first session lease and the second session lease respectively corresponding to the first service cluster and the second service cluster are respectively: a time point at which the first service cluster creates the first session lease and a time point at which the second service cluster creates the second session lease.

[0095] In addition, only when the client determines, through local timing, that both the first session lease and the second session lease expire, the client determines to release the lock, and enters a next lock contention logic.

[0096] In addition, it should be noted that, the client sends a heartbeat packet to the first service cluster at a set time interval from the first timing start point, and resets, if a response message fed back by the first service cluster is received, the first timing start point to a time point at which the response message is received; and sends a heartbeat packet to the second service cluster at a set time interval from the second timing start point, and resets, if a response message fed back by the second service cluster is received, the second timing start point to a time point at which the response message is received. It may be understood that resetting of the first and second timing start points is equivalent to restarting timing of the validity periods of the corresponding session leases from new timing start points, which effectively achieves renewal of the session leases. Certainly, normal receiving and sending of a next heartbeat packet and a next response message also cause resetting of a timing start point again.

[0097] Similarly, in a process of locally timing the validity periods of the first session lease and the second session lease, the first service cluster and the second service cluster also reset the local third timing start point and the local fourth timing start point each time the heartbeat packet sent by the client is received and the corresponding response message is fed back.

[0098] The following describes a timing mechanism for a validity period of a session lease with reference to FIG. 7. As shown in FIG. 7, when duration of a validity period of a session lease on a client side is consistent with duration of a validity period of a same session lease on a service cluster side, in the foregoing primary-secondary lock mechanism, a client starts timing of a validity period of a first session lease (which is indicated as *lease1* in the figure) when receiving a lock creation success message fed back by ZooKeeper cluster 1 (which is used as a first timing start point in this case), and starts timing of a validity period of a second session lease (which is indicated as *lease2* in the figure) when receiving a lock creation success

message fed back by ZooKeeper cluster 2 (which is used as a second timing start point in this case). Both ZooKeeper cluster 1 and ZooKeeper cluster 2 start timing of corresponding validity periods of *lease1* and *lease2* after receiving the respective initial heartbeat packets sent by the client (respectively corresponding to a third timing start point and a fourth timing start point in the figure). Therefore, timing start points of the client, ZooKeeper cluster 1 and ZooKeeper cluster 2 are guaranteed to have a sequence due to causality (where the client sends the initial heartbeat packets after receiving lock creation success messages). In addition, through monotonic relative time timing (to be specific, a CPU tick-based timing method), it can be guaranteed that, when the client no longer initiates renewal (to be specific, the client cannot perform normal receiving/transmitting of a heartbeat packet), the session lease of the client always expires before expiration of the validity period of the session lease of the service cluster. For example, as shown in FIG. 7, it is assumed that the client no longer sends next heartbeat packets after sending the initial heartbeat packets to ZooKeeper cluster 1 and ZooKeeper cluster 2 respectively. When timing of duration of the validity period of *lease1* that is timed from the first timing start point reaches a time point T1, the client determines that the validity period of *lease1* expires, and when timing of duration of the validity period of *lease2* that is timed from the second timing start point reaches a time point T1', the client determines that the validity period of *lease2* expires. Because ZooKeeper cluster 1 does not receive a heartbeat packet again, when timing of duration of the validity period of *lease1* that is timed from the third timing start point reaches a time point T2, it is determined that the validity period of *lease1* expires. Similarly, because ZooKeeper cluster 2 does not receive a heartbeat packet again, when timing of duration of the validity period of *lease2* that is timed from the fourth timing start point reaches a time point T3, it is determined that the validity period of *lease2* expires. In addition, T1 is earlier than T2, and T1' is earlier than T3. Therefore, based on a relative time timing method, an objective is achieved in which the validity period of the session lease of the client expires earlier than the validity period of the session lease of the corresponding service cluster.

[0099] In the foregoing embodiments, it is assumed that a validity period of a session lease on the client is equal to that on each service cluster side. However, during actual application, network latency between the client and a first service cluster as well as a second service cluster may sometimes be significant. In an extreme case, for example, when the client sends a heartbeat packet to the first service cluster, the first service cluster may feed back a response message after a delay that lasts for the duration of the validity period of the first session lease. In this case, the client considers that the client still holds a lock (because the response message

is received within the local validity period of the first session lease of the client, which indicates successful renewal), but the first service cluster determines that the client loses the lock (because local timing of the validity period of the first session lease at the first service cluster expires). As shown in FIG. 8, a client sends a heartbeat packet to ZooKeeper cluster 1 at a time point T1. It is assumed that network latency is extremely low and can be ignored. In other words, it is assumed that ZooKeeper cluster 1 receives the heartbeat packet at the time point T1. Then, assuming that the client receives a response message sent by ZooKeeper cluster 1 at a time point T2 after a delay that lasts for one full validity period of *lease1*, the client determines at this point that renewal of *lease1* is successful. In addition, it is assumed that the client immediately sends a next heartbeat packet to ZooKeeper cluster 1. In this case, as shown in FIG. 8, ZooKeeper cluster 1 starts timing at the time point T1, and determines that *lease1* of the client expires when a timing duration reaches the time point T2, where the timing duration is equivalent to one full validity period of *lease1*. In this case, ZooKeeper cluster 1 disconnect a communication connection to the client, so that the foregoing "next heartbeat packet" cannot be received, causing the client to fail to renew *lease1* and lose a lock.

[0100] To avoid this situation, a first service cluster may determine that a validity period of a first session lease maintained locally is at least twice of the validity period of the first session lease maintained by the client. Similarly, a second service cluster determines that a validity period of a second session lease maintained locally is at least twice of the validity period of the second session lease maintained by the client.

[0101] That is to say, for the first session lease (for example, *lease1*) corresponding to the client, the first service cluster notifies the client that the validity period of the first session lease is, for example, 20 seconds, but the first service cluster locally records that the validity period of the first session lease is 40 seconds (a case of twice the validity period). Similarly, for the second session lease (for example, *lease2*) corresponding to the client, the second service cluster notifies the client that the validity period of the second session lease is, for example, 20 seconds, but the second service cluster locally records that the validity period of the second session lease is 40 seconds (a case of twice the validity period). In this way, in the foregoing example, even if the first service cluster sends a response message of the heartbeat packet to the client after it is determined through local timing that half of the 40-second validity period of the first session lease has passed (reaching 20 seconds), the first service cluster determines that the client still holds the lock because the 40-second validity period has not yet expired, which matches a case in which the first session lease of the client is successfully renewed.

[0102] In conclusion, with a primary-secondary lock design, timing of a validity period of

a session lease does not rely on a physical clock but on trusted relative time. This avoids a situation caused by physical clock drift in which a session lease of the client expires later than a session lease of the service cluster. In addition, by setting a validity period of the session lease of the service cluster to be at least twice a validity period of the session lease of the client, impact of the network latency on renewal of the session lease of the client can be reduced.

[0103] FIG. 9 is a flowchart of a distributed lock processing method according to an embodiment of the present disclosure. The method may be performed by the client using a distributed lock service in the foregoing embodiments. As shown in FIG. 9, the method includes the following steps:

[0104] 901: A client acquires a first session lease from a first service cluster, and sends a first request for lock creation to the first service cluster by using the first session lease.

[0105] 902: The client acquires a second session lease from a second service cluster, and sends a second request for lock creation to the second service cluster by using the second session lease, where both the first service cluster and the second service cluster provide a distributed lock service, and the first session lease and the second session lease are respectively bound to set validity periods.

[0106] 903: The client determines that the client acquires a lock upon receiving a lock creation success message sent by the first service cluster and a lock creation success message sent by the second service cluster.

[0107] In an optional embodiment, the method further includes: the client determines to discard the lock, if it is determined, based on the validity periods of the first session lease and the second session lease, that a target session lease expires, where the target session lease is a session lease that expires later between the first session lease and the second session lease.

[0108] In an optional embodiment, regarding timing of a validity period of a session lease, the client times the validity period of the first session lease based on a CPU tick by using a time point at which the lock creation success message sent by the first service cluster is received as a first timing start point, and times the validity period of the second session lease based on the CPU tick by using a time point at which the lock creation success message sent by the second service cluster is received as a second timing start point.

[0109] In an optional embodiment, in a scenario of leader election and service discovery, the second service cluster is configured to provide a service address registration service. Therefore, in a process in which the client first creates the lock (in other words, contends for the lock) from the first service cluster and then creates the lock from the second service cluster, the client registers a service address with the second service cluster upon receiving the lock

creation success message sent by the second service cluster, so that an access object of the client acquires the service address of the client from the second service cluster.

[0110] The following describes in detail a distributed lock processing apparatus according to one or more embodiments of the present disclosure. A person skilled in the art may understand that these apparatuses may be constructed by using commercially available hardware components and configured according to steps in this solution.

[0111] FIG. 10 is a schematic structural diagram of a distributed lock processing apparatus according to an embodiment of the present disclosure. The apparatus is used in a client using a distributed lock service. As shown in FIG. 10, the apparatus includes a first creation module 11, a second creation module 12, and a determining module 13.

[0112] The first creation module 11 is configured to: acquire a first session lease from a first service cluster, and send a first request for lock creation to the first service cluster by using the first session lease.

[0113] The second creation module 12 is configured to: acquire a second session lease from a second service cluster, and send a second request for lock creation to the second service cluster by using the second session lease, where both the first service cluster and the second service cluster provide the distributed lock service, and the first session lease and the second session lease are respectively bound to set validity periods.

[0114] The determining module 13 is configured to determine that a lock is acquired by the client upon receipt of a lock creation success message sent by the first service cluster and a lock creation success message sent by the second service cluster.

[0115] The apparatus shown in FIG. 10 can perform steps performed by the client in the foregoing embodiments. For detailed execution processes and technical effects, refer to descriptions in the foregoing embodiments, and details are not described herein again.

[0116] In a possible design, a structure of the distributed lock processing apparatus shown in FIG. 10 may be implemented as an electronic device. As shown is FIG. 11, the electronic device may include a processor 21, a memory 22, and a communication interface 23. The memory 22 stores executable codes, and the executable codes, when executed by the processor 21, causes the processor 21 to implement at least the distributed lock processing method provided in the foregoing embodiments.

[0117] In addition, an embodiment of the present disclosure provides a non-transitory machine-readable storage medium, where the non-transitory machine-readable storage medium stores executable codes, and the executable codes, when executed by a processor of an electronic device, causes the processor to implement at least the distributed lock processing

method provided in the foregoing embodiments.

[0118] An embodiment of the present disclosure provides a computer program product, including a computer program, where the computer program, when executed by a processor, implements at least the distributed lock processing method provided in the foregoing embodiments.

[0119] The apparatus embodiments described above are merely exemplary. The network elements described as separate parts may or may not be physically separate. Some or all of the modules may be selected according to actual needs to achieve the objectives of the solutions of the embodiments. A person of ordinary skill in the art may understand and implement the embodiments without creative efforts.

[0120] Through the descriptions of the foregoing implementations, a person skilled in the art may clearly understand that the implementations may be implemented by a necessary universal hardware platform, or may certainly be implemented by a combination of hardware and software. Based on this understanding, the essence of the foregoing technical solutions, or the parts contributing to the prior art, may be embodied in a form of a computer product. The present disclosure may use a form of a computer program product implemented on one or more computer-usable storage media (including but not limited to a disk memory, a CD-ROM (Compact Disc Read-Only Memory), an optical memory, and the like) including computer-usable program codes.

[0121] Finally, it should be noted that the foregoing embodiments are merely used for describing the technical solutions of the present disclosure, but are not intended to limit the present disclosure. Although the present disclosure is described in detail with reference to the foregoing embodiments, it should be appreciated by a person of ordinary skill in the art that, modifications may still be made to the technical solutions recorded in the foregoing embodiments, or equivalent replacements may be made to some technical features thereof; and these modifications or replacements will not cause the essence of corresponding technical solutions to depart from the spirit and scope of the technical solutions in the embodiments of the present disclosure.

CLAIMS

1. A distributed lock processing system, comprising:

a first service cluster and a second service cluster that provide a distributed lock service, and a client using the distributed lock service, wherein

the client is configured to: sequentially acquire a first session lease from the first service cluster and acquire a second session lease from the second service cluster; send a first request for lock creation to the first service cluster by using the first session lease and send a second request for lock creation to the second service cluster by using the second session lease; and upon receiving lock creation success messages respectively sent by the first service cluster and the second service cluster, determine that a lock is acquired, wherein the first session lease and the second session lease are respectively bound to set validity periods;

the first service cluster is configured to: in response to the first request, create the lock for the client when it is determined that no other client occupies the lock, and send a lock creation success message to the client; and

the second service cluster is configured to: in response to the second request, create the lock for the client when it is determined that no other client occupies the lock, and send a lock creation success message to the client.

2. The system according to claim 1, wherein the client is configured to determine to discard the lock, if it is determined, based on the validity periods of the first session lease and the second session lease, that a target session lease expires, wherein the target session lease is a session lease that expires later between the first session lease and the second session lease.

3. The system according to claim 1 or 2, wherein the client is configured to: time a validity period of the first session lease based on a central processing unit (CPU) tick by using a time point at which the lock creation success message sent by the first service cluster is received as a first timing start point, and time a validity period of the second session lease based on the CPU tick by using a time point at which the lock creation success message sent by the second service cluster is received as a second timing start point.

4. The system according to claim 3, wherein the client is configured to: send a heartbeat packet to the first service cluster at a set time interval from the first timing start point, and if a response message fed back by the first service cluster is received, reset the first timing start point to a time point at which the response message fed back by the first service cluster is received; and send a heartbeat packet to the second service cluster at a set time interval from

the second timing start point, and if a response message fed back by the second service cluster is received, reset the second timing start point to a time point at which the response message fed back by the second service cluster is received.

5. The system according to any one of claims 1 to 4, wherein the first service cluster is configured to determine that a validity period of the first session lease maintained locally is at least twice a validity period of the first session lease maintained by the client; and

the second service cluster is configured to determine that a validity period of the second session lease maintained locally is at least twice a validity period of the second session lease maintained by the client.

6. The system according to claim 5, wherein the first service cluster is configured to: reset a third timing start point of the validity period of the first session lease maintained locally to a time point at which an initial heartbeat packet sent by the client is received, time, based on a CPU tick, the validity period of the first session lease maintained locally, and delete the lock and the first session lease when it is determined that the first session lease locally maintained expires; and

the second service cluster is configured to: reset a fourth timing start point of the validity period of the second session lease maintained locally to a time point at which an initial heartbeat packet sent by the client is received, time, based on the CPU tick, the validity period of the second session lease maintained locally, and delete the lock and the second session lease when it is determined that the second session lease maintained locally expires.

7. The system according to any one of claims 1 to 6, wherein the second service cluster is configured to provide a service address registration service; and

the client is configured to register a service address with the second service cluster upon receiving the lock creation success message sent by the second service cluster, so that an access object of the client acquires the service address of the client from the second service cluster.

8. The system according to any one of claims 1 to 6, wherein the system further comprises at least one other service cluster providing the distributed lock service; and

the client is configured to: after sending a request for lock creation to each service cluster, determine that the client acquires the lock upon receiving lock creation success messages sent by at least a target quantity of service clusters, wherein the target quantity is greater than half of a quantity of service clusters.

9. A distributed lock processing method, applied to a client using a distributed lock service, comprising:

acquiring a first session lease from a first service cluster, and sending a first request for

lock creation to the first service cluster by using the first session lease;

acquiring a second session lease from a second service cluster, and sending a second request for lock creation to the second service cluster by using the second session lease, wherein both the first service cluster and the second service cluster provide the distributed lock service, and the first session lease and the second session lease are respectively bound to set validity periods; and

determining that a lock is acquired by the client upon receiving a lock creation success message sent by the first service cluster and a lock creation success message sent by the second service cluster.

10. The method according to claim 9, wherein the method further comprises:

determining to discard the lock, if it is determined, based on the validity periods of the first session lease and the second session lease, that a target session lease expires, wherein the target session lease is a session lease that expires later between the first session lease and the second session lease.

11. The method according to claim 10, wherein the method further comprises:

timing a validity period of the first session lease based on a central processing unit (CPU) tick by using a time point at which the lock creation success message sent by the first service cluster is received as a first timing start point; and

timing a validity period of the second session lease based on the CPU tick by using a time point at which the lock creation success message sent by the second service cluster is received as a second timing start point.

12. The method according to any one of claims 9 to 11, wherein the second service cluster is configured to provide a service address registration service; and

the method further comprises:

registering a service address with the second service cluster upon receiving the lock creation success message sent by the second service cluster, so that an access object of the client acquires the service address of the client from the second service cluster.

13. An electronic device, comprising: a memory, a processor, and a communication interface, wherein the memory stores executable codes, and the executable codes, when executed by the processor, causes the processor to perform the distributed lock processing method according to any one of claims 9 to 12.

14. A non-transitory machine-readable storage medium, wherein the non-transitory machine-readable storage medium stores executable codes, and the executable codes, when executed by a processor of an electronic device, causes the processor to perform the distributed

lock processing method according to any one of claims 9 to 12.

15. A computer program product, comprising a computer program, wherein the computer program, when executed by a processor, implements the method according to any one of claims 9 to 12.