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

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
23 February 2006 (23.02.2006)

PCT

(10) International Publication Number
WO 2006/020338 A1

(51) International Patent Classification:
G06F 9/46 (2006.01)

(21) International Application Number:
PCT/US2005/025887

(22) International Filing Date: 20 July 2005 (20.07.2005)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:

10/917,715	12 August 2004 (12.08.2004)	US
60/652,368	11 February 2005 (11.02.2005)	US
11/168,967	27 June 2005 (27.06.2005)	US

(71) Applicant *(for all designated States except US)*: **ORACLE INTERNATIONAL CORPORATION** [US/US]; 500 Oracle Parkway, Redwood Shores, CA 94065 (US).

(72) Inventors; and

(75) Inventors/Applicants *(for US only)*: **COLRAIN, Carol** [AU/US]; 20 Dockside Circle, Redwood Shores, CA 94065 (US). **SIMMONS, Charles** [US/US]; 3535 Altamont Way, Redwood City, CA 94062 (US). **SEMLER, Daniel** [AU/US]; 1800 Notre Dame Avenue, Belmont, CA 94402 (US). **POMMERENK, Stefan** [DE/US]; 27156 Columbia Way, Hayward, CA 94542 (US). **PRUSCINO, Angelo** [IT/US]; 436 Distel Drive, Los Altos, CA 94022 (US).

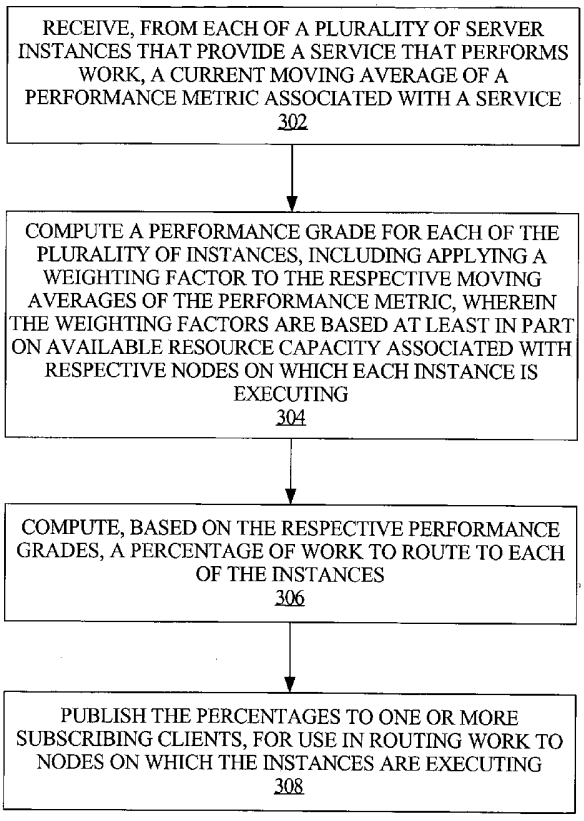
(74) Agents: **HENKHAUS, John** et al.; Hickman, Palermo, Truong & Becker, LLP, 2055 Gateway Place, Suite 550, San Jose, CA 95110-1089 (US).

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

(84) Designated States *(unless otherwise indicated, for every kind of regional protection available)*: ARIPO (BW, GH,

[Continued on next page]

(54) Title: RUNTIME LOAD BALANCING OF WORK ACROSS A CLUSTERED COMPUTING SYSTEM USING CURRENT SERVICE PERFORMANCE LEVELS



(57) Abstract: Runtime load balancing of work across a clustered computing system involves servers calculating, and clients utilizing, current service performance grades of each instance in the system. A performance grade for an instance is based on performance metrics for that instance, where the computation used may vary by policy. Examples of possible policies include: (a) using estimated bandwidth as a performance grade, (b) using spare capacity as a performance grade, or (c) using response time as a performance grade. Clients distribute work requests across servers in the system as the requests arrive. Work requests can be distributed according to performance grades, and/or flags associated with the performance grades. Automatically and intelligently directing work requests to the best server instances, based on real-time service performance metrics, minimizes the need to manually relocate work within the clustered system.

WO 2006/020338 A1

GM, KE, LS, MW, MZ, NA, SD, SL, SZ, TZ, UG, ZM, ZW), Eurasian (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European (AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HU, IE, IS, IT, LT, LU, LV, MC, NL, PL, PT, RO, SE, SI, SK, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

Published:

— *with international search report*

— *before the expiration of the time limit for amending the claims and to be republished in the event of receipt of amendments*

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

RUNTIME LOAD BALANCING OF WORK ACROSS A CLUSTERED COMPUTING SYSTEM USING CURRENT SERVICE PERFORMANCE LEVELS

FIELD OF THE INVENTION

[0001] The present invention relates generally to distributed computing systems and, more specifically, to techniques for runtime load balancing of work across a distributed computing system using current service performance grades.

BACKGROUND OF THE INVENTION

[0002] Many enterprise data processing systems rely on distributed database servers to store and manage data. Such enterprise data processing systems typically follow a multi-tier model that has a distributed database server in the first tier, one or more computers in the middle tier linked to the database server via a network, and one or more clients in the outer tier.

CLUSTERED COMPUTING SYSTEM

[0003] A clustered computing system is a collection of interconnected computing elements that provide processing to a set of client applications. Each of the computing elements is referred to as a node. A node may be a computer interconnected to other computers, or a server blade interconnected to other server blades in a grid. A group of nodes in a clustered computing system that have shared access to storage (e.g., have shared disk access to a set of disk drives or non-volatile storage) and that are connected via interconnects is referred to herein as a work cluster.

[0010] A clustered computing system is used to host clustered servers. A server is combination of integrated software components and an allocation of computational resources, such as memory, a node, and processes on the node for executing the integrated software components on a processor, where the combination of the software and computational resources are dedicated to providing a particular type of function on behalf of clients of the server. An example of a server is a database server. Among other functions of database management, a database server governs and facilitates access to a particular database, processing requests by clients to access the database.

[0011] Resources from multiple nodes in a clustered computing system can be allocated to running a server's software. Each allocation of the resources of a particular node for the server is referred to herein as a "server instance" or instance. A database server can be clustered, where the server instances may be collectively referred to as a cluster. Each

instance of a database server facilitates access to the same database, in which the integrity of the data is managed by a global lock manager.

SERVICES FOR MANAGING APPLICATIONS ACCORDING TO SERVICE LEVELS

[0012] Services are a feature for database workload management that divide the universe of work executing in the database, to manage work according to service levels. Resources are allocated to a service according to service levels and priority. Services are measured and managed to efficiently deliver the resource capacity on demand. High availability service levels use the reliability of redundant parts of the cluster.

[0013] Services are a logical abstraction for managing workloads. Services can be used to divide work executing in a database cluster into mutually disjoint classes. Each service can represent a logical business function, e.g., a workload, with common attributes, service level thresholds, and priorities. The grouping of services is based on attributes of the work that might include the application function to be invoked, the priority of execution for the application function, the job class to be managed, or the data range used in the application function of a job class. For example, an electronic-business suite may define a service for each responsibility, such as general ledger, accounts receivable, order entry, and so on. Services provide a single system image to manage competing applications, and the services allow each workload to be managed in isolation and as a unit. A service can span multiple server instances in a cluster or multiple clusters in a grid, and a single server instance can support multiple services.

[0014] Middle tier and client/server applications can use a service, for example, by specifying the service as part of the connection. For example, application server data sources can be set to route to a service. In addition, server-side work sets the service name as part of the workload definition. For example, the service that a job class uses is defined when the job class is created, and during execution, jobs are assigned to job classes and job classes run within services.

DATABASE SESSIONS

[0015] In order for a client to interact with a database server on a database cluster, a session is established for the client. Each session belongs to one service. A session, such as a database session, is a particular connection established from a client to a server, such as a database instance, through which the client issues a series of requests (e.g., requests for execution of database statements). For each database session established on a database instance, session state data is maintained that reflects the current state of a database session. Such information contains, for example, the identity of the client for which the session is

established, the service used by the client, and temporary variable values generated by processes executing software within the database session. Each session may each have its own database process or may share database processes, with the latter referred to as multiplexing.

BRIEF DESCRIPTION OF THE DRAWINGS

[0016] Embodiments of the present invention are depicted by way of example, and not by way of limitation, in the figures of the accompanying drawings and in which like reference numerals refer to similar elements and in which:

[0017] FIG. 1 is a block diagram that illustrates an operating environment in which an embodiment can be implemented;

[0018] FIG. 2 is a block diagram that illustrates a server operating environment, in which an embodiment of the invention may be implemented;

[0019] FIG. 3 is a flow diagram that illustrates a method for determining how much work to route to nodes in a distributed computing system, according to an embodiment; and

[0020] FIG. 4 is a block diagram that depicts a computer system upon which an embodiment of the invention may be implemented.

DETAILED DESCRIPTION OF EMBODIMENTS OF THE INVENTION

[0021] Techniques for runtime load balancing of work across a distributed computing system are described. Such techniques compute and make available to client subscribers, current service level performance information and work distribution advisories, with which work routing decisions can be made. Service level performance information and work distribution advisories regarding the different instances of the system can be used to allow balancing of the work across the system.

FUNCTIONAL OVERVIEW OF EMBODIMENTS

[0022] Runtime load balancing of work across a clustered computing system involves servers calculating, and clients utilizing, the current service performance levels of each instance in the system. Such performance levels ("performance grades") are based on performance metrics, and corresponding percentage distribution advisories are posted for use by various types of client subscribers.

[0023] Within a multi-instance server, various performance metrics are gathered for each instance. These performance metrics may include operations completed per second, elapsed time per operation, CPU utilization, I/O utilization, network utilization, and the like. A moving average of the metrics is usually used in order to smooth out any short-term

variations. In one embodiment, each instance within the server periodically sends its performance metrics to a centralized location.

[0024] The server then computes a performance grade for each instance. The computation used may vary by policy. Examples of possible policies include: (a) using estimated bandwidth as a performance grade, (b) using spare capacity as a performance grade, or (c) using response time as a performance grade. The server may compute a performance grade for each instance without regard to the performance of other instances, or the server may holistically look at all instances to produce a grade for each instance. The server publishes the performance grades of the instances to the client subscribers.

[0025] The performance grades may be used by clients to effectively route work to instances without regard to the number of client subscribers. In one embodiment, performance grades are posted via events, which can be subscribed to by many client subscribers. Non-limiting examples of subscribers include connection pools, load balancers, job schedulers, etc. Any client that wants to route service-based work within the system can use the runtime load balancing performance grades.

[0026] Using the described techniques, clients distribute work requests across servers in a clustered computing environment as the requests arrive. Work requests can be distributed according to performance grades, or the client may use other information available to the client to route the work. For example, the client may wish to route work to where the requestor last interacted (referred to as a "sticky" policy). Automatically and intelligently directing work requests to the best server instances, based on real-time service performance metrics, minimizes the need to manually relocate work within the clustered system.

[0027] In general, basing work request routing decisions on performance grades recognizes, for non-limiting examples, differences in various machine's current workload and computing power, sessions that are blocked in wait mode, failures that block processing, and competing services having different levels of priority. In other words, work requests are routed based on "bandwidth," where the work distribution percentage of a node is proportional of that node's bandwidth to the total bandwidth of all nodes that can support the service whose load is being balanced. The bandwidth of a cluster, for example, is the sum of the bandwidths of the nodes in the cluster. There are a number of ways to estimate the bandwidth of a node. In one embodiment, node bandwidth is estimated by measuring the throughput of a service on a node in units of work completed per second, and to scale the throughput up by the unused capacity of the node.

OPERATING ENVIRONMENT

[0028] FIG. 1 is a block diagram that illustrates an operating environment in which an embodiment can be implemented. Use of element identifiers ranging from *a* to *n*, for example, clients 102a-102n, services 106a-106n, instances 108a-108n, and nodes 110a-110n, does not mean that the same number of such components are required. In other words, *n* is not necessarily equal for the respective components. Rather, such identifiers are used in a general sense in reference to multiple similar components.

CLUSTERED COMPUTING ENVIRONMENT

[0029] One or more clients 102a-102n are communicatively coupled to a server cluster 104 ("server") that is connected to a shared database 112. Server 104 refers collectively to a cluster of server instances 108a-108n and nodes 110a-110n on which the instances execute. Other components may also be considered as part of the server 104, such as automatic workload repository 118. However, the actual architecture in which the foregoing components are configured may vary from implementation to implementation. Clients 102a-102n may be applications executed by computers interconnected to an application server or some other middleware component between clients and server 104 via, for example, a network. In addition, one server instance may be a client of another server instance. Any or all of clients 102a-102n may operate as subscribers of published events, as described herein.

[0030] In the context of a database cluster, database 112 comprises data and metadata that is stored on a persistent memory mechanism, such as a set of hard disks that are communicatively coupled to nodes 110a-110n, each of which is able to host one or more instances 108a-108n, each of which hosts at least a portion of one or more services. Such data and metadata may be stored in database 112 logically, for example, according to relational database constructs, multidimensional database constructs, or a combination of relational and multidimensional database constructs. Nodes 110a-110n can be implemented as a conventional computer system, such as computer system 400 illustrated in FIG. 4.

[0031] As described, a database server is a combination of integrated software components and an allocation of computational resources (such as memory and processes) for executing the integrated software components on a processor, where the combination of the software and computational resources are used to manage a particular database, such as database 112. Among other functions of database management, a database server typically facilitates access to database 112 by processing requests from clients to access the database 112. Instances 108a-108n, in conjunction with respective nodes 110a-110n, host services 106a-106n.

OPERATING ENVIRONMENT-SERVER

[0032] FIG. 2 is a block diagram that illustrates a server operating environment, in which an embodiment of the invention may be implemented. Server 202 operates similarly to server cluster 104 of FIG. 1, instances 202a-202n operate similarly to instances 108a-108n of FIG. 1. Therefore, unless otherwise stated, the descriptions and interrelationships of such components in reference to FIG. 1 apply to the similarly operating components depicted in FIG. 2.

[0033] FIG. 2 is referenced in describing, generally, the manner in which service performance is derived among a plurality of instances in a clustered computing environment. Each instance 202a, 202b, 202c, 202n comprises a respective manageability monitor (MMON) 203a, 203b, 203c, 203n. Each MMON 203a-203n is, in one embodiment, a dedicated manageability background process, which handles automatic management within the server 202. Using MMON, components in the server 202 can schedule periodic performance of system and service monitoring-type actions.

[0034] Each MMON 203a-203n periodically computes metric values and checks for occurrence of system events. MMONs 203a-203n are capable of comparing metric thresholds with actual values and generating alerts if necessary. MMONs 203a-203n can post service performance grades and advisories, for subscribing clients, via a notification system as described herein in reference to FIG. 1.

[0035] One MMON from the server 202 serves as a master, for metric collection and performance grade derivation purposes. For example, the instance with the lowest instance number may be elected as the master. In this illustration, MMON 203a of instance 202a is considered the master. The master periodically requests, from the other instance MMONs, the locally generated performance statistics. From that information, the master calculates a performance grade for each instance in the system, as described in more detail hereafter. The master then posts the derived performance grades to the clients, such as by writing to event queues. In one embodiment, the foregoing process is performed for all services running on the server 202 regardless of which instance(s) 202a-202n are providing a given service.

SERVICES 106

[0036] As previously described herein, services are, generally, a logical abstraction for managing workloads. More specific to the context of embodiments of the invention, a service, such as service 106a-106n, has a name and a domain, and may have associated goals, service levels, priority, and high availability attributes. The work performed as part of a service includes any use or expenditure of computer resources, including, for example, CPU

processing time, storing and accessing data in volatile memory, read and writes from and/or to persistent storage (i.e. disk space), and use of network or bus bandwidth.

[0037] In one embodiment, a service is work that is performed by a database server during a session, and typically includes the work performed to process and/or compute queries that require access to a particular database. The term query as used herein refers to a statement that conforms to a database language, such as SQL, and includes statements that specify operations to add, delete, or modify data and create and modify database objects, such as tables, objects views, and executable routines. A system, including a clustered computing system, may support many services.

[0038] Services can be provided by one or more database server instances. Thus, multiple server instances may work together to provide a service to a client. In FIG. 1, service 106a (e.g., FIN) is depicted, with dashed brackets, as being provided by instance 108a, service 106b (e.g., PAY) is depicted as being provided by instances 108a and 108b, and service 106n is depicted as being provided by instances 108a-108n.

[0039] Generally, the techniques described herein are as service-centric, where events occurring within server 104 can be identified and/or characterized based on the service(s) which is affected by the event. The payload of notification events is described hereafter.

NOTIFICATION SYSTEM

[0040] In one embodiment, a notification system as described hereafter is used to publish performance related information to client subscribers. However, how the performance information is published may vary from implementation to implementation, and is not limited to the notification system described.

[0041] In general, a daemon is a process that runs in the background and that performs a specified operation at predefined times or in response to certain events. In general, an event is an action or occurrence whose posting is detected by a process. Notification service daemon 118 is a process that receives alert and advisory information from server 104, such as from background manageability monitors that handle automatic management functions or clusterware that is configured to manage the cluster of instances 106a-106n. The server 104 posts service level performance events automatically and periodically, for subscribers to such events, such as runtime load balancing clients 102a-102n. In one embodiment, service level performance events are posted periodically based on the service request rate.

[0042] Notification service daemon 118 has a publisher-subscriber relationship with event handler 120 through which service performance information that is received by daemon 118 from server 104 is transmitted as work distribution advisory events to event handler 120.

In general, an event handler is a function or method containing program statements that are executed in response to an event. In response to receiving event information from daemon 118, event handler 120 at least passes along the event type and attributes, which are described herein. A single event handler 120 is depicted in FIG. 1 as serving all subscribers. However, different event handlers may be associated with different subscribers. The manner in which handling of advisory events is implemented by various subscribers to such events is unimportant, and may vary from implementation to implementation.

[0043] For a non-limiting example, notification service daemon 118 may use the Oracle Notification System (ONS) API, which is a messaging mechanism that allows application components based on the Java 2 Platform, Enterprise Edition (J2EE) to create, send, receive, and read messages.

[0044] "Subscribers" represents various entities that may subscribe to and respond to notification events for various respective purposes. Non-limiting examples of subscribers include clients 102a-102n, connection pool managers, mid-tier applications, batch jobs, callouts, paging and alert mechanisms, high availability logs, and the like.

SERVER DERIVATION OF WORK DISTRIBUTION

SERVICE MEASURES

[0045] A performance metric is data that indicates the quality of performance realized by services, for one or more resources. A performance metric of a particular type that can be used to gauge a characteristic or condition that indicates a service level of performance is referred to herein as a service measure. Service measures include, for example, completed work per second, elapsed time for completed calls, resource consumption and resource demand, wait events, and the like, some of which are described in more detail herein. Service measures are automatically maintained, for every service.

[0046] One approach to generating performance metrics, including service-based performance metrics on which service measures are based, which may be used for load balancing across a database cluster, are described in U.S. Patent Application No. 10/917,715 filed on August 12, 2004, entitled "Managing Workload By Service", which is incorporated by this reference in its entirety for all purposes as if fully disclosed herein.

[0047] For example, a background process may generate performance metrics from performance statistics that are generated for each session and service hosted on a database instance. Like performance metrics, performance statistics can indicate a quality of performance. However, performance statistics, in general, include more detailed information about specific uses of specific resources. Performance statistics include, for example, how

much time CPU time was used by a session, the throughput of a call, the number of calls a session made, the response time required to complete the calls for a session, how much CPU processing time was used to parse queries for the session, how much CPU processing time was used to execute queries, how many logical and physical reads were performed for the session, and wait times for input and output operations to various resources, such as wait times to read or write to a particular set of data blocks. Performance statistics generated for a session are aggregated by services and service subcategories (e.g. module, action) associated with the session.

COMPUTING A PERFORMANCE GRADE

[0048] Performance grades can be computed in a variety of different ways. Generally, the server allows an administrator to specify a service level goal for each service. This goal defines which service measures are important for the service, e.g., response time measures or throughput measures. Within a service, it is generally assumed that clients will route work to instances in proportion to the performance grade of each instance. This section presents examples of performance grade computations.

[0049] When computing a performance grade, various problems can occur that may prevent the grade from being meaningful. Thus, in addition to supplying a performance grade, an enumerated value that describes additional information about the grade is also provided. The possible flags include:

[0050] GOOD – The performance grade was computed for this instance.

[0051] VIOLATING – The service on this instance is in violation of the service's service level agreement. For example, and administrator may have specified that the service should always provide a two second response time and the response time is currently averaging three seconds. In this case, the performance grade has been computed and is meaningful.

[0052] NODATA – We may not have been able to obtain any performance metrics from an instance. For example, the instance may be in the process of crashing or it may be hung or otherwise unresponsive. This flag advises the client that we were unable to compute a performance grade, and the instance is probably not a good place to send work to.

[0053] UNKNOWN – During start up or during periods of low service utilization, we may not be able to compute a meaningful value for the performance grade. This flag indicates that we were unable to compute the performance grade, but that the instance is useable. In this case, the server would generally assume that this instance is pretty much the same as all the other GOOD or VIOLATING instances, and compute a performance grade based on that assumption.

[0054] In some cases, it is desirable to normalize a performance grade into a small range of values. This can be done by dividing the performance grade of each instance by the sum of the performance grades at all instances to obtain a value in the closed interval of [0, 1] for each instance. This value may be further scaled. For example, if this value is multiplied by 100, then the resulting value gives the percentage of the workload that should be handed to this instance.

BANDWIDTH

[0055] One service level goal may be to estimate the bandwidth of each instance, and publish that bandwidth as a performance grade. One way of estimating bandwidth is to measure the actual throughput of each instance and to estimate how busy the instance is. By definition, the bandwidth of an instance is the throughput that is obtained when the instance is 100% busy. Thus

Estimated Bandwidth = Throughput * 100 / %busy;

where "%busy" is a number in the half open interval (0, 100). Note that the UNKNOWN flag is used when either throughput or the estimated %busy is zero.

[0056] It is usually necessary to estimate how busy an instance is. One method is to measure how busy the CPU is. Other techniques include measuring how busy a specific resource class is, such as disk I/O or network bandwidth; or to use a weighted average across multiple resource classes.

[0057] It is common to use a moving average of both the throughput measurement (work completed per second) and the measurement of how busy the system is (e.g. CPU utilization).

[0058] When this goal is used, the system will attempt to balance the workload handed to each instance to be proportional to the bandwidth of the instance. That is, the system will attempt to keep the ratio throughput/bandwidth at each instance constant. Since throughput/bandwidth = throughput/throughput/%busy, this goal will attempt to make sure that %busy is constant across all nodes, i.e., all instances are kept relatively equally busy.

RESPONSE TIME

[0059] Another service level goal may be to publish the response time of the service on an instance as the performance grade of the instance. Response time is a measure of the elapsed time from when a unit of work arrived in the server to when the unit of work was completed. This includes time spent waiting for resources to become available as well as time spent using available resources.

[0060] When using response time as the performance grade, the resulting performance grade is not very stable, especially when workloads are high. Small changes in the workload,

such as in response to load balancing requests, can generate large changes in the performance grade. The performance grade values generated will tend to oscillate and converge to a desired value. The oscillations may be dampened by remembering the previously published performance grade and averaging the new grade with the previous grade.

[0061] In general, a "momentum" term may be introduced to control the amount of oscillation dampening and the rate of convergence:

new = (momentum * old + current) / (1 + momentum).

[0062] Using the response time in this fashion is attractive because it does not require estimating system utilization or figuring out which resources a service is using. So this policy is more likely to work across a broad range of workloads. On the other hand, this approach tends to respond to changes in the overall behavior of a system more slowly than the bandwidth metric does.

[0063] When this policy is used, the system will attempt to maintain throughput/response_time constant across all instances. Essentially, this will attempt to queue the same amount of work at each instance: At any point in time, if no new work came into the system, all of the instances would complete the currently queued work at the same time.

SPARE CAPACITY

[0064] Another possible service level goal is to estimate the spare capacity of each instance and publish that as the performance grade. One definition of spare capacity is the number of units of work that an instance could have performed during a measurement interval had there been work to perform. That is, estimate the idleness of the instance and divide that by the amount of time that it takes to complete a unit of work, not counting time spent waiting for resources.

[0065] If units of work are waiting for resources, then the idleness of the instance will be zero, so the calculation can be simplified by simply dividing the estimated idleness of the instance by the average response time of units of work executed at the instance.

[0066] Like the Response Time service level goal, the Spare Capacity goal has a tendency to generate oscillating performance grades. Thus, it is desirable to smooth out the oscillations by remembering the previously published performance grade and averaging that with the newly measured performance grade.

ADJUSTMENTS

[0067] The computations described above may be adjusted in various fashions. For example, it is desirable to send a small amount of work to each instance in order to measure

the performance of the instance. Also, it may be desirable to strongly avoid some instances in certain conditions (such as when the CPU is 100% busy, or all memory has been allocated, etc.) These types of adjustments will generally introduce oscillations when boundaries are reached and are thus more appropriate for those service level goals which explicitly dampen oscillations.

[0068] Also, performance grades should be values greater than zero. It may be necessary to adjust a grade of zero up slightly, or to use a slightly different computation. For example, when estimating spare capacity it may be desirable to use the formula "(%free + 0.01) / elapsed_time" instead of "%free / elapsed_time".

[0069] Generally, the server will want to estimate a performance grade for instances that are marked UNKNOWN. When estimating a performance grade, the server will generally assume that an UNKNOWN instance is similar to other instances whose performance grades can be computed. If performance grades cannot be computed for any instances, then the performance grade may be set to a constant non-zero value.

[0070] The server may be able to obtain good measurements for some, but not all, of the performance metrics it uses to compute a performance grade for an instance. In this case, the server may wish to estimate values for the missing performance metrics based on the behavior of other instances and then calculate the performance grade using the estimated metrics. For example, when estimating spare capacity for a new instance of a service, it may be possible to measure the idleness of the instance quite well, but not yet possible to measure the elapsed time at that instance because no work has yet been sent to the instance. In this case, the spare capacity of the UNKNOWN node could be estimated to be the average spare capacity of the known nodes; or, the elapsed time of the UNKNOWN node could be estimated as the same as the average elapsed times of the known nodes, and then divide the measured idleness by the estimated elapsed time.

[0071] When there are instances which are VIOLATING their service level agreements, it may be desirable to adjust performance grades. The server will generally want to distinguish between the case where all instances are VIOLATING (or about to violate) because the demand for the service is too high, and the case where one or a small number of instances are VIOLATING but other instances are working well.

[0072] The server may want to look at the performance metric that is being violated and compute the standard deviations of the performance metric across instances. VIOLATING instances whose performance metric is multiple standard deviations away from the average performance metric may have their performance grade forced to a very low value.

[0073] The performance metric used to compute outlying violators need not be the same as any of the performance metrics used to compute the performance grade. For example, if the service level goal may be BANDWIDTH, but the service level agreement may require that the response time stay below, say, 5 seconds. In this case, if the average response time was 3 seconds with a standard deviation of 3 seconds, the performance grade of an instance with a 20 second response time may be forced to a very small value.

PUBLISHING PERFORMANCE GRADES

[0074] The server may publish performance grades to its clients in various fashions. In one embodiment, each client subscriber subscribes to service events, where the event payload contains a performance grade for each instance offering the service. A separate event is published for each service. The posting process acquires these data once for all active services, as described in reference to FIG. 2, and then posts an event per service.

[0075] As discussed, in one embodiment notification service daemon 118 has a publisher/subscriber relationship with event handler 120, through which certain event information that is received by daemon 118 from database server 104 is transmitted to event handler 120. In response to receiving event information from daemon 118, event handler 120 invokes a method of connection pool manager 114, passing along the event type and property, which are described hereafter.

[0076] In one embodiment, a message format that is used for communicating service performance information in an event payload, comprises name/value pairs. Specifically, a service event may comprise the following:

Event Type = performance grades (e.g., "Database/Service/Grades/$service_name");

Version = version number of the publication protocol;

Database name = unique name identifying database;

Grades tuple (repeating for plurality of instances offering service)= instance name, flag (i.e., GOOD, VIOLATING, NODATA, UNKNOWN), grade (i.e., work distribution percentage); and

Timestamp = time event was calculated at the server.

[0077] Different techniques may be used to determine the frequency with which performance grades are published. In one embodiment, performance grades may be published at regularly scheduled intervals (e.g. every 30 seconds). In another embodiment, performance grades are only published when they change significantly. In another embodiment, performance grades are generated frequently (e.g. every 3 seconds), and only published when they change significantly, or if they haven't been published recently. The

rate at which performance grades are published may also be made dependent on the work load, i.e., infrequent publications at low work loads and frequent publications at high work loads.

USING PERFORMANCE GRADES

[0078] A client may use performance grades, in addition to other information available to the client to decide how to route each unit of work. In one embodiment, the client would use information available to the client to select which instances are eligible to receive the work, use the performance grades to decide how much work to send to the eligible instances, and select an eligible instance based on those percentages.

[0079] For example, the client might pay attention to which instances the client currently had idle connections. Or the client might implement a form of locality of reference, which would restrict the collection of eligible instances. After a set of eligible instances have been selected, if some of those instances have a NODATA performance grade and others do not, the NODATA instances would be eliminated from consideration.

[0080] The fraction of the work that the client should send to each of the remaining eligible instances is given by computing the performance grade of an instance divided by the sum of the performance grades of all instances. The client can then use a random number generator or other technique to select an eligible instance such that the probability of selecting an instance is equal to the faction of work that should be sent to that instance.

[0081] The client may need to perform some amount of work to make it likely that instances can be considered eligible. For example, in a connection pool, the client may close connections to instances that have a large number of idle connections, and open connections at instances that have few idle connections. Or the client may decide to close idle connections and create new connections to try and keep the number of open connections to each instance proportional to the performance grade of the instance.

[0082] Depending on the protocol defined between the client and server, the client may need to estimate the performance grade of instances that are marked with the UNKNOWN or NODATA flags. Since NODATA instances will only be used when all eligible instances have NODATA, these instances can be set to have a constant performance grade, for example, 1. If all instances are marked UNKNOWN or NODATA, then the UNKNOWN instances can have their performance grade set to a constant value as well. Otherwise, the client should set the performance grade of UNKNOWN instances to the average performance grade of GOOD and VIOLATING instances.

A METHOD FOR DETERMINING WORK ROUTING

[0083] FIG. 3 is a flow diagram that illustrates a method for determining how much work to route to nodes in a distributed computing system, according to an embodiment of the invention. The distributed computing system comprises a plurality of computing nodes that each hosts a server instance that provides a service that performs work, as described in reference to FIG. 1. In some contexts, such distributed computing systems are also referred to as computing clusters and/or computing grids. In one embodiment, the method depicted in FIG. 3 is performed by a software application executing on an electronic computing system, such as computer system 400 of FIG. 4. For example, the method may be performed by a database server, such as any of instances 108a-108n of FIG. 1 or instances 201a-202n of FIG. 2.

[0084] At block 302, receive, from each of a plurality of the server instances, a current moving average of a performance metric associated with a service. For example, manageability monitor 203a of instance 202a of server 202 (FIG. 2) receives the moving average of the completed work per second, or of the elapsed time for completed calls, from each of the manageability monitors 203b-203n of instances 202b-202n. Such information may be received, for example, via an inter-process communication mechanism.

[0085] At block 304, compute a performance grade for each of the plurality of instances. In one embodiment, computation of the performance grade includes applying one or more weighting factors to the respective moving averages of the performance metric. The weighting factors are based, at least in part, on the available resource capacity associated with the respective nodes on which each instance is executing. As described herein for embodiments of the invention, the weighting factors may be based on any one or more of the available CPU processing, IO processing and network intercommunication processing that is available from/to each node. Further as described herein, the weighting factor may be further based on the resource demand for the service, i.e., the amount of resources required by the service. For example, the weighting factors may be based on any one or more of the required CPU processing, IO processing, and network intercommunication processing that is required to execute the service to perform work.

[0086] At block 306, compute, based on the respective performance grades, a percentage of work to route to each of the instances. As described herein, the performance grades for each instance may be normalized relative to the plurality of instances, so that percentages are derived which advise distribution of work efficiently and intelligently across the nodes. For example, the service time and/or throughput associated with each node may be computed as

described herein, depending on the service goal, and normalized to derive respective work distribution percentages in accordance with the goal.

[0087] At block 308, publish the percentages to one or more subscribing clients so that the clients can use the advise contained in the percentages to route work intelligently among the system nodes. For example, goodness events may be queued and published via a notification system, such as notification service daemon 118 and event handler 120 of FIG. 1.

IMPLEMENTATION MECHANISMS

[0088] The approach for runtime load balancing of work across a clustered computing system, as described herein, may be implemented in a variety of ways and the invention is not limited to any particular implementation. The approach may be integrated into a system or a device, or may be implemented as a stand-alone mechanism. Furthermore, the approach may be implemented in computer software, hardware, or a combination thereof.

HARDWARE OVERVIEW

[0089] FIG. 4 is a block diagram that depicts a computer system 400 upon which an embodiment of the invention may be implemented. Computer system 400 includes a bus 402 or other communication mechanism for communicating information, and a processor 404 coupled with bus 402 for processing information. Computer system 400 also includes a main memory 406, such as a random access memory (RAM) or other dynamic storage device, coupled to bus 402 for storing information and instructions to be executed by processor 404. Main memory 406 also may be used for storing temporary variables or other intermediate information during execution of instructions to be executed by processor 404. Computer system 400 further includes a read only memory (ROM) 408 or other static storage device coupled to bus 402 for storing static information and instructions for processor 404. A storage device 410, such as a magnetic disk or optical disk, is provided and coupled to bus 402 for storing information and instructions.

[0090] Computer system 400 may be coupled via bus 402 to a display 412, such as a cathode ray tube (CRT), for displaying information to a computer user. An input device 414, including alphanumeric and other keys, is coupled to bus 402 for communicating information and command selections to processor 404. Another type of user input device is cursor control 416, such as a mouse, a trackball, or cursor direction keys for communicating direction information and command selections to processor 404 and for controlling cursor movement on display 412. This input device typically has two degrees of freedom in two axes, a first axis (e.g., x) and a second axis (e.g., y), that allows the device to specify positions in a plane.

[0091] The invention is related to the use of computer system 400 for implementing the techniques described herein. According to one embodiment of the invention, those techniques are performed by computer system 400 in response to processor 404 executing one or more sequences of one or more instructions contained in main memory 406. Such instructions may be read into main memory 406 from another machine-readable medium, such as storage device 410. Execution of the sequences of instructions contained in main memory 406 causes processor 404 to perform the process steps described herein. In alternative embodiments, hard-wired circuitry may be used in place of or in combination with software instructions to implement the invention. Thus, embodiments of the invention are not limited to any specific combination of hardware circuitry and software.

[0092] The term "machine-readable medium" as used herein refers to any medium that participates in providing instructions to processor 404 for execution. Such a medium may take many forms, including but not limited to, non-volatile media, volatile media, and transmission media. Non-volatile media includes, for example, optical or magnetic disks, such as storage device 410. Volatile media includes dynamic memory, such as main memory 406. Transmission media includes coaxial cables, copper wire and fiber optics, including the wires that comprise bus 402. Transmission media can also take the form of acoustic or light waves, such as those generated during radio-wave and infra-red data communications.

[0093] Common forms of machine-readable media include, for example, a floppy disk, a flexible disk, hard disk, magnetic tape, or any other magnetic medium, a CD-ROM, any other optical medium, punchcards, papertape, any other physical medium with patterns of holes, a RAM, a PROM, and EPROM, a FLASH-EPROM, any other memory chip or cartridge, a carrier wave as described hereinafter, or any other medium from which a computer can read.

[0094] Various forms of machine-readable media may be involved in carrying one or more sequences of one or more instructions to processor 404 for execution. For example, the instructions may initially be carried on a magnetic disk of a remote computer. The remote computer can load the instructions into its dynamic memory and send the instructions over a telephone line using a modem. A modem local to computer system 400 can receive the data on the telephone line and use an infra-red transmitter to convert the data to an infra-red signal. An infra-red detector can receive the data carried in the infra-red signal and appropriate circuitry can place the data on bus 402. Bus 402 carries the data to main memory 406, from which processor 404 retrieves and executes the instructions. The instructions received by main memory 406 may optionally be stored on storage device 410 either before or after execution by processor 404.

[0095] Computer system 400 also includes a communication interface 418 coupled to bus 402. Communication interface 418 provides a two-way data communication coupling to a network link 420 that is connected to a local network 422. For example, communication interface 418 may be an integrated services digital network (ISDN) card or a modem to provide a data communication connection to a corresponding type of telephone line. As another example, communication interface 418 may be a local area network (LAN) card to provide a data communication connection to a compatible LAN. Wireless links may also be implemented. In any such implementation, communication interface 418 sends and receives electrical, electromagnetic or optical signals that carry digital data streams representing various types of information.

[0096] Network link 420 typically provides data communication through one or more networks to other data devices. For example, network link 420 may provide a connection through local network 422 to a host computer 424 or to data equipment operated by an Internet Service Provider (ISP) 426. ISP 426 in turn provides data communication services through the world wide packet data communication network now commonly referred to as the "Internet" 428. Local network 422 and Internet 428 both use electrical, electromagnetic or optical signals that carry digital data streams. The signals through the various networks and the signals on network link 420 and through communication interface 418, which carry the digital data to and from computer system 400, are exemplary forms of carrier waves transporting the information.

[0097] Computer system 400 can send messages and receive data, including program code, through the network(s), network link 420 and communication interface 418. In the Internet example, a server 430 might transmit a requested code for an application program through Internet 428, ISP 426, local network 422 and communication interface 418.

[0098] The received code may be executed by processor 404 as it is received, and/or stored in storage device 410, or other non-volatile storage for later execution. In this manner, computer system 400 may obtain application code in the form of a carrier wave.

EXTENSIONS AND ALTERNATIVES

[0099] Alternative embodiments of the invention are described throughout the foregoing description, and in locations that best facilitate understanding the context of the embodiments. Furthermore, the invention has been described with reference to specific embodiments thereof. It will, however, be evident that various modifications and changes may be made thereto without departing from the broader spirit and scope of the invention. For example, embodiments of the invention are described herein in the context of a database

server; however, the described techniques are applicable to any distributed computing system over which system connections are allocated or assigned, such as with a system configured as a computing cluster or a computing grid. Therefore, the specification and drawings are, accordingly, to be regarded in an illustrative rather than a restrictive sense.

[0100] In addition, in this description certain process steps are set forth in a particular order, and alphabetic and alphanumeric labels may be used to identify certain steps. Unless specifically stated in the description, embodiments of the invention are not necessarily limited to any particular order of carrying out such steps. In particular, the labels are used merely for convenient identification of steps, and are not intended to specify or require a particular order of carrying out such steps.

CLAIMS

What is claimed is:

1. A computer-implemented method for determining how much work to route to computing nodes in a computing system that comprises a plurality of nodes that each hosts a server instance that provides a service that performs work, the method comprising:

based on a current moving average of a performance metric, from each of a plurality of server instances that provide a particular service, that is associated with the particular service, computing a performance grade for each of the plurality of server instances; and

computing, based on the respective performance grades, a percentage of work to route to each of the plurality of server instances.

2. The method of Claim 1, further comprising:

publishing the percentages to one or more subscribing clients; and

routing work, by at least one of the subscribing clients and based on the percentages, to particular nodes in the computing system.

3. The method of Claim 2, wherein the step of publishing includes posting events to an event queue, wherein each event is associated with one particular service.

4. The method of Claim 2, wherein the step of publishing the performance grades includes periodically publishing the percentages, wherein the period is based on a rate at which requests for the service are received.

5. The method of Claim 2, further comprising:

publishing a flag to the one or more subscribing clients, in association with each percentage, wherein the flag indicates any one from the group consisting of (a) the performance grade was computed for this instance, (b) the service on this instance is violating a service level agreement associated with the service, (c) the performance grade was not computed for this instance, and (d) the performance grade was not computed for this instance, but work can be routed to this instance; and

routing work, by at least one of the subscribing clients and based on the percentage and associated flags, to nodes in the computing system.

6. The method of Claim 1, wherein the step of computing performance grades includes computing a performance grade based on a performance metric associated with response time of the service as provided by the server instance.

7. The method of Claim 1, wherein the step of computing performance grades includes computing a performance grade based on a performance metric associated with throughput of the respective node on which the server instances is executing.

8. The method of Claim 1, wherein the step of computing performance grades includes computing a performance grade based on a performance metric associated with spare capacity of the respective node on which the server instances is executing.

9. The method of Claim 1,

wherein the step of computing performance grades includes applying one or more weighting factors to the respective moving averages of the performance metric;

wherein the weighting factors are based, at least in part, on any one or more from the group consisting of available CPU processing, available IO processing, and available network communication processing within the computing system.

10. A machine-readable medium carrying one or more sequences of instructions which, when executed by one or more processors, causes the one or more processors to perform the method recited in Claim 1.

11. A machine-readable medium carrying one or more sequences of instructions which, when executed by one or more processors, causes the one or more processors to perform the method recited in Claim 2.

12. A machine-readable medium carrying one or more sequences of instructions which, when executed by one or more processors, causes the one or more processors to perform the method recited in Claim 3.

13. A machine-readable medium carrying one or more sequences of instructions which, when executed by one or more processors, causes the one or more processors to perform the method recited in Claim 4.

14. A machine-readable medium carrying one or more sequences of instructions which, when executed by one or more processors, causes the one or more processors to perform the method recited in Claim 5.

15. A machine-readable medium carrying one or more sequences of instructions which, when executed by one or more processors, causes the one or more processors to perform the method recited in Claim 6.

16. A machine-readable medium carrying one or more sequences of instructions which, when executed by one or more processors, causes the one or more processors to perform the method recited in Claim 7.

17. A machine-readable medium carrying one or more sequences of instructions which, when executed by one or more processors, causes the one or more processors to perform the method recited in Claim 8.

18. A machine-readable medium carrying one or more sequences of instructions which, when executed by one or more processors, causes the one or more processors to perform the method recited in Claim 9.

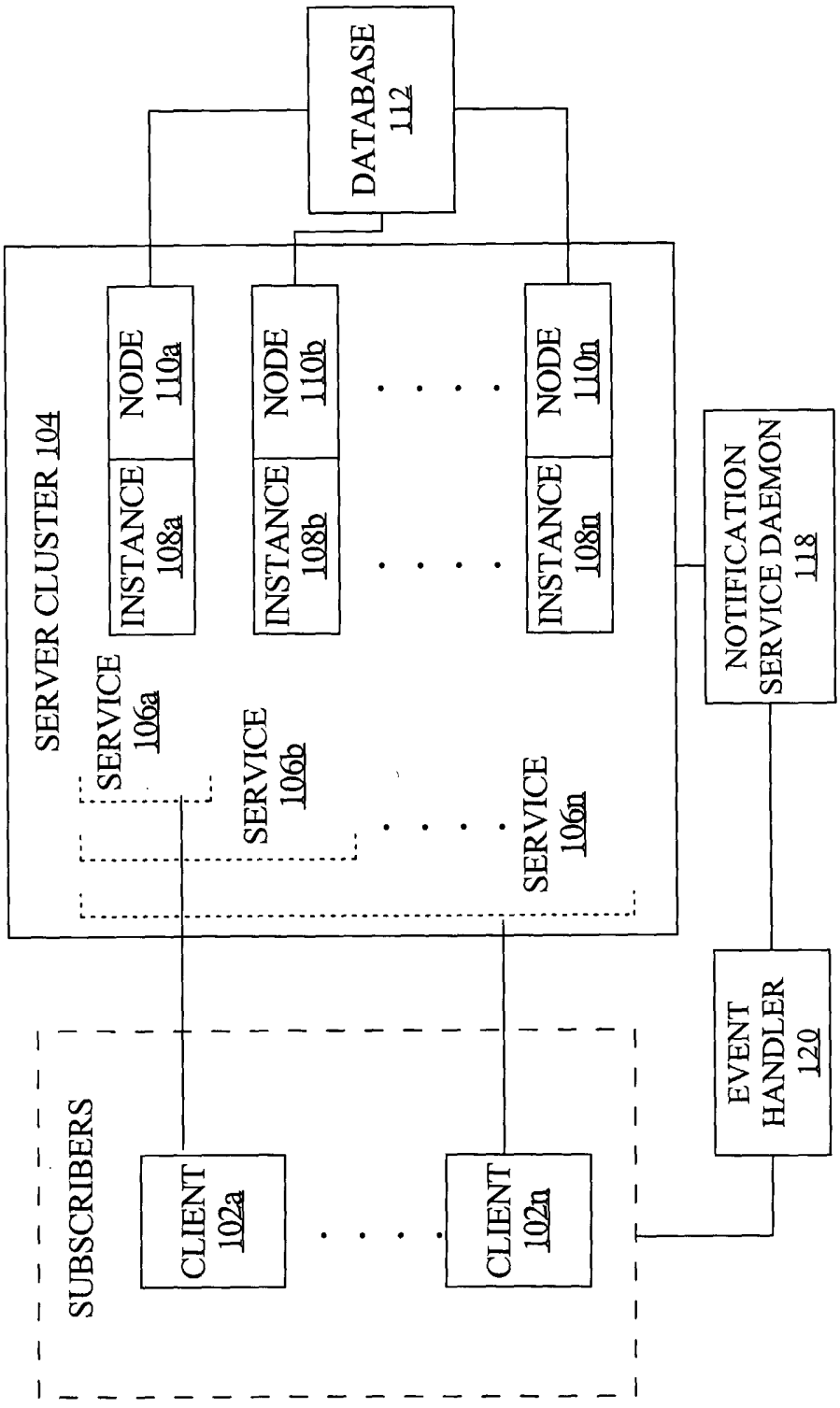

DATABASE 112
SERVER CLUSTER 104
NODE 110a
NODE 110b
NODE 110n
INSTANCE 108a
INSTANCE 108b
INSTANCE 108n
SERVICE 106a
SERVICE 106b
SERVICE 106n
NOTIFICATION SERVICE DAEMON 118
EVENT HANDLER 120
SUBSCRIBERS
CLIENT 102a
CLIENT 102n

FIG. 1

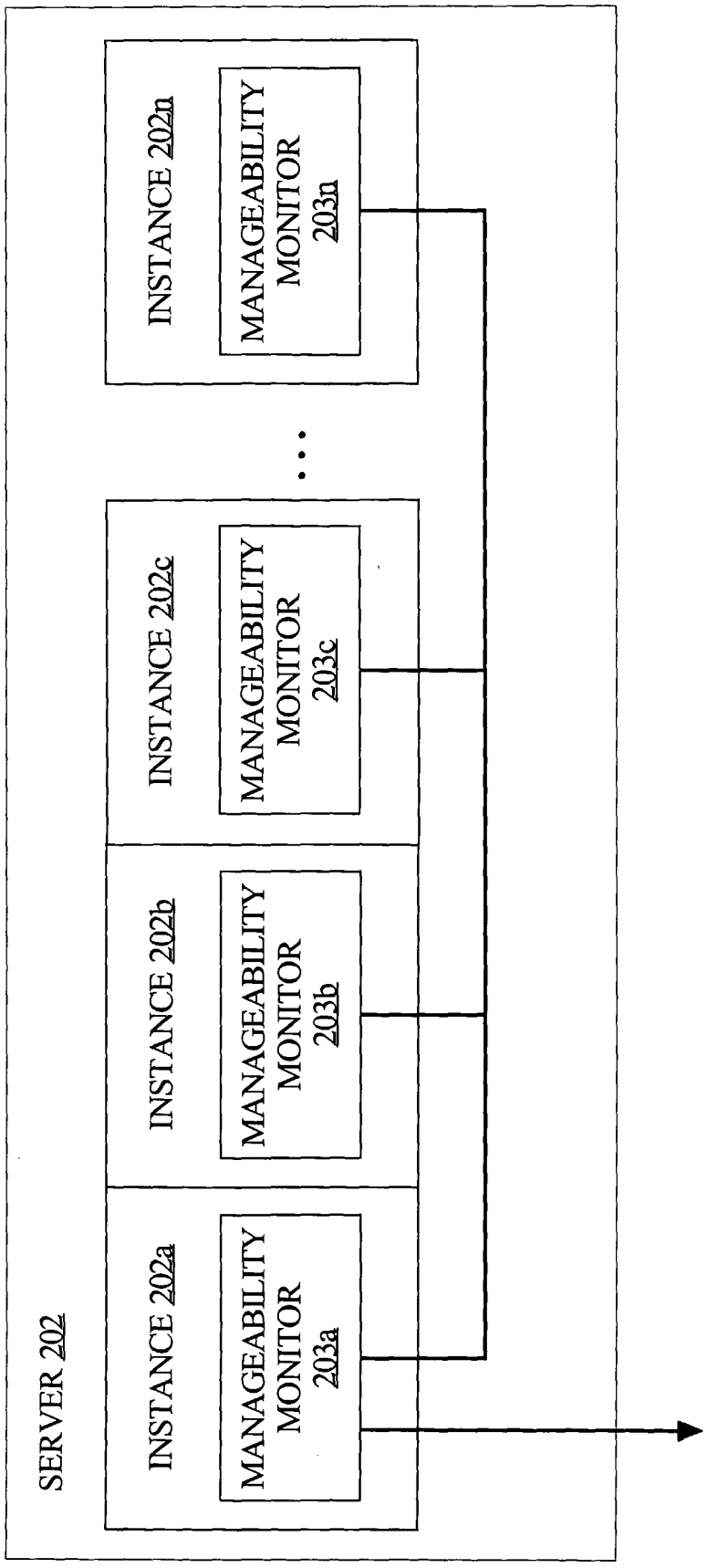

SERVER 202
INSTANCE 202a
MANAGEABILITY MONITOR 203a
INSTANCE 202b
MANAGEABILITY MONITOR 203b
INSTANCE 202c
MANAGEABILITY MONITOR 203c
INSTANCE 202n
MANAGEABILITY MONITOR 203n

FIG. 2

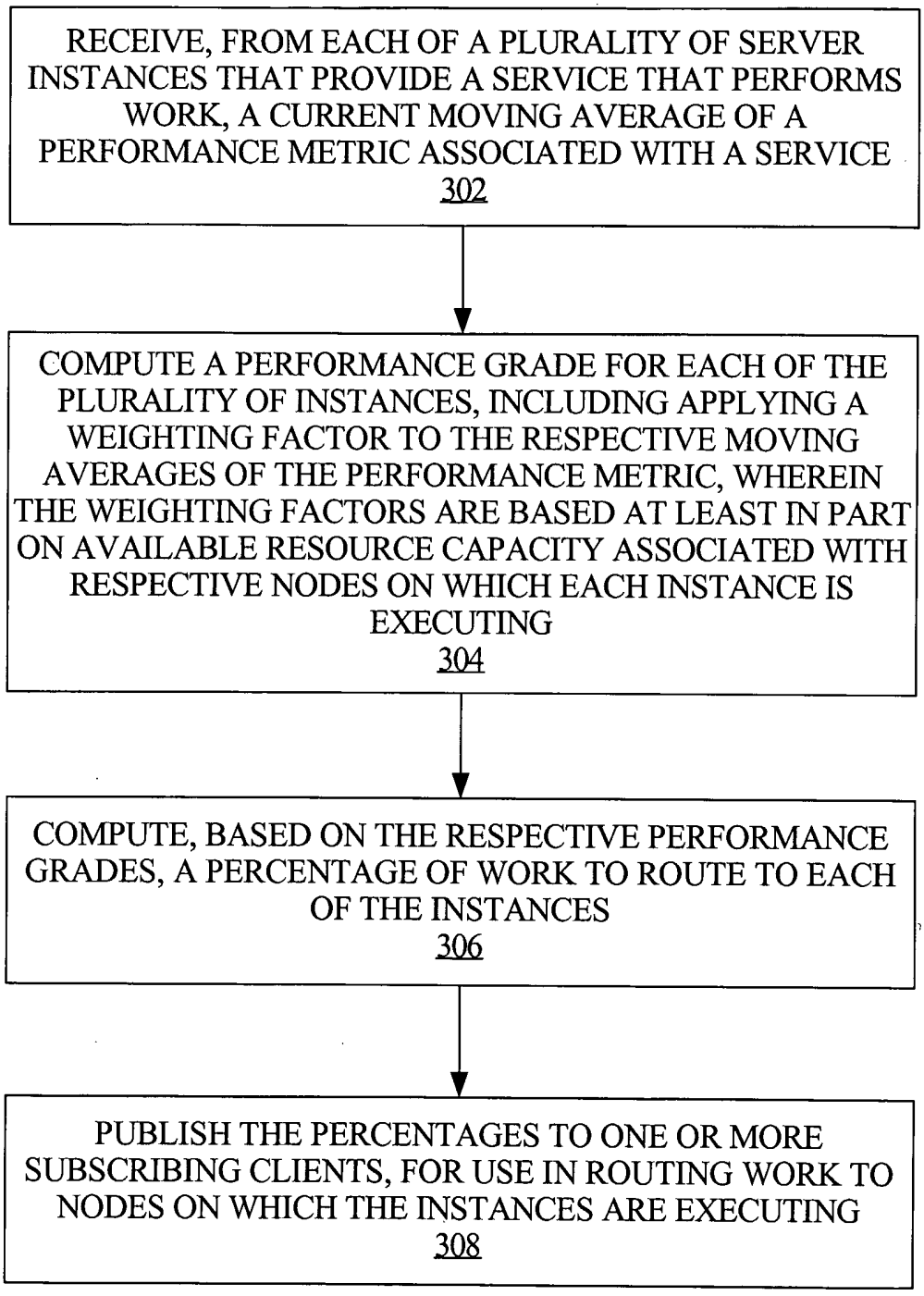

RECEIVE, FROM EACH OF A PLURALITY OF SERVER INSTANCES THAT PROVIDE A SERVICE THAT PERFORMS WORK, A CURRENT MOVING AVERAGE OF A PERFORMANCE METRIC ASSOCIATED WITH A SERVICE
302
COMPUTE A PERFORMANCE GRADE FOR EACH OF THE PLURALITY OF INSTANCES, INCLUDING APPLYING A WEIGHTING FACTOR TO THE RESPECTIVE MOVING AVERAGES OF THE PERFORMANCE METRIC, WHEREIN THE WEIGHTING FACTORS ARE BASED AT LEAST IN PART ON AVAILABLE RESOURCE CAPACITY ASSOCIATED WITH RESPECTIVE NODES ON WHICH EACH INSTANCE IS EXECUTING
304
COMPUTE, BASED ON THE RESPECTIVE PERFORMANCE GRADES, A PERCENTAGE OF WORK TO ROUTE TO EACH OF THE INSTANCES
306
PUBLISH THE PERCENTAGES TO ONE OR MORE SUBSCRIBING CLIENTS, FOR USE IN ROUTING WORK TO NODES ON WHICH THE INSTANCES ARE EXECUTING
308

FIG. 3

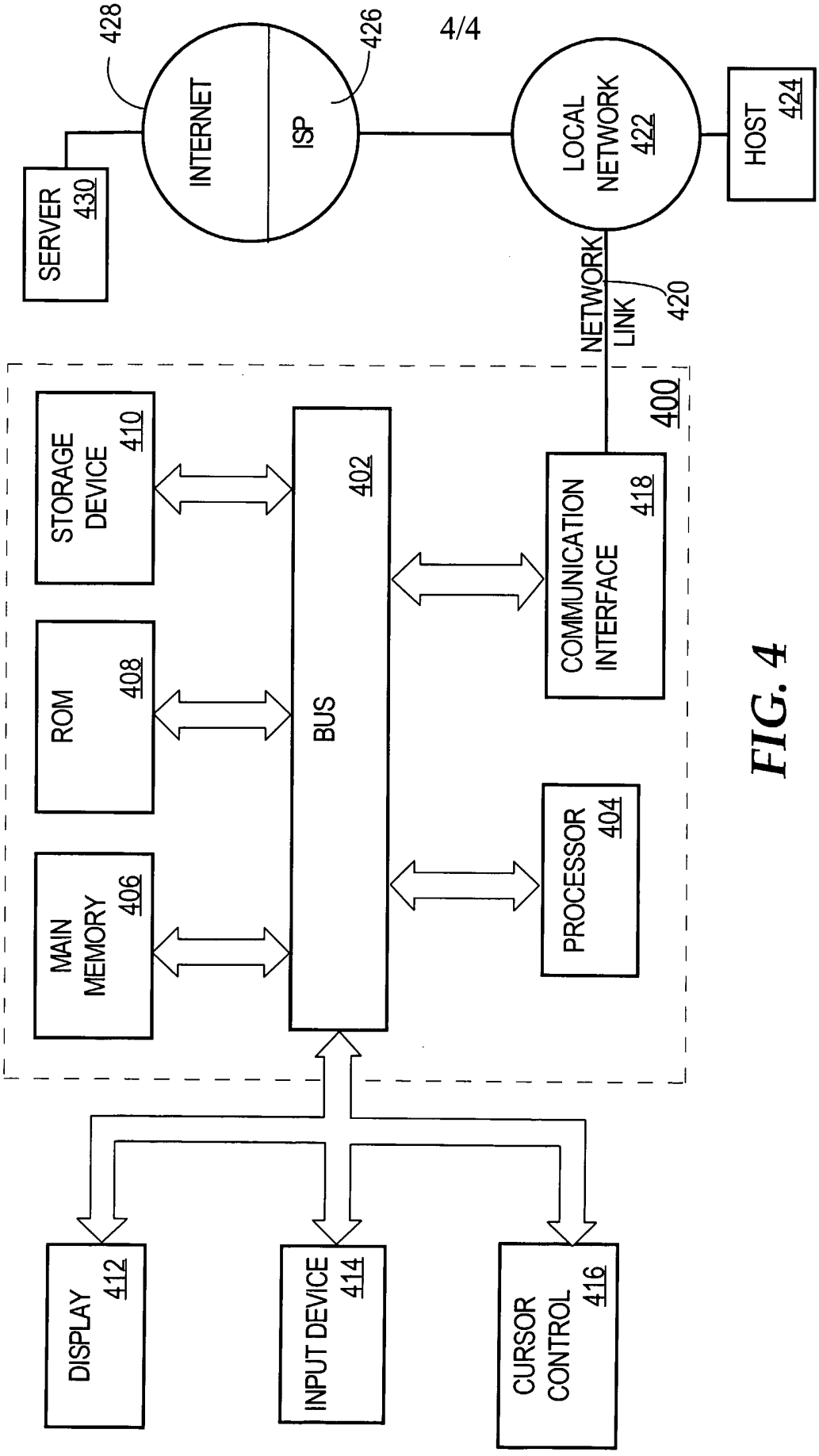

428
INTERNET
ISP
426
SERVER
430
LOCAL
NETWORK
422
HOST
424
NETWORK
LINK
420
400
STORAGE
DEVICE
410
402
BUS
COMMUNICATION
INTERFACE
418
ROM
408
MAIN
MEMORY
406
PROCESSOR
404
DISPLAY
412
INPUT DEVICE
414
CURSOR
CONTROL
416

FIG. 4

INTERNATIONAL SEARCH REPORT

International application No
PCT/US2005/025887

A. CLASSIFICATION OF SUBJECT MATTER
G06F9/46

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
G06F

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

Electronic data base consulted during the international search (name of data base and, where practical, search terms used)
EPO-Internal

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	WO 03/014929 A (NOKIA CORPORATION; PARTANEN, JUKKA, T) 20 February 2003 (2003-02-20) page 3 - page 4 page 7 - page 9	1,6-10, 15-18
Y	page 7 - page 9	2-5, 11-14
Y	US 2004/103195 A1 (CHALASANI NANCHARIAH RAGHU ET AL) 27 May 2004 (2004-05-27) page 2, paragraph 21 - paragraph 25 page 2, paragraph 27 - page 3, paragraph 30 page 6, paragraph 63 - page 7, paragraph 74 page 8, paragraph 86	2-5, 11-14

-/--

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

[X] See patent family annex.

* Special categories of cited documents :

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier document but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art.

"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
16 January 2006	06/02/2006

Name and mailing address of the ISA/	Authorized officer
European Patent Office, P.B. 5818 Patentlaan 2 NL - 2280 HV Rijswijk Tel. (+31-70) 340-2040, Tx. 31 651 epo nl, Fax: (+31-70) 340-3016	Dewyn, T

INTERNATIONAL SEARCH REPORT

International application No
PCT/US2005/025887

C(Continuation). DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	US 2003/039212 A1 (LLOYD MICHAEL A ET AL) 27 February 2003 (2003-02-27) page 14, paragraph 278	1-18
A	US 2004/064548 A1 (ADAMS SAMUEL S ET AL) 1 April 2004 (2004-04-01) the whole document	1-18
A	US 2003/135609 A1 (CARLSON MARK A ET AL) 17 July 2003 (2003-07-17) the whole document	1-18
A	WO 00/10084 A (MICROSOFT CORPORATION) 24 February 2000 (2000-02-24) the whole document	1-18

INTERNATIONAL SEARCH REPORT

Information on patent family members

International application No
PCT/US2005/025887

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
WO 03014929 A	20-02-2003	EP 1415227 A1	06-05-2004
		US 2004205761 A1	14-10-2004
US 2004103195 A1	27-05-2004	NONE	
US 2003039212 A1	27-02-2003	NONE	
US 2004064548 A1	01-04-2004	AU 2003267663 A1	23-04-2004
		WO 2004032453 A1	15-04-2004
US 2003135609 A1	17-07-2003	WO 03062983 A2	31-07-2003
WO 0010084 A	24-02-2000	NONE	