

SERVER APPARATUS, SYSTEM, AND METHOD FOR PROCESSING RESOURCE INVOCATION REQUESTS

TECHNICAL FIELD

[0001] Various aspects of this disclosure relate to a server apparatus, system, and method for processing resource invocation requests.

BACKGROUND

[0002] The following discussion of the background is intended to facilitate an understanding of the present disclosure only. It should be appreciated that the discussion is not an acknowledgment or admission that any of the material referred to was published, known, or is part of the common general knowledge of the person skilled in the art in any jurisdiction as of the priority date of the disclosure.

[0003] With the rapid advancement of computer application technologies, numerous services now rely on external resources to perform their functions effectively. These resources, often sourced from third parties, may be used for enhancing system capabilities, optimizing costs, and addressing data security concerns. However, such resources frequently impose limitations on the number of concurrent accesses they can handle. If these limits are exceeded, resource invocation failures occur, potentially disrupting the operation of the services that depend on them.

[0004] A common example can be found in intelligent voice services, where third-party resources are integrated to improve functionality and efficiency. When the number of concurrent accesses to these resources surpasses the allowed threshold, the service may malfunction or become entirely inoperable. This creates a critical challenge in maintaining system stability, highlighting the need for effective mechanisms to monitor and control concurrent access to external resources.

[0005] Currently, in distributed high-concurrency scenarios, if concurrency control is not implemented for resource invocations, the actual number of concurrent users may exceed the available quantity. On the other hand, attempting to control concurrency by obtaining the current concurrent count in real-time may lead to contention among multiple invocation operations, impacting system performance.

[0006] Existing concurrency control solutions also have issues with accurately controlling the number of concurrent accesses in abnormal situations, thereby affecting normal usage.

[0007] Accordingly, the disclosure seeks to provide a server apparatus, system, and method for processing data, to ameliorate the aforementioned drawback at least in part.

SUMMARY

[0008] The disclosure seeks to provide a system, server apparatus, and method for processing resource invocation requests/data.

[0009] According to an aspect of the present disclosure there is provided a server apparatus, the server apparatus comprises a processor, the processor configured to: obtain a resource invocation request, the resource invocation request associated with a target resource; obtain a first real-time concurrent count of the target resource; increase the first real-time concurrent count by a first predetermined number to obtain a first updated real-time concurrent count; compare the first updated real-time concurrent count with a first threshold; wherein in a positive determination that the first updated real-time concurrent count is greater than the first threshold, obtain a second real-time concurrent count of the target resource; subtract a second predetermined number from the second real-time concurrent count to obtain a second updated concurrent count, and reject the resource invocation request.

[0010] In some embodiments, in a positive determination that the first updated concurrent count is less than or equal to the first threshold, the processor is configured to accept the resource invocation request and create an invocation session (e.g. for a resource requestor) to invoke the target resource.

[0011] In some embodiments, the processor is configured to assign the real-time concurrent count as the first updated real-time concurrent count.

[0012] In some embodiments, if the first real-time concurrent count is not obtained or is an invalid number, the processor is configured to initialize the first real-time concurrent count to an initial value.

[0013] In some embodiments, the processor is configured to reserve the use of the target resource after the increase of the first real-time concurrent count by the first predetermined number.

[0014] In some embodiments, the second predetermined number is the same as the first predetermined number, or the second predetermined number may be different from the first predetermined number.

[0015] In some embodiments, the processor is configured to determine a failure status of the creation of the invocation session.

[0016] In some embodiments, in the event the invocation session fails, the processor is configured to obtain a third real-time concurrent count, and subtract a third predetermined number from the third real-time concurrent count to obtain a third updated concurrent count.

[0017] In some embodiments, the processor is configured to create a concurrency information table for storage of a first key-value pair, the first key-value pair comprises a unique identifier associated with the target resource as the key, and a real-time concurrent count of the target resource as the value.

[0018] In some embodiments, the processor is configured to store at least a second key-value pair and a third key-value pair; wherein the second key-value pair comprises the unique identifier associated with the target resource as the key, and a session identifier as the value; and wherein the third key-value pair comprises the unique identifier associated with the target resource as the key, and a time stamp associated with a start time of the session as the value.

[0019] In some embodiments, the processor is configured to determine a success status of the creation of the invocation session, wherein in a positive determination that the invocation session is successful, the processor is configured to determine one or more states of the invocation session.

[0020] In some embodiments, if the one or more states is indicative of a completion of the invocation session, the processor is configured to obtain a fourth real-time concurrent count of the target resource, and subtract or decrease a fourth predetermined value from the fourth real-time concurrent count to obtain a fourth updated concurrent count.

[0021] In some embodiments, the processor is configured to create a session identifier, the session identifier comprises a first state indicative of the completion of the invocation session and a second state indicative of a resource utilization state.

[0022] In some embodiments, the processor is configured to compare a usage duration of the invocation session with a predetermined duration, and wherein where the usage duration is greater than the predetermined duration, the processor is configured to obtain a fifth real-time concurrent count of the target resource, and subtract or decrease a fifth predetermined value from the fifth real-time concurrent count to obtain a fifth updated concurrent count.

[0023] According to another aspect of the present disclosure there is provided a system for processing resource invocation requests, the system comprises a target resource provider; the server apparatus as defined, the server apparatus arranged in data communication with the

target resource provider to determine whether to reject or accept the resource invocation request and to create the invocation session for a resource requestor to invoke the target resource in a positive determination that the resource invocation request is accepted.

[0024] In some embodiments, the system may comprise a communication interface configured to create the invocation session for a resource requestor to invoke the target resource.

[0025] In some embodiments, the target resource is an external resource of a distributed system, or an internal resource of the distributed system.

[0026] In another aspect of the present disclosure there is provided a method for processing resource invocation requests, the method comprises: obtaining a resource invocation request, the resource invocation request associated with a target resource; obtaining a first real-time concurrent count of the target resource; increasing the first real-time concurrent count by a first predetermined number to obtain a first updated real-time concurrent count; comparing the first updated real-time concurrent count with a first threshold; wherein in a positive determination that the first updated real-time concurrent count is greater than the first threshold, obtaining a second real-time concurrent count of the target resource; subtracting a second predetermined number from the second real-time concurrent count to obtain a second updated concurrent count, and rejecting the resource invocation request.

[0027] According to another aspect of the present disclosure there is provided a non-transitory computer-readable medium storing computer executable code comprising instructions for processing data according to any one of the described methods.

BRIEF DESCRIPTION OF THE DRAWINGS

[0028] The disclosure will be better understood with reference to the detailed description when considered in conjunction with the non-limiting examples and the accompanying drawings, in which:

- FIG. 1A is a schematic block diagram of a server apparatus for processing resource invocation requests.
- FIG. 1B is a schematic block diagram comprising the server apparatus of FIG. 1A, and one or more target resource providers, implemented as a system for processing resource invocation requests.

- FIG. 2 is a flow chart depicting a method of responding to a resource requestor's invocation request for a target resource.
- FIG. 3 is a flow chart illustrating a resource invocation control method involving a session information management table in the form of a concurrency information table.
- FIG. 4 is a flow chart illustrating a session state management mechanism to control concurrency adjustments based on determining the usage status of an invocation (call) session in the case of successful invocation (call) session creation.
- FIG. 5 is a flow chart illustrating a session state management mechanism based on state identifiers employed to control the adjustment of concurrent counts.
- FIG. 6 is a flow chart illustrating a periodic detection mechanism employed to improve reliability of concurrency control, which includes periodically detecting a usage duration of created invocation sessions, in order to optimally handle potential timeout anomalies in session calls.
- FIG. 7 is a general flow chart of a method for processing resource invocation requests according to the present disclosure.

DETAILED DESCRIPTION

[0029] The following detailed description refers to the accompanying drawings that show, by way of illustration, specific details and embodiments in which the disclosure may be practiced. These embodiments are described in sufficient detail to enable those skilled in the art to practice the disclosure. Other embodiments may be utilized and structural, and logical changes may be made without departing from the scope of the disclosure. The various embodiments are not necessarily mutually exclusive, as some embodiments can be combined with one or more other embodiments to form new embodiments.

[0030] Embodiments, which are non-limiting examples, described in the context of one of the enclosure systems, server devices, or methods are analogously valid for the other systems, devices, or methods. Similarly, embodiments described in the context of a system are analogously valid for a device or a method, and vice-versa.

[0031] Features that are described in the context of an embodiment may correspondingly be applicable to the same or similar features in the other embodiments. Features that are described in the context of an embodiment may correspondingly be applicable to the other embodiments, even if not explicitly described in these other embodiments. Furthermore, additions and/or

combinations and/or alternatives as described for a feature in the context of an embodiment may correspondingly be applicable to the same or similar feature in the other embodiments.

[0032] In the context of various embodiments, the articles “a”, “an” and “the” as used with regard to a feature or element include a reference to one or more of the features or elements. Furthermore, as used in the present disclosure and the appended claims, the term “by” may also mean “from”, depending on the context. Furthermore, as used in the present disclosure and the appended claims, the term “if” may also mean “when” or “upon”, depending on the context. Furthermore, as used in the present disclosure and the appended claims, the words “and/or” may refer to and encompass any and all possible combinations of one or more of the associated listed items.

[0033] As used herein, the term “data” may be understood to include information in any suitable analogue or digital form, for example, provided as a file, a portion of a file, a set of files, a signal or stream, a portion of a signal or stream, a set of signals or streams, waveforms, and the like. The term data, however, is not limited to the aforementioned examples and may take various forms and represent any information as understood in the art.

[0034] As used herein, the term “first”, “second”, “third”, “fourth”, “fifth”, etc. are used to distinguish one element/feature from another, and, unless otherwise stated, may not denote order, priority or sequence.

[0035] As used herein, the term “module” refers to, forms part of, or includes an application Specific Integrated Circuit (ASIC); an electronic circuit; a combinational logic circuit; a field programmable gate array (FPGA); a processor (shared, dedicated, or group) that executes code; other suitable hardware components that provide the described functionality; or a combination of some or all of the above, such as in a system-on-chip. The term module may include memory (shared, dedicated, or group) that stores code executed by the processor. A single module or a combination of modules may be regarded as a device. A processor may include one or more modules. For example, multiple modules described in this disclosure may form a processor.

[0036] As used herein, the term “associate”, “associated”, and “associating” indicate a defined relationship (or cross-reference) between two items.

[0037] As used herein, the term “obtain”, in the context of obtaining data, broadly include pull technology used any time a transfer of data is initiated by a request sent from a client to a server. Push technology, on the other hand, is implemented any time a transfer of information is initiated by a server without waiting on a request from a client. In some embodiments, the term obtain may include receive.

[0038] As used herein, “memory” may be understood as a non-transitory computer-readable medium in which data or information can be stored. References to “memory” included herein may thus be understood as referring to volatile or non-volatile memory, including random access memory (“RAM”), read-only memory (“ROM”), flash memory, solid-state storage, magnetic tape, hard disk drive, optical drive, etc., or any combination thereof. Furthermore, it is appreciated that registers, shift registers, processor registers, data buffers, etc., are also embraced herein by the term memory. It is appreciated that a single component referred to as “memory” or “a memory” may be composed of more than one different type of memory, and thus may refer to a collective component including one or more types of memory. It is readily understood that any single memory component may be separated into multiple collectively equivalent memory components, and vice versa. Furthermore, while memory may be depicted as separate from one or more other components (such as in the drawings), it is understood that memory may be integrated within another component, such as on a common integrated chip.

[0039] As used herein, the term “concurrency control” refers to the set of techniques and mechanisms designed to manage simultaneous interactions, processes, and data accesses for various computer applications to facilitate consistent and accurate operations. For example, concurrency control for intelligent voice services, which may include voice-enabled systems, may often involve multiple users, devices, and services interacting concurrently, necessitating robust solutions to handle competing requests, avoid conflicts, and maintain performance. Concurrency control for resource invocations in intelligent voice services may include mechanisms and strategies designed to manage simultaneous requests for shared resources in the form of e.g., devices, services, or data. In some embodiments, concurrency handling/control may include metadata for prioritization, locking, or managing simultaneous requests targeting the same resource.

[0040] As used herein, the term “resource invocation” broadly refers to initiating an operation or accessing a resource, such as a device, service, application, or dataset, to perform or execute a specific task or function. In the context of intelligent systems, this involves triggering actions, which may be through commands, such as voice inputs, that result in the utilization of hardware, software, or data resources. A resource invocation request broadly refers to any formalized or structured communication sent by a user, application, or system to initiate an action or operation on a specific resource. The resource invocation request may specify the target resource to be accessed, the action to be performed, and any required parameters or contextual information. A target specification may identify the resource to be

invoked, for example, a device, application programming interface-API, or service. An action to be performed may describe the operation to be performed on the resource, for example, read, write, execute, or modify. A parameterization may include any inputs, settings, or configurations required for the operation, for example, a volume level for a speaker, a brightness control for a lamp, etc. In some embodiments, the resource invocation request may comply with a specific communication protocol, such as Hypertext Transfer Protocol (HTTP) for web services, MQ Telemetry Transport (MQTT) for Internet-of-Things (IoT) devices, or custom messaging for proprietary systems.

[0041] As used herein, the term “concurrent count” refers to the number of simultaneous invocations or executions of a resource, such as a serverless function or service, at any given moment. This metric is utilized for understanding how well a system can handle multiple requests and for ensuring that resources are allocated efficiently.

[0042] As used herein, the term “configured to” broadly refers to an arrangement, a design, and/or a program to perform a specific function. It implies a purposeful arrangement or adaptation for achieving the stated functionality. For example, a processor configured to process data may include hardware (e.g. electronic circuitry, chips), and/or software working in tandem to process the data.

[0043] According to an aspect of the disclosure and with reference to FIG. 1, there is provided a server apparatus for processing resource invocation requests. The server apparatus may be part of a distributed system, the server apparatus arranged or operable to invoke limited concurrent resources, such as the control of third-party voice resource invocations in an intelligent voice service system. The server apparatus may be used to achieve concurrency control of target resources that may be accessed by multiple requestors.

[0044] The server apparatus may comprise a processor and a memory, the processor is capable of being configured to execute instructions stored in the memory to receive a request. In the embodiment illustrated in FIG. 1A and FIG. 1B, the server apparatus may be a communications server apparatus. The communications server apparatus may be in the form of a server computer 100, the server computer 100 may be a single server as illustrated schematically in FIG. 1A, or have the functionality performed distributed across multiple server components.

[0045] In some embodiments, the server computer 100 includes a communication interface 102. The communication interface 102 may be configured to send and receive data, which may include resource invocation requests. The communication interface 102 may include a

transmitter module and/or a receiver module allowing the server apparatus to communicate over a communications network. The communication interface 102 may include one or more user-interfaces configured to provide users for user control and may include, for example, one or more computing peripheral devices such as display monitors, computer keyboards and the like.

[0046] The server computer 100 may further include a processor in the form of processing unit 104 and a memory 106. The memory 106 may be used by the processing unit 104 to store, for example, the resource invocation requests, one or more status data of the target resource, one or more concurrent counts of the target resource to be processed.

[0047] As shown in FIG. 1B, the processing unit 104 may be configured to receive, from a terminal device 110, a resource invocation request 111 for processing. The resource invocation request 111 may be parsed to obtain information or data relating to one or more target resources.

[0048] The server computer 100 and one or more terminal devices 110 may be connected via a network 120 to form a system 150 for processing the data. The network 120 may be an Internet or Intranet network. In some embodiments, the system 150 may comprise one or more target resource providers 130.

[0049] The server computer 100 may obtain a first real-time concurrent count 112 of the target resource(s). This may be achieved using a monitoring algorithm installed on either the target resource providers 130 or the server computer 100. The processing unit 104 may then be configured to increase the first real-time concurrent count by a first predetermined number to obtain a first updated real-time concurrent count 113. The first updated real-time concurrent count 113 may represent a reservation of the target resources to be invoked. It is appreciable that the reservation of the target resource does not utilize or consume the target resource, but reserves a resource quota for utilization or consumption.

[0050] The processing unit 104 may then be operable/configured to compare the first updated real-time concurrent count 113 with a first threshold. In a positive determination that the first updated real-time concurrent count is greater than the first threshold, the processing unit 104 is then configured to obtain a second real-time concurrent count of the target resource; subtract a second predetermined number from the second real-time concurrent count to obtain a second updated concurrent count, and reject the resource invocation request. Such a configuration achieves a technical solution of providing a relatively more precise concurrency control through a pre-reservation mechanism. The pre-reservation mechanism avoids

concurrency conflicts by pre-allocating resource quotas before attempting to create an invocation (call) session. In the event the reserved quota is more than the first threshold, the resource invocation request is rejected and the reservation is released.

[0051] In some embodiments, in a positive determination that the first updated concurrent count is less than or equal to the first threshold, the processing unit 104 is configured to accept the resource invocation request and create an invocation session for a resource requestor to invoke the target resource.

[0052] In some embodiments, the processor is configured to assign the real-time concurrent count as the first updated real-time concurrent count.

[0053] In some embodiments, the processor is configured to reserve the use of the target resource after the increase of the first real-time concurrent count by the first predetermined number.

[0054] In some embodiments, in the event the invocation session fails, the processor is configured to obtain a third real-time concurrent count, and subtract a third predetermined number from the third real-time concurrent count to obtain a third updated concurrent count.

[0055] In some embodiments, the processor is configured to create a concurrency information table for storage of a first key-value pair, the first key-value pair comprises a unique identifier associated with the target resource as the key, and a real-time concurrent count of the target resource as the value.

[0056] In some embodiments, the processor is configured to determine a success status of the creation of the invocation session.

[0057] In some embodiments, in a positive determination that the invocation session is successful, the processor is configured to determine one or more states of the invocation session.

[0058] In some embodiments, if the usage status is indicative of a completion of the invocation session, the processor is configured to obtain a fourth real-time concurrent count of the target resource, and subtract or decrease a fourth predetermined value from the fourth real-time concurrent count to obtain a fourth updated concurrent count.

[0059] In some embodiments, the processor is configured to create a session identifier, the session identifier comprises a first state indicative of the completion of the invocation session and a second state indicative of a resource utilization state.

[0060] In some embodiments, the processor is configured to compare a usage duration of the invocation session with a predetermined duration, and wherein where the usage duration is

greater than the predetermined duration, the processor is configured to obtain a fifth real-time concurrent count of the target resource, and subtract or decrease a fifth predetermined value from the fifth real-time concurrent count to obtain a fifth updated concurrent count.

[0061] FIG. 2 is a flow chart depicting a method 200 of responding to a resource requestor's invocation request for a target resource. The resource requestor may be, or may make use of, a terminal device 110. The method may be implemented as executable software codes stored in one or more non-transitory computer-readable medium of the processing unit 104 and/or server computer 100. The method 200 may comprise the steps of:

[0062] Step S201: In response to a resource requestor's invocation request for a target resource, obtain the first real-time concurrent count of the target resource. The first real-time concurrent count may refer to a current concurrent count of invocations for the target resource. In some embodiments, the target resource can be an external resource of a distributed system, such as a voice resource provided by a third-party, or an internal resource of the distributed system, such as a resource stored in any computer node within the distributed system.

[0063] In some embodiments, the method 200 may include a step S201A, wherein if the first real-time concurrent count is not obtained or is an invalid number, the processor is configured to initialize the first real-time concurrent count to an initial value, such as 0 value.

[0064] The first real-time concurrent count may be obtained through a distributed cache, the distributed cache may be implemented using a Remote Dictionary Server (Redis). The first real-time concurrent count or other concurrent counts can be stored using key-value pairs, such as the key-value pairs in Redis. For example, a unique resource identifier can be used as a key to store its concurrent count. In one example, an atomic operation application programming interface (API) of Redis (such as the Redis INCR command) may be used to improve accuracy of counting in high-concurrency scenarios.

[0065] In some embodiments, a concurrent information management mechanism is introduced on the basis of concurrent count control, thereby ensuring the accuracy of the aforementioned concurrency control in distributed high-concurrency environments, particularly its reliability during anomalies.

[0066] In step S202, the first real-time concurrent count may be increased by a first predetermined or preset number to obtain a first updated real-time concurrent count.

[0067] The first concurrent count adjustment for attempting to reserve the resource can be performed via step S202, which involves increasing the first real-time concurrent count obtained in step S201 by a first predetermined value to obtain the first updated concurrent

count. In some embodiment, the first predetermined value can be set to 1, representing the reservation of one concurrent target resource quota. This reservation mechanism can lock the concurrent quota before the resource is actually used, thereby avoiding concurrency conflicts that may occur in traditional solutions. Furthermore, even if the target resource is ultimately not actually used due to various reasons, such as eventually deemed exceeding the concurrent limit or session creation failure, the method provided can still correctly restore the concurrent count through subsequent concurrent count adjustments.

[0068] In the example where the first predetermined value is 1, the current concurrent count can be increased from n to $n+1$, ensuring accuracy in high-concurrency scenarios. Where the concurrent counts are stored as key-value pairs under Redis, which may include using a concurrent information table, the first updated concurrent count before reservation can be stored in the value of a first key-value pair, replacing the previously stored value, such as the first real-time concurrent count obtained previously. For example, if the current first key-value pair is "service_001:n", after performing the first concurrent count adjustment, it becomes "service_001: n+1", where "service_001" is the unique identifier for the target resource, and "n" and "n+1" represent the concurrent counts before and after reservation, respectively.

[0069] In step S203, the first updated real-time concurrent count may be compared with a first threshold, which is also a concurrent count threshold of the target resource.

[0070] Step S204 includes determining whether the first updated concurrent count (e.g. $n+1$) obtained in step S202 is greater than the first threshold. In a specific example, assuming the concurrent count threshold for the target resource is k , it may be determined whether the first updated concurrent count ($n+1$) is greater than k .

[0071] In step S205, in a positive determination that the first updated concurrent count is greater than the first threshold, perform step S206 for a second concurrent count adjustment.

[0072] Step S206 involves, when the first updated concurrent count is greater than the first (concurrent count) threshold, performing a second concurrent count adjustment by subtracting a second predetermined (preset) value from the second real-time concurrent count to obtain the second updated concurrent count. The second concurrent count adjustment will effectively release the "reserved" resource quota. In one example, the second real-time concurrent count of the target resource can be obtained and be denoted as m . The value of the second real-time concurrent count m may be equal to $n+1$ or may not be equal to $n+1$. In one example, the actual real-time concurrent status may have changed when obtaining the second real-time concurrent count, for example, other resource invocation sessions may have ended and may release

resources not accounted for during the obtainment of the first concurrent count. By obtaining two real-time concurrent counts at different times, changes in the concurrent count in high-concurrency scenarios may be accurately determined. For example, after obtaining the first real-time concurrent count n , other requests may have released resources, causing the actual concurrent count to decrease, or other requests may have successfully reserved resources, causing the actual concurrent count to increase. After obtaining the second real-time concurrent count, it can be subtracted by a preset value to obtain the second updated concurrent count. In the case where a concurrent information table is used, the first key-value pair can be updated to “service_001:m-1”. In this way, the reserved concurrent quota can be promptly released, and the current invocation request can be rejected.

[0073] It may be appreciable that the second predetermined number may be the same as the first predetermined number in some embodiments. Alternatively, the second predetermined number may be different from the first predetermined number in other embodiments.

[0074] Through the mechanism/method of obtaining real-time concurrent counts twice for the determination of the updated concurrent count in steps S202 and S206 (if applicable), the maximum allowed concurrent count can be effectively managed, while guaranteeing high availability and performance during the preemptive calculation of concurrent resources. Such an arrangement may improve concurrent invocation control in resource-constrained scenarios. As an example, when receiving one or more invocation requests for the target resource from multiple resource requestors, for one of the invocation requests, after obtaining its first real-time concurrent count and deriving the first updated concurrent count, subsequent threshold determinations can be continued. Meanwhile, for the next invocation request, the corresponding first real-time concurrent count can be obtained. In some embodiments, the mechanism can reduce the serial waiting time for concurrency control among multiple invocation requests. In addition, as judgments are made based on the first updated concurrent count determined from the first real-time concurrent count, situations may arise where resources are tight when obtaining the first real-time concurrent count, but by the time of the comparison with the thresholds, resources may have been scaled up or released by other requests, making resources potentially available. However, even if it appears to reduce resource utilization efficiency in such cases, it may be appreciable that the solution provided can effectively control invocation requests to not avoid exceeding the maximum allowed concurrent count, ensuring high availability, and thus improving resource utilization efficiency.

[0075] In step S207, in a positive determination that the first updated concurrent count is less than or equal to the first (concurrent count) threshold, the resource invocation request is deemed to be accepted, and an invocation session for the resource requestor to invoke the target resource will be created.

[0076] The first and second thresholds may be the maximum allowed concurrent access count for the target resource. In one example, the maximum allowed concurrent access count may be based on limitations imposed by the resource provider or system performance constraints. In another example, the concurrent count threshold may include a threshold that considers a safety margin based on the maximum allowed concurrent access count, which can be referred to as a safety threshold. In some other examples, the concurrent count threshold may be a preset value. In other examples, the concurrent count threshold may be a dynamically changing value, particularly suitable for scenarios where resources can be dynamically scaled.

[0077] In step S207, when it is determined positively that the first updated concurrent count does not exceed the concurrent count threshold, it may be determined that the target resource has sufficient concurrent quota for the corresponding resource requestor to use. At this point, the invocation session creation process can be initiated to create an invocation session for the resource requestor, enabling them to actually use the target resource.

[0078] In the case of creating a call session for a target resource for a resource requestor (i.e., when executing step S207), the resource invocation control method may further include:

[0079] Step S208: determining a creation status of the invocation (call) session. It is possible to determine in step S209 whether the call session is successfully created by checking for network anomalies or system failures. The creation of a call session may fail due to reasons such as network anomalies or system failures.

[0080] Step S209: In the event that the creation status indicates is a failure status of the creation of the invocation session, i.e. the call session creation fails, obtaining a third real-time concurrent count for the target resource, subtracting a third predetermined value or third preset value from the third real-time concurrent count to obtain a third updated concurrent count, and rejecting the invocation request. Put in another way, the step S210 performs a third concurrency adjustment, i.e., releasing the pre-reserved concurrency quota when the session creation fails. The third real-time concurrent count for the target resource may be obtained, and the third predetermined or preset value may be subtracted from it to obtain the third updated concurrent count. Through this mechanism, even in the case of session creation failure, the pre-reserved

concurrency quota may be promptly released for use by other invocation requests, and thus enhance resource utilization efficiency.

[0081] Where there is a concurrency information table utilized, the first key-value can be updated based on the third updated concurrent count. For example, if the third real-time concurrent count is p , the first key-value can be updated to "service_001:p-1", where "service_001" is a unique identifier for the target resource, and "p-1" is the third updated concurrent count after subtracting the third predetermined value of 1. The configuration of obtaining the third real-time concurrent count for adjustment to obtain the third updated concurrent count updates the value of the first key-value (key-value) in the system, and facilitates the consistency of updating the concurrency information table based on the real-time status of the target resource. Combined with the aforementioned first and second concurrency adjustment mechanisms, the use of the third concurrent count, third updated concurrent count, and the third predetermined value facilitates the maintenance of accurate control over concurrency at various stages of processing the invocation request, thereby improving concurrent invocation control issues in resource-constrained scenarios.

[0082] FIG. 3 is a flow chart illustrating a resource invocation control method 300 involving a session information management table in the form of a concurrency information table. The method 300 may include:

[0083] Step S301: Creating a concurrency information table. The concurrency information table may be used to record resource information and concurrency information. As described, the concurrency information table may be used to store a first key-value pair, where the unique resource identifier serves as the key and the respective first, second or third concurrent count as the value. The concurrency information table can also be used to record session information, as further elaborated below.

[0084] In one example, the first key-value pair may be in the form of "service_001:n". Correspondingly, the first real-time concurrent count, such as n , can be obtained by reading the value of the first key-value pair. In another example, when the first real-time concurrent count is not obtained, for example, because the first key-value pair does not yet exist in Redis, the first key-value pair can be created. At this point, step S301, namely creating the concurrency information table, can be executed as needed.

[0085] In step S302, the method may check for a successful creation of an invocation session based on steps S207. Upon the successful creation of an invocation session, the session information of the created invocation session may be recorded in the concurrency information

table. For example, the session information may include a session identifier, a session start time in the form of a time stamp, as may be further elaborated. In some embodiments, the session identifier may be used to uniquely identify the current invocation session, and the session start time may be used for subsequent determination on timeouts (timeout judgments). The session identifier may also be used for state identification control, i.e., by recording whether it is in an in-use state or a not-in-use state, to prevent duplicate release of concurrency quotas.

[0086] In some embodiments, the concurrency information table may be used to store a first key-value pair, where the unique resource identifier serves as the key and the concurrent count as the value. The concurrency information table can also use a key-value structure to record session information. In one embodiment, the concurrency information table can use a second key-value pair to record the session identifier. In further optional embodiments, the concurrency information table can also use a third key-value pair to record the session start time. In other words, the second key-value pair may comprise a unique identifier associated with the target resource as the key, and a session identifier as the value. The third key-value pair may comprise the unique identifier associated with the target resource as the key, and a time stamp associated with the start time of the session as the value.

[0087] The second key-value pair can be used to store the session identifier, i.e., the session state. For example, a combination of the resource identifier and the unique (session) identifier can be used as the key, such as in the form of "resource_identifier : session_identifier", and the session identifier can be used as the value. For instance, when the unique resource identifier is "service_001" and the unique session identifier is "session_123", a key-value pair in the form of "service_001:session_123:1" or "service_001:session_123:0" can be formed, where "service_001:session_123" serves as the key, and the value indicates the state indicated by the session identifier, such as setting the value to "1" to represent a first preset state (i.e., "in-use" state) or to "0" to represent a second preset state (i.e. "not in use" state). In one embodiment, the second key-value pair may be stored using a hash structure. A distributed cache (such as Redis) with a hash data structure may be used to implement the aforementioned storage solution. Correspondingly, resource information, concurrency information, and session information can be correlated, and managed uniformly in the same concurrency information table.

[0088] In step S303: In response to a change in the state of an invocation session, update the session information of the invocation session recorded in the concurrency information table. In some embodiments, the transition of the session identifier state from binary "1" to "0" can

be achieved by updating the second key-value pair. For example, updating the value corresponding to “service_001:session_123” from binary “1” to “0” indicates that the session has transitioned from the “in-use” state to the “not-in-use” state. It may be appreciable that step S303 is not only applicable to the scenario of session identifier state transitions described in this step but can also be applied to other scenarios of session state changes, such as, but not limited to, one or more intermediate states between the in-use state and the not-in-use state. Furthermore, the concurrency table is not limited to recording session state changes, it can optionally record changes in other session information as well. This unified information update mechanism improves the timeliness and accuracy of session information in the concurrency information table.

[0089] In step S304: The session information of invocation sessions recorded in the concurrency information table may be regularly or periodically detected, monitored, and if necessary, updated. The period may be predetermined by a user or a server administrator. In some embodiments, the detection and monitoring may be performed on-demand.

[0090] In some embodiments, the detection of session duration can be achieved by checking the session start time recorded in the aforementioned third key-value pair. The third key-value pair can use a combination of resource identifiers and session identifiers as the key, for example, in the form of “resource identifier : session identifier”, and the session start timestamp can be used as the value. For instance, when the unique resource identifier is “service_001” and the unique session identifier is “session_123”, a key-value pair such as “service_001:session_123:1698765432” can be formed, where 1698765432 represents the timestamp of the session start time.

[0091] It may be appreciable that step S304 is not only applicable to the session duration detection scenario described in this step but can also be used to detect other session information. The unified detection mechanism improves the validity of session information in the concurrency information table. In a specific embodiment, the third key-value pair may be stored using a Hash structure. A distributed cache (such as Redis) with a hash data structure can also be used to implement the aforementioned storage solution. Correspondingly, resource information, concurrency information, and session information can be uniformly managed within the same concurrency information table.

[0092] A session state management mechanism can also be employed to control concurrency adjustments when the call session is successfully created for the resource requestor. Such session state management mechanism can effectively prevent the issue of

repeated release of concurrency quotas in a distributed environment. For example, when a session simultaneously triggers timeout cleanup and abnormal termination due to network anomalies, without a state management mechanism, it may lead to repeated release of the concurrency quota.

[0093] FIG. 4 is a flow chart illustrating a session state management method 400 to control concurrency adjustments based on determining the usage status of an invocation (call) session in the case of successful invocation (call) session creation.

[0094] The session state management method 400 can include the steps of:

[0095] Step S401: Determining the usage status of the call session in the case of successful call session creation. The call session may include a complete resource invocation process between the resource requestor and the target resource, which may commence or start from the successful creation of the call session and end when the resource usage is completed. In some embodiments, the resource requestor can only begin to use the target resource after the system successfully creates a call session for it, and the call session will end when the resource usage is completed, either based on normal completion or abnormal termination. As a non-limiting example, in the context of an intelligent voice scenario, after an application successfully obtains permission to invoke a third-party voice resource and creates a session, the call session involves starting to process the voice task until it is completed or an anomaly occurs.

[0096] Step S402: Determining whether the call session is in use. The usage status can be determined by detecting whether the call session is actively engaged in resource invocation.

[0097] Step S403: Obtaining a fourth real-time concurrent count for the target resource in the case of call session completion, and subtracting a fourth predetermined value from the fourth real-time concurrent count to obtain a fourth updated concurrent count.

[0098] It is appreciable that there are no restrictions on the circumstances of session completion, which may include, but are not limited to, normal completion (e.g., resource invocation completed, resource user actively ends the session), and abnormal completion (e.g., session execution timeout, network connection interruption, etc.).

[0099] In step S403, a fourth concurrency adjustment may be performed, i.e., releasing the concurrency quota when the session is completed. Specifically, the fourth real-time concurrent count for the target resource may be obtained, and the fourth predetermined value is subtracted from it to obtain the fourth updated concurrent count. This mechanism improves that the system can promptly release the occupied concurrency quota when the session ends, allowing other invocation requests to use it, thereby improving resource utilization efficiency.

[00100] When using a concurrency information table, the first key-value (key-value) can be updated based on the fourth updated concurrent count. In a specific example, if the fourth real-time concurrent count is q , the first key-value can be updated to "service_001 : $q-1$ ", where "service_001" is a unique identifier for the target resource, and " $q-1$ " is the fourth updated concurrent count after subtracting the preset value.

[00101] The mechanism of obtaining the fourth real-time concurrent count for adjustment ensures that the value of the first key-value (key-value) in the system, such as the concurrency information table, can be updated based on the real-time status of the resource. Combined with the aforementioned first, second, and optional third concurrency adjustment mechanisms, this mechanism enables the system to control concurrency at various stages such as reservation, usage, and release, thereby further mitigating the issue of concurrent invocation control in resource-constrained scenarios.

[00102] FIG. 5 is a flow chart illustrating a session state management method 500 based on state identifiers employed to control the adjustment of concurrent counts. The session state management method 500 may be used to avoid a situation where concurrency quotas are released repeatedly during the process of ending an invocation session, e.g. ending a session call, and the method could further include the steps of:

[00103] Step S501: In the case where the invocation session is successfully created, creating a session identifier for the invocation session and set the session identifier to a first preset state. In this step S501, a session identifier may be created for the invocation session and initially set to a first preset state. For example, the first preset state can be set to indicate an "in-use" or being used status. In a specific example, when the invocation session is successfully created, such as after obtaining permission to invoke third-party voice resources in a smart voice scenario, a unique session identifier can be created for the invocation session, and the invocation session status can be marked as "in-use".

[00104] In step S502, at the end of using the invocation session, the method 500 may perform a step of identifying the state of the session identifier. In the step S502, it is possible to identify whether the session identifier is currently in the first preset state, such as the "in-use" state. For example, when the invocation session ends or is completed due to reasons such as normal completion or abnormal interruption, the status of the unique session identifier can be first checked to determine whether the concurrency resource quota for the session has been released. Such checking of the status of the session identifier can prevent repeated adjustments to the concurrent count. Where a concurrency information table is used, the system can identify the

current state of the session by querying a second key-value pair (where the key is a combination of the resource identifier and the session identifier, and the value is a numerical representation of the status).

[00105] In step S503, if the session identifier is identified as being in the first preset state, the method performs setting/changing the session identifier to a second preset state. In this step, if the session identifier is identified as being in the first preset state, the session identifier can be updated from the first preset state to the second preset state. For example, the second preset state can represent a "not-in-use" status. Where a concurrency information table is used, the transition of the session identifier's state can be achieved by updating the second key-value pair. In some embodiments, step S503 can be combined with step S303 or serve as a sub-step of it.

[00106] In step S504, only when the session identifier is identified as being in the first preset state, obtain the fourth real-time concurrent count of the target resource, and subtract a preset value from the fourth real-time concurrent count to obtain the fourth updated concurrent count.

[00107] In some embodiments of step S504, the fourth concurrent count adjustment is executed only after it is confirmed that the session identifier is in the first preset state. In one embodiment, the fourth real-time concurrent count of the target resource can be obtained, and the preset value is subtracted from it to obtain the fourth updated concurrent count. Where a concurrency information table is used, if the fourth real-time concurrent count is denoted as q , then the first key-value may be updated to "service_001:q-1", indicating the release of one concurrency resource quota. At the same time, relevant information of the invocation session in the concurrency information table can be cleared, including updating or clearing the session status information in the second key-value as needed, and clearing the session start time information recorded in the third key-value as needed. It may be appreciated that by associating the concurrent count adjustment with the session state, each session will release the concurrency quota at most once.

[00108] By introducing the session identifier and its state transition method, and associating the fourth concurrent count adjustment with the session identifier state, the situation of repeatedly releasing concurrency quotas can be avoided. The method 500, combined with the aforementioned first, second, and optional third concurrent count adjustment mechanisms, can better facilitate the accuracy of concurrent count tracking, thereby effectively addressing the issue of concurrency invocation control in resource-constrained scenarios. Particularly in distributed systems, where the same session may simultaneously trigger end events due to

various reasons such as timeouts or anomalies, the method 500 ensures that the concurrency quota is not mistakenly released multiple times. It is contemplated that although the method 500 illustrates the use of state transitions associated with session usage state for concurrency control, the embodiment can also use a concurrency information table to assist in managing session information. As a non-limiting example, the session usage state and start time information can be recorded in the concurrency information table to support the judgment or determination of session usage state and the detection of session duration.

[00109] FIG. 6 is a flow chart illustrating a periodic detection method 600 employed to ensure reliability of concurrency control, which includes periodically detecting a usage duration of created invocation sessions, in order to optimally handle potential timeout anomalies in session calls.

[00110] The periodic detection method 600 may be deployed to ensure the reliability of concurrency control, to optimally handle potential timeout anomalies in session calls, and may include the steps of:

[00111] Step S601: Periodically detecting the usage duration of created session calls.

[00112] In step S601, a periodic detection can be performed on the created session calls. For example, the duration from the start of each session call to the current time can be detected. The preset duration can be set to a value greater than the normal business processing duration, for example, the preset duration can be set to 7 minutes.

[00113] In embodiments where a concurrency information table is used, step S601 can be combined with step S304 or serve as a sub-step of step S304.

[00114] Step S602: In the case where the usage duration of a created session call exceeds the preset duration, obtain the fifth real-time concurrent count of the target resource, and subtract a preset value from the fifth real-time concurrent count to obtain the fifth updated concurrent count.

[00115] In step S602, a fifth concurrency adjustment can be performed, that is, releasing concurrency quotas upon session timeout. Specifically, the fifth real-time concurrent count of the target resource can be obtained, and the preset value can be subtracted from it to obtain the fifth updated concurrent count. In embodiments where a concurrency information table is used, if the fifth real-time concurrent count is r , the first key-value can be updated to "service_001:r-1", indicating the release of one concurrency quota. At the same time, relevant information of the session in the concurrency information table can be cleared, including the session state

information in the second key-value and the session start time information in the third key-value.

[00116] It is contemplated that by periodically detecting and promptly releasing concurrency quotas occupied by timed-out sessions, long-term occupation of concurrency quotas under abnormal circumstances can be avoided. This method 600, combined with the aforementioned session state management mechanism, can maintain effective control over concurrent counts in both normal and abnormal scenarios. Particularly in distributed systems, even if an anomaly occurs on a certain node, preventing it from normally releasing concurrency quotas, this method 600 can automatically correct the issue through timeout detection, thereby ensuring the reliability of concurrency control. Furthermore, by integrating concurrent count control and session information management into the same concurrency information table, unified management of target resource invocations can be achieved, supporting the full lifecycle management from session creation, state transitions to timeout handling.

[00117] FIG. 7 shows a generalized flow chart of a method for processing resource invocation requests according to the present disclosure. The method comprises the steps of:

[00118] Step S701: obtaining a resource invocation request, the resource invocation request associated with a target resource;

[00119] Step S702: obtaining a first real-time concurrent count of the target resource;

[00120] Step S703: increasing the first real-time concurrent count by a first predetermined number to obtain a first updated real-time concurrent count;

[00121] Step S704: comparing the first updated real-time concurrent count with a first threshold;

[00122] wherein in a positive determination that the first updated real-time concurrent count is greater than the first threshold,

[00123] Step S705: obtaining a second real-time concurrent count of the target resource;

[00124] Step S706: subtracting a second predetermined number from the second real-time concurrent count to obtain a second updated concurrent count, and

[00125] Step S707: rejecting the resource invocation request.

[00126] In some embodiments, in a positive determination that the first updated concurrent count is less than or equal to the first threshold, the method may comprise a step of accepting the resource invocation request and creating an invocation session for a resource requestor to invoke the target resource.

[00127] In some embodiments, the method comprises a step of assigning the real-time concurrent count as the first updated real-time concurrent count.

[00128] In some embodiments, if the first real-time concurrent count is not obtained or is an invalid number, the method comprises a step of initializing the first real-time concurrent count to an initial value.

[00129] In some embodiments, the method comprises reserving the use of the target resource after the increase of the first real-time concurrent count by the first predetermined number.

[00130] In some embodiments, the second predetermined number is the same as the first predetermined number. Alternatively, the second predetermined number is different from the first predetermined number.

[00131] In some embodiments, the method comprises determining a failure status of the creation of the invocation session.

[00132] In some embodiments, in the event the invocation session fails, the method comprises obtaining a third real-time concurrent count, and subtract a third predetermined number from the third real-time concurrent count to obtain a third updated concurrent count.

[00133] In some embodiments, the method comprises to create a concurrency information table for storage of a first key-value pair, the first key-value pair comprises a unique identifier associated with the target resource as the key, and a real-time concurrent count of the target resource as the value.

[00134] In some embodiments, the method comprises storing at least a second key-value pair and a third key-value pair; wherein the second key-value pair comprises the unique identifier associated with the target resource as the key, and a session identifier as the value; and wherein the third key-value pair comprises the unique identifier associated with the target resource as the key, and a time stamp associated with a start time of the session as the value.

[00135] In some embodiments, the method comprises determining a success status of the creation of the invocation session, and wherein in a positive determination that the invocation session is successful, the method comprises determining one or more states of the invocation session. If the one or more states is indicative of a completion of the invocation session, the processor is configured to obtain a fourth real-time concurrent count of the target resource, and subtract or decrease a fourth predetermined value from the fourth real-time concurrent count to obtain a fourth updated concurrent count.

[00136] In some embodiments, the method comprises creating a session identifier, the session identifier comprises a first state indicative of the completion of the invocation session and a second state indicative of a resource utilization state.

[00137] In some embodiments, the method comprises comparing a usage duration of the invocation session with a predetermined duration, and wherein where the usage duration is greater than the predetermined duration, the processor is configured to obtain a fifth real-time concurrent count of the target resource, and subtract or decrease a fifth predetermined value from the fifth real-time concurrent count to obtain a fifth updated concurrent count.

[00138] The steps of the disclosed methods are not restricted to the sequence set forth in the claims unless explicitly required. Steps may be performed in a different order, concurrently, or omitted entirely, depending on the embodiment.

[00139] This present disclosure provides a technical solution for improving concurrency control through a pre-reservation mechanism and state management. The pre-reservation mechanism avoids concurrency conflicts by pre-allocating or reserving resource quotas before attempting to create a call session. The state management ensures accurate concurrency control by monitoring the creation status, usage status, etc., of sessions, and facilitates high availability and high performance for target resource invocations through these mechanisms.

[00140] It may be appreciable that the server apparatus may be used in various applications, potentially in any applications involving concurrent resources and the resource invocation requests associated with such concurrent resources, etc.

[00141] The methods described herein may be performed and the various processing or computation units and the devices and computing entities described herein may be implemented by one or more circuits. In an embodiment, a "circuit" may be understood as any kind of a logic implementing entity, which may be hardware, software, firmware, or any combination thereof. Thus, in an embodiment, a "circuit" may be a hard-wired logic circuit or a programmable logic circuit such as a programmable processor, e.g. a microprocessor. A "circuit" may also be software being implemented or executed by a processor, e.g. any kind of computer program, e.g. a computer program using a virtual machine code. Any other kind of implementation of the respective functions which are described herein may also be understood as a "circuit" in accordance with an alternative embodiment.

[00142] While the disclosure has been particularly shown and described with reference to specific embodiments, it should be understood by those skilled in the art that various changes in form and detail may be made therein without departing from the spirit and scope of the

disclosure as defined by the appended claims. The scope of the disclosure is thus indicated by the appended claims and all changes which come within the meaning and range of equivalency of the claims are therefore intended to be embraced.

CLAIMS

1. A server apparatus, the server apparatus comprises a processor, the processor configured to:
 - obtain a resource invocation request, the resource invocation request associated with a target resource;
 - obtain a first real-time concurrent count of the target resource;
 - increase the first real-time concurrent count by a first predetermined number to obtain a first updated real-time concurrent count;
 - compare the first updated real-time concurrent count with a first threshold;
 - wherein in a positive determination that the first updated real-time concurrent count is greater than the first threshold,
 - obtain a second real-time concurrent count of the target resource;
 - subtract a second predetermined number from the second real-time concurrent count to obtain a second updated concurrent count, and
 - reject the resource invocation request.
2. The server apparatus of claim 1, wherein in a positive determination that the first updated concurrent count is less than or equal to the first threshold, the processor is configured to accept the resource invocation request and create an invocation session for a resource requestor to invoke the target resource.
3. The server apparatus of claim 2, wherein the processor is configured to assign the real-time concurrent count as the first updated real-time concurrent count.
4. The server apparatus of claim 1 or 2, wherein if the first real-time concurrent count is not obtained or is an invalid number, the processor is configured to initialize the first real-time concurrent count to an initial value.
5. The server apparatus of any one of the preceding claims, wherein the processor is configured to reserve a usage of the target resource after the increase of the first real-time concurrent count by the first predetermined number.

6. The server apparatus of any one of the preceding claims, wherein the second predetermined number is the same as the first predetermined number.
7. The server apparatus of any one of claims 1 to 5, wherein the second predetermined number is different from the first predetermined number.
8. The server apparatus of claim 2, wherein the processor is configured to determine a failure status of the creation of the invocation session.
9. The server apparatus of claim 8, wherein in a positive determination that the invocation session fails, the processor is configured to obtain a third real-time concurrent count, and subtract a third predetermined number from the third real-time concurrent count to obtain a third updated concurrent count.
10. The server apparatus of any one of the preceding claims, wherein the processor is configured to create a concurrency information table for storage of a first key-value pair, the first key-value pair comprises a unique identifier associated with the target resource as the key, and a real-time concurrent count of the target resource as the value.
11. The server apparatus of claim 10, wherein the processor is configured to store at least a second key-value pair and a third key-value pair; wherein the second key-value pair comprises the unique identifier associated with the target resource as the key, and a session identifier as the value; and wherein the third key-value pair comprises the unique identifier associated with the target resource as the key, and a time stamp associated with a start time of the session as the value.
12. The server apparatus of any one of claims 2 to 7, wherein the processor is configured to determine a success status of the creation of the invocation session.
13. The server apparatus of claim 12, wherein in a positive determination that the invocation session is successful, the processor is configured to determine one or more states of the invocation session.

14. The server apparatus of claim 13, wherein if the one or more states is indicative of a completion of the invocation session, the processor is configured to obtain a fourth real-time concurrent count of the target resource, and subtract or decrease a fourth predetermined value from the fourth real-time concurrent count to obtain a fourth updated concurrent count.
15. The server apparatus of claim 14, wherein the processor is configured to create a session identifier, the session identifier comprises a first state indicative of the completion of the invocation session and a second state indicative of a resource utilization state.
16. The server apparatus of any one of claims 13 to 15, wherein the processor is configured to compare a usage duration of the invocation session with a predetermined duration, and wherein where the usage duration is greater than the predetermined duration, the processor is configured to obtain a fifth real-time concurrent count of the target resource, and subtract or decrease a fifth predetermined value from the fifth real-time concurrent count to obtain a fifth updated concurrent count.
17. A system for processing resource invocation requests, the system comprises
a target resource provider;
the server apparatus of claim 2, the server apparatus arranged in data communication with the target resource provider to determine whether to reject or accept the resource invocation request; and create the invocation session for to invoke the target resource in a positive determination to accept the resource invocation request.
18. The system of claim 17, wherein the target resource is an external resource of a distributed system, or an internal resource of the distributed system.
19. A method for processing resource invocation requests, the method comprises:
obtaining a resource invocation request, the resource invocation request associated with a target resource;
obtaining a first real-time concurrent count of the target resource;
increasing the first real-time concurrent count by a first predetermined number to obtain a first updated real-time concurrent count;
comparing the first updated real-time concurrent count with a first threshold;

wherein in a positive determination that the first updated real-time concurrent count is greater than the first threshold,

obtaining a second real-time concurrent count of the target resource;

subtracting a second predetermined number from the second real-time concurrent count to obtain a second updated concurrent count, and

rejecting the resource invocation request.

20. A non-transitory computer-readable medium storing computer executable code comprising instructions for processing data according to the method of claim 19.