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

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
5 October 2017 (05.10.2017)

WIPO | PCT

(10) International Publication Number
WO 2017/167686 A1

(51) International Patent Classification:
G06F 9/50 (2006.01)

(21) International Application Number:
PCT/EP2017/057185

(22) International Filing Date:
27 March 2017 (27.03.2017)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
16305388.7 1 April 2016 (01.04.2016) EP

(71) Applicant: **ALCATEL LUCENT** [FR/FR]; 148/152 Route de la Reine, 92100 BOULOGNE-BILLANCOURT (FR).

(72) Inventors: **JIAO, Lei**; 334 Deschutes Hall, 1202 University of Oregon, Eugene, Oregon 97403 (US). **LUGONES, Diego**; Alcatel-Lucent Ireland Ltd, BLANCHARDSTOWN BUS TECH PK, SNUGBOROOUGH RD, DUBLIN, 15 (IE). **JIN, Yue**; Alcatel-Lucent Ireland Ltd, BLANCHARDSTOWN BUS TECH PK, SNUGBOROOUGH RD, DUBLIN, 15 (IE). **SALA, Alessandra**; Alcatel-Lucent Ireland Ltd, BLANCHARDSTOWN BUS TECH PK, SNUGBOROOUGH RD, DUBLIN, 15 (IE). **HILT, Volker Friedrich**; Alcatel-Lucent Ireland Ltd, BLANCHARDSTOWN BUS TECH PK, SNUGBOROOUGH RD, DUBLIN, 15 (IE).

(74) Agent: **BERTHIER, Karine**; Alcatel-Lucent International, 148/152 Route de la Reine, 92100 BOULOGNE-BILLANCOURT (FR).

(81) Designated States *(unless otherwise indicated, for every kind of national protection available)*: AE, AG, AL, AM, AO, AT, AU, AZ, BA, BB, BG, BH, BN, BR, BW, BY, BZ, CA, CH, CL, CN, CO, CR, CU, CZ, DE, DJ, DK, DM, DO, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, GT, HN, HR, HU, ID, IL, IN, IR, IS, JP, KE, KG, KH, KN, KP, KR, KW, KZ, LA, LC, LK, LR, LS, LU, LY, MA, MD, ME, MG, MK, MN, MW, MX, MY, MZ, NA, NG, NI, NO, NZ, OM, PA, PE, PG, PH, PL, PT, QA, RO, RS, RU, RW, SA, SC, SD, SE, SG, SK, SL, SM, ST, SV, SY, TH, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, ZA, ZM, ZW.

(84) Designated States *(unless otherwise indicated, for every kind of regional protection available)*: ARIPO (BW, GH, GM, KE, LR, LS, MW, MZ, NA, RW, SD, SL, ST, SZ, TZ, UG, ZM, ZW), Eurasian (AM, AZ, BY, KG, KZ, RU,

[Continued on next page]

(54) Title: A METHOD AND SYSTEM FOR SCALING RESOURCES, AND A COMPUTER PROGRAM PRODUCT

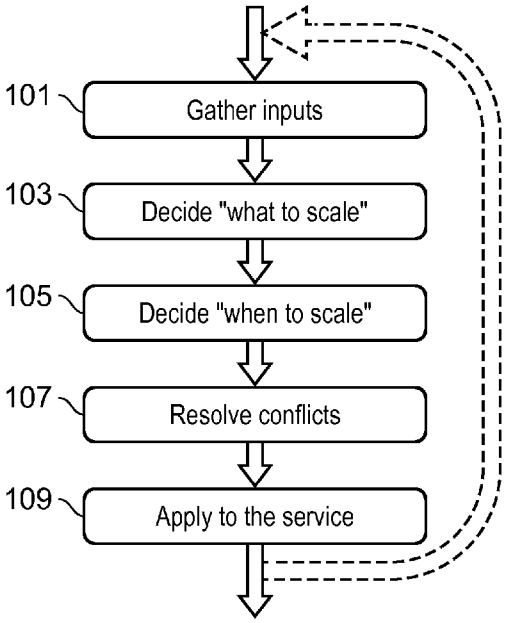


FIG. 1

(57) Abstract: A method of determining a set of virtual machine instances in a virtualized system to be scaled in response to a variation in workload of a service or application component, the method comprising generating respective sets of aggregated metrics for respective ones of multiple virtual machines, generating a directed graph in which nodes representing a resource of one or more of the multiple virtual machines are linked using one or more edges, the weight of each of the edges representing a time-lagged partial correlation value and a time lag value representing a measure of when to scale one resource relative to another, determining the set of virtual machine instances to be scaled by removing edges of the directed graph whose weight falls below a threshold measure and removing nodes whose incident edges have been removed, and scaling the weight of remaining edges of the directed graph relative to a selected node of the graph.

WO 2017/167686 A1

TJ, TM), European (AL, AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HR, HU, IE, IS, IT, LT, LU, LV, MC, MK, MT, NL, NO, PL, PT, RO, RS, SE, SI, SK, SM, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, KM, ML, MR, NE, SN, TD, TG).

Published:

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

A METHOD AND SYSTEM FOR SCALING RESOURCES, AND A COMPUTER PROGRAM PRODUCT

TECHNICAL FIELD

Aspects relate, in general, to a method and system for scaling resources, and a computer program product.

BACKGROUND

In a virtualized system comprising multiple virtual machine (VM) instances executing over physical hardware, multiple ones of the VMs can interact or be provisioned to execute a service or application or a component thereof. Typically such services or applications have variable loads that lead to variable resource requirements. If resources are too few and the service is thus under provisioned, the one or more service elements may become overloaded and fail to meet required performance criteria. If the service is overprovisioned, resources are wasted and costs are increased above the optimal level.

Accordingly, it is typical for one or more of the VMs for a service or application (service elements) to be scaled to suit the demand. More specifically, the resources consumed by a service (e.g., CPU, memory, storage, and network resource) can be adjusted in two ways: increasing or decreasing the number of VMs (horizontal scaling) and/or increasing or decreasing the amount of resources that are assigned to the VMs (vertical scaling).

Typically, resource scaling in response to a variation in the load of a system is reactive in that the performance of the service and the system running it is monitored and all elements related to a service or application are scaled up or down as required. However, when workload grows it does not necessarily mean that every VM needs more resources. It may be the case that only a critical subset of the VMs needs to adjust their resources. Furthermore, any VMs that need more resources may not require them simultaneously.

SUMMARY

According to an example, there is provided a method of determining a set of virtual machine instances in a virtualized system to be scaled in response to a variation in workload of a service or application component, the method comprising generating respective sets of aggregated metrics for respective ones of multiple virtual machines,

generating a directed graph in which nodes representing a resource of one or more of the multiple virtual machines are linked using one or more edges, the weight of each of the edges representing a time-lagged partial correlation value and a time lag value representing a measure of when to scale one resource relative to another, determining the set of virtual machine instances to be scaled by removing edges of the directed graph whose weight falls below a threshold measure and removing nodes whose incident edges have been removed, and scaling the weight of remaining edges of the directed graph relative to a selected node of the graph. Generating an aggregated metric can include determining, at periodic intervals, utilisation of one or more resources by respective ones of the multiple virtual machines, and generating an aggregated vector sum of the resource utilisations for respective ones of the multiple virtual machines. A node of the directed graph can represent a resource of a virtual machine or an aggregated resource measure of a set of virtual machines that execute the same service or application component. The method can further comprise, for respective ones of the nodes of the directed graph, calculating a single-source shortest path towards every node in the directed graph and generating a vector of the calculated lengths, each vector comprising components representing the time lag of each node in the graph relative to a source node for the vector in question, whereby to provide a serialised set of time lags for the directed graph. The method can further comprise selecting a vector from the set with a minimum norm value, in the event that the selected vector includes negative valued components, adding the absolute value of the minimum valued component to components of the selected vector, and determining a resource to monitor based on the component of the selected vector with the minimum value. The method can further comprise resolving conflicts in the case of horizontal scaling for the application or service component by adding resources of a virtual machine at the earliest time required by any of the resources or removing resources of a virtual machine at the latest time required by any of the resources, or by adding or removing a virtual machine instance for a resource.

According to an example, there is provided a system including multiple virtual machines in program execution on physical computing hardware supported by a hypervisor, the multiple virtual machines being operable to execute a service or application component in a cloud infrastructure, the system including an apparatus forming an orchestration engine or server to generate respective sets of aggregated metrics for respective ones of multiple virtual machines, generate a directed graph in which nodes representing a resource of one or more of the multiple virtual machines are linked using one or more edges, the weight of each of the edges representing a

time-lagged partial correlation value and a time lag value representing a measure of when to scale one resource relative to another, determine the set of virtual machine instances to be scaled by removing edges of the directed graph whose weight falls below a threshold measure and removing nodes whose incident edges have been removed, and scale the weight of remaining edges of the directed graph relative to a selected node of the graph.

The orchestration engine is operable to determine, at periodic intervals, utilisation of one or more resources by respective ones of the multiple virtual machines, and generate an aggregated vector sum of the resource utilisations for respective ones of the multiple virtual machines. A node of the directed graph can represent a resource of a virtual machine or an aggregated resource measure of a set of virtual machines that execute the same service or application component. The orchestration engine can, for respective ones of the nodes of the directed graph, calculate a single-source shortest path towards every node in the directed graph and generate a time-lag vector of the calculated lengths, each vector comprising components representing the time lag of each node in the graph relative to a source node for the vector in question, whereby to provide a serialised set of time lags for the directed graph. The orchestration engine can select a vector from the set with a minimum norm value, in the event that the selected vector includes negative valued components, add the absolute value of the minimum valued component to components of the selected vector, and determine a resource to monitor based on the component of the selected vector with the minimum value. The orchestration engine can resolve conflicts in the case of horizontal scaling for the application or service component by adding resources of a virtual machine at the earliest time required by any of the resources or removing resources of a virtual machine at the latest time required by any of the resources, or by adding or removing a virtual machine instance for a resource. The orchestration engine can generate a resource request for the service or application component providing an instruction to add or remove one or more virtual machine instances or increase or decrease the amount of resources available to one or more virtual machine instances at times selected according to the time lag values specified in a time-lag vector.

According to an example, there is provided a computer program product, comprising a computer usable medium having computer readable program code embodied therein, said computer readable program code adapted to be executed to implement a method of determining a set of virtual machine instances in a virtualized system to be scaled in

response to a variation in workload of a service or application component as provided herein.

BRIEF DESCRIPTION OF THE DRAWINGS

Embodiments will now be described, by way of example only, with reference to the accompanying drawings, in which:

Figure 1 is a schematic representation of a method according to an example;

Figure 2 is a schematic representation of a virtualised system according to an example;

Figures 3a-c are schematic representations relating to the calculation of a partial correlation for a time lag of 2 between two CPU utilizations X and Y excluding the influence of a workload;

Figure 4 is a schematic representation of a partial correlation graph according to an example;

Figures 5a and 5b are schematic representations of two path length vectors according to an example;

Figure 6 is a schematic representation of conflict resolution according to an example;

Figure 7 is a schematic representation of conflict resolution according to an example; and

Figure 8 is a schematic representation of an apparatus such as an orchestration engine or server according to an example.

DESCRIPTION

Example embodiments are described below in sufficient detail to enable those of ordinary skill in the art to embody and implement the systems and processes herein described. It is important to understand that embodiments can be provided in many alternate forms and should not be construed as limited to the examples set forth herein.

Accordingly, while embodiments can be modified in various ways and take on various alternative forms, specific embodiments thereof are shown in the drawings and

described in detail below as examples. There is no intent to limit to the particular forms disclosed. On the contrary, all modifications, equivalents, and alternatives falling within the scope of the appended claims should be included. Elements of the example embodiments are consistently denoted by the same reference numerals throughout the drawings and detailed description where appropriate.

The terminology used herein to describe embodiments is not intended to limit the scope. The articles “a,” “an,” and “the” are singular in that they have a single referent, however the use of the singular form in the present document should not preclude the presence of more than one referent. In other words, elements referred to in the singular can number one or more, unless the context clearly indicates otherwise. It will be further understood that the terms “comprises,” “comprising,” “includes,” and/or “including,” when used herein, specify the presence of stated features, items, steps, operations, elements, and/or components, but do not preclude the presence or addition of one or more other features, items, steps, operations, elements, components, and/or groups thereof.

Unless otherwise defined, all terms (including technical and scientific terms) used herein are to be interpreted as is customary in the art. It will be further understood that terms in common usage should also be interpreted as is customary in the relevant art and not in an idealized or overly formal sense unless expressly so defined herein.

Existing solutions for scaling an application or service (or component thereof) in a virtual infrastructure are typically prohibitive and inapplicable for large-scale complicated cloud services. Multiple service chains/paths, potentially geographically distributed, may exist within the service, which involve and intertwine many VMs making it difficult to identify which resources of which VMs are the right ones that need to be scaled, not to mention when to scale them.

Deploying and leveraging external tools such as traffic monitors and packet inspectors to detect the relationships between VMs or resources for scaling is also costly in terms of performance, due to the intensive resource contention among VMs in consolidated cloud environments. Furthermore, approaches that explore the global many-to-many communications are not scalable and become very limited in cloud environments where a huge number of tenant processes share virtualized resources.

Existing scaling processes fall broadly into three categories:

1) Scaling based on thresholds or rules:

This approach requires setting thresholds or scaling rules for metrics of each VM that runs a different service component. An example of a scaling rule can be "when the average CPU utilization becomes more than 90% for 5 minutes, booting 1 additional VM." One drawback of this method is that all VMs need to be monitored to check ongoing resource utilizations. In the case of many VMs in a large-scale service, this could be impossible and would require significant effort to determine proper scaling rules for each VM. Another drawback is that it is a reactive approach, i.e., VMs scale in response to workload variations. Booting new VMs requires a considerable amount of time during which the extra workload could not be served and the service quality consequently deteriorates.

2) Scaling based on profiling and modeling:

This approach automatically profiles VMs and builds models of (the number of) VMs against workload. It feeds trial workloads to the service, records the metrics of each VM, and builds a model which can be used to determine how to scale VMs for a real workload in a production environment. The disadvantage of this approach is that the profiling process itself is often resource-intensive and time-consuming, during which time the cloud service will be unable to serve the normal workload. Another shortcoming is that this approach, as well as the thresholds or rules approach as described above, ignores the possible relationships among VMs and treats every VM independently, which is not an accurate reflection of a large-scale service that is composed of multiple VMs.

3) Scaling based on communication dependency:

This approach detects and leverages the communication dependency among VMs for scaling. This method often defines and detects dependency based on communication, which is a very strong assumption and can miss hidden dependencies among VMs. Secondly, to determine the lines of communication between VMs, existing methods use flow or packet inspection, which is impossible in an environment with high resource contention. Thirdly, communication-based dependency is generally not a good indicator for scaling VMs. Two VMs communicating with each other does not always mean that they need to be scaled together, e.g., one VM, which scales for a varying

workload constantly sends log messages to the other VM, which may not need to be scaled no matter how the workload varies. That is, the fact that two VMs communicate with each other does not imply relevant resource dependency, and vice versa.

According to an example, a light-weight and efficient method is provided, which is un-intrusive and out of the critical data path. The method can dynamically detect a subset of resources to be scaled and determine when to scale that subset for a large-scale, complex cloud service that faces workload variations. Time-lagged partial correlation values are used, based upon which global scaling decisions can be made using only local VM metrics such as CPU, memory, storage and network utilization.

Information about resource utilization of a distributed service is determined, indirectly, by leveraging local resource metrics in order to build a global understanding of service utilization and how components interact with each other. This information allows creation of a service structure (e.g., a graph or a matrix) that can be used to orchestrate concurrent and accurate scaling actions on those service components that have direct impact on service performance.

This process is fast, scalable, and consumes negligible resource for execution. It involves local VM metrics and is out of the data path of a service and does not therefore profile or use external tools which can be intrusive and influence the normal operation of the service. For a source VM, other related VMs that need to be scaled when the specified source is scaled are determined, and the time at which these resources should be scaled is calculated. Thirdly, it is proactive and predictive, because the computation runs online, periodically and the future time when the scaling operations need to happen, captured by the time lag, is pre-calculated such that the scaling operations can start in advance and complete in time.

The present method and system is applicable to systems in which both horizontal and vertical scaling can be used. Figure 1 is a schematic representation of a method according to an example. Figure 1 provides a broad overview of certain steps taken in order to determine a subset of virtual machines to scale and generate data representing a time for scaling for each virtual machine. The specific steps are described in more detail below, but briefly, in block 101input information in the form of metrics for virtual machines is gathered and aggregated, subject to horizontal or vertical scaling. In block 103, resources to scale are identified by computing time-lagged partial correlation graphs, each of which corresponds to a type of resource such

as CPU, memory, etc. In the case of vertical scaling, a node (or vertex) in a graph represents the corresponding resource of a single VM; in the case of horizontal scaling, a vertex in a graph represents the corresponding resource of a set of VMs that run the same service components. Each edge in the graphs is weighted by a partial correlation value and a time lag. Those VMs that are not reflected in the graphs are the ones that do not need to adjust their resources.

In block 105, the time lags are serialized using a process to determine the shortest path between nodes in a generated directed graph representing the interactions between virtual machine. A vector norm calculation can be used in order to serialize the time lags. More specifically, the time lags are calibrated in the partial correlation graphs based on a carefully selected vertex, i.e., time lags (which is the value of when to scale one resource relative to when to scale the other) are converted to a value that is relative to a common base node such that it is known when to scale every other resource after the base resource scales.

In block 107, conflicts in decisions made by multiple metrics (in the case of horizontal scaling) are resolved. This is necessary only for horizontal scaling where a VM, rather than a resource, is the smallest block to manipulate. An adjustment of the time lags is performed to resolve the potential conflicts that different types of resources of the VM may desire inconsistent scaling decisions for the VM.

In block 109 resources are scaled and the service/application is operated according to the final result. The procedure outlined in the blocks of figure 1 can be executed periodically so that a cloud service always responds to workload variations dynamically.

Figure 2 is a schematic representation of a logical architecture of a virtualised system according to an example. In an example, a method as described herein can be embodied within an orchestration framework that operates a given virtualized and distributed service running in an arbitrary cloud infrastructure. A cloud infrastructure 200 provides a pool of computing, storage and networking resources 202 that are virtualized and shared across various services 201. Cloud environments such as OpenStack, Amazon EC2 and Google's Compute and Container Engines can be represented by this architecture, as well as others cloud vendors with Infrastructure as a Service (IaaS) offerings.

Herein, an 'instance' is a collection or sub-set of computing, storage and networking resources that can be aggregated as Virtual Machines as well as other technologies such as Linux containers. Likewise, a service is a collection of instances that serve a particular purpose or application. Examples of services, in this context, are IMS (IP Multimedia Subsystem) network elements, CDN (Content distribution Networks) nodes, N-tier services, and so on. Accordingly, reference herein to a VM is not intended to be limiting, and can be any suitable software container in which a service or application is executed over physical hardware, and which may be spawned, deleted and scaled and so on such as the referenced Linux containers for example.

As is typical, cloud resources are managed, provisioned and monitored by the use of Application Programmer Interfaces (APIs) that allow remote access to various features of the resources. In an example, a monitoring API 203 collects resource metrics (e.g., CPU, memory, storage, network utilization), which are transmitted to (207) and used during the creation of dependency structures using, for example, an orchestration server 204 operable to process data as outlined with reference to figure 1. Once the dependency structures are created, a scaling solution is communicated (209) to the orchestration API 205 in order to instruct scaling of the instances 202 needed to handle an incoming workload. Accordingly, the amount of resources required by a service is determined and then scaling operations are performed.

That is, resource metrics (e.g., CPU, memory, storage, network utilization, and others if necessary) of instances (VMs or Linux Containers) are used to compute a time-lagged partial correlation graph of resources and to scale the resources following the graph. The method is generally speaking composed of five stages, where the output of one stage can be input to the next, and which can be executed periodically.

According to an example, metrics of each VM 202 are recorded at some predetermined frequency, such as every 5 seconds for example, and are aggregated as vector sums. For example, for both horizontal and vertical scaling, if a VM has multiple virtual CPUs, all virtual CPU utilizations are aggregated as if there was only a single virtual CPU with the VM; for horizontal scaling, the virtual CPU utilizations of the multiple VMs that run the same service components are further aggregated as if there was only a single VM that runs these service components. This aggregation procedure also applies to other metrics, such as those noted above (for example, memory, storage and/or network utilization).

Resources to be scaled are identified by computing time-lagged partial correlation graphs. That is, to determine which resources need to be scaled, the time-lagged partial correlation of available metrics is determined using cross correlation and partial correlation.

5

More specifically, the cross correlation $\rho_{XY}(\tau)$ at time lag

τ ($\tau \in Z, |\tau| < T$) is

$$\rho_{XY}(\tau) = \frac{\sum_{t=1}^{T}(x_t - \bar{x})(y_{t+\tau} - \bar{y})}{\sqrt{\sum_{t=1}^{T}(x_t - \bar{x})^2}\sqrt{\sum_{t=1}^{T}(y_t - \bar{y})^2}}$$

10

where

$$\bar{x} = \frac{1}{T}\sum_{t=1}^{T} x_t \quad \text{and} \quad \bar{y} = \frac{1}{T}\sum_{t=1}^{T} y_t$$

15 for two time series data $X = \{x_1, x_2, ..., x_T\}$ and $Y = \{y_1, y_2, ..., y_T\}$.

In an example, X and Y can be the aggregated CPU utilizations of two VMs, or other metrics such as memory utilizations for example. Across all time lags, the highest cross correlation $\rho_{XY}(\tau_{XY})$ where:

20

$$\tau_{XY} = \arg\max_{\tau \in Z, |\tau| < T} \{\rho_{XY}(\tau)\}$$

is of the interest.

25 However, calculating $\rho_{XY}(\tau_{XY})$ straightforwardly for all pairs of resources results in that many pairs may have similar correlation values, which would indicate that those pairs are correlated and need to be scaled. As would be understood by one skilled in the art, these correlations are because the utilizations of the resources of a service, principally, are all driven by the workload of the service.

30

Thus, according to an example, in order to detect the correlations between resources at a finer level by excluding the influence of the workload, the time-lagged partial correlation is defined as:

$$\rho_{XY\backslash Z}(\tau) = \frac{\rho_{XY}(\tau) - \rho_{XZ}(\tau)\rho_{ZY}(0)}{\sqrt{1-\rho_{XZ}(\tau)^2}\sqrt{1-\rho_{ZY}(0)^2}}$$

where $\rho_{XY}(\tau)$ is the cross correlation mentioned as above.

There are three time series involved: X and Y explained as above, as well as $Z = \{z_1, z_2, ..., z_T\}$, where the former two can represent the CPU utilizations and the latter one can be the workload.

Note that this concept extends the standard definition of partial correlation which does not capture any time lag information. Analogously, $\rho_{XY\backslash Z}(\tau_{XY\backslash Z})$ where $\tau_{XY\backslash Z} = \arg\max_{\tau \in Z, |\tau| < T} \{\rho_{XY\backslash Z}(\tau)\}$ is of the interest.

Calculating $\rho_{XY\backslash Z}(\tau_{XY\backslash Z})$ for all pairs of resources respectively can result in a clear separation: some pairs have higher correlation values while others have lower ones, overcoming the disadvantage of cross correlation.

Figures 3a-c are schematic representations relating to the calculation of a partial correlation for a time lag of 2 between two CPU utilizations X and Y excluding the influence of a workload Z. Each time series contains 4 samples (which may be the differences between the original samples and their corresponding averages) each of which is represented by a visual symbol as shown. Samples that are aligned at the same column need to be multiplied. Figures 3a-c visualize the calculation of by how much X is influenced by Y 2 time units ago (i.e., $\rho_{XY}(2)$), by how much X is influenced by Z 2 time units ago (i.e., $\rho_{XZ}(2)$), and by how much Y is influenced by Z with no time lag (i.e., $\rho_{ZY}(0)$). Note that, to exclude the influence of Z from both X and Y, Z needs to be shifted by 2 time units which also makes it have no time lag with Y.

A time-lagged partial correlation graph is produced after calculating the time-lagged partial correlation $\rho_{XY\setminus Z}(\tau_{XY\setminus Z})$ for all pairs of resources. More specifically, according to an example, a graph in which every pair of vertices are connected is produced: each vertex or node represents a resource of a VM (in case of vertical scaling) or a resource of a set of VMs that executes the same service components (in case of horizontal scaling), and each edge represents the corresponding time-lagged partial correlation. A threshold can be used to filter out those edges that have weights below the threshold. A suitable threshold value is 0.8. The correlation value calculated is in the range between 0 and 1, both inclusive. So in fact any value in this range can be used as the threshold.

Nodes whose incident edges have been filtered out can be removed in order to provide a directed graph comprising a subset of nodes representing resources to be scaled. That is, following the removal of nodes with no incident edges, the vertices that remain are the resources that need to be scaled; the edges that remain represent co-scaling, i.e., if one resource is scaled then the other connected resource also needs to be scaled. The time lag is $\tau_{XY\setminus Z}$, i.e., scaling one resource only after a time lag of $\tau_{XY\setminus Z}$ when the other is scaled. This graph is a directed graph.

Determining when to scale the resources is determined, in an example, by serializing the time lags via shortest path and vector norm calculations. That is, the time-lagged partial correlation graph has the time lag between two resources, which is the value that indicates when to scale one resource relative to the other. However, it is more desirable to know when to scale every other resource relative to one selected resource. Accordingly, the time lags in a given time-lagged partial correlation graph are serialized as noted below. Note that the correlation values are not important and thus removed; the edge weights in the graph may therefore simply be referred to as the time lags.

For every edge (i.e., the original edge) in a time-lagged partial correlation graph, the companion edge which is between the same pair of vertices but which has the opposite direction of original edge and uses the negation of the weight of the original edge as its weight is added. This aligns with intuition: for example, if resource 1 needs to be scaled 2 minutes after resource 2, this is equivalent to saying that resource 2 needs to be scaled 2 minutes earlier than resource 1. The original edge and its companion capture these two situations respectively. In this connection, figure 4 is a schematic representation of a partial correlation graph according to an example. Solid circles and

arrows represent resources and correlations respectively and the numbers are resource IDs and time lags. The companion edges, with corresponding weights can be seen.

According to an example, each node 401 is used as a source and the single-source shortest path towards every other node in the graph from the source is calculated. The lengths of these paths are recorded in a vector. In the case where the graph has no negative cycles, a standard Bellman-Ford process can be used, which is a known algorithm used to compute shortest paths from a single source vertex to all of the other vertices in a weighted digraph. In the case where there exists negative cycles (i.e., no shortest path exists as one can always loop the negative cycle to reduce the path length), the loop-free shortest path becomes the target instead, which leads to a NP-hard problem, and heuristics can be used to find an approximate shortest path. That is, in the case where there are negative cycles, there is no shortest path as one can always loop the negative cycle to reduce the path length forever. Accordingly, one or some edges in this negative cycle can be removed (so that the cycle is broken and there are no cycles any more). Afterwards, the shortest path can be determined as before.

The rationale is that, firstly, the time lags, no matter positive or negative, should be transitive, e.g., if resource 1 needs to be scaled 2 minutes after resource 2 and resource 2 needs to be scaled 1 minute before resource 4, then it should be case that resource 1 needs to be scaled 1 minute after resource 4, which is why the path length can represent the accumulated time lag. Secondly, the shortest path is of interest based on the assumption that the correlation is more likely to occur in a short period of time rather than in a longer one.

Figures 5a and 5b are schematic representations of two path length vectors according to an example using resources (VMs) 1 and 5 as the sources, where the ith element in the vector records the length of the shortest path from the source to resource i. For example, in figure 5a a path length vector using node 1 as the source is shown, with path lengths to other resources in the graph. Similarly, figure 5b shows a path length vector using node 5 as the source, with path lengths to other resources in the graph.

For all path length vectors, one vector is selected and used as the result, from which the time at which resources should be scaled can be determined. The selected path length vector has the time lag of every vertex in the graph relative to the source vertex

of this vector, i.e., all time lags are serialized and it is now known at which time lag to scale each resource in the service. However, different path length vectors using different sources may give different information about when to scale a resource, and therefore one of the path vectors needs to be selected as the result.

According to an example, a path length vector is selected using a norm function. The norm can be the 1-norm or the maximum norm for example, where, for a vector $(x_1, x_2, ..., x_N)$, the 1-norm is defined as $l_1 = \sum_{i=1}^{N} |x_i|$ and the maximum norm is defined as $l_{\max} = \max_i \{|x_i|\}$.

The intuition behind selecting the vector with the minimum norm value is analogous to what has been mentioned above: the correlation is more likely to occur within a short period of time. Figures 5a and 5b show two vectors, where one may prefer the first to the second, as the first has both a smaller 1-norm value ($l_1 = 8$) and a smaller maximum norm value ($l_{\max} = 4$). Note that after selecting the vector, one may prefer to make every element of the vector non-negative by adding the minimum value out of the elements to every element, e.g., $\vec{v_1} = (0, -2, 0, -1, -4, 1)$ will be changed to $\vec{v_1} = (4, 2, 4, 3, 0, 5)$ so that it is known that only resource 5 needs to be monitored, and when it needs to be scaled, resources 1 and 3 need to be scaled 4 minutes later, resource 2 needs to be scaled 2 minutes later, and so on.

According to an example, it may be necessary to resolve conflicts of the decisions made by multiple metrics in the case of horizontal scaling. That is, in horizontal scaling, a VM is the smallest block to be manipulated and adding/removing resources is implemented by adding/removing VMs. If multiple metrics (e.g., CPU, memory, network, storage) are used to make scaling decisions, there might be a conflict: for example, in a VM, its CPU resource may need to be increased after 1 minute but its memory resource may need to be increased after 2 minutes. Therefore, at what time exactly should the increase of CPU or memory be implemented by adding one more VM that runs the same service components. According to an example a VM is added 1 minute later so that both the additional CPU and memory are ready when they are needed, although the additional memory is ready 1 minute before actually required.

Generally, there are two types of conflicts that need to be resolved:

Type 1 Conflict: To resolve the first type of conflict where multiple resources require adding (or removing) VMs at different times, the present method ensures that every VM is provided at the earliest time as required by any of its resources and thus all resources are scaled in time by adopting the following approach. As noted above multiple path length vectors are produced, each of which corresponds to a metric. Recall that a path length vector is a set of time values relative to the absolute time to scale the resource that has the time lag 0.

Let $\overrightarrow{v_m}$ denote the path length vector corresponding to the metric $m \in \mathrm{M}$, $\overrightarrow{v_{m\,i}}$ denote the time lag for resource i (or more precisely, the VM that contains resource i) in $\overrightarrow{v_m}$, and $t_{m,base}$, (as will be further explained below) denote the absolute time of scaling the resource that has the time lag 0. Given all of these, the absolute time to manipulate the VM that runs the same service components as the VM i is calculated as $t_i = \min_{m \in \mathrm{M}} \{t_{m,base} + \overrightarrow{v_{m\,i}}\}$. Note that a path length vector may need to add some values of $+\infty$ in order to match the length of other path length vectors.

An example is as follows. Suppose the path length vector for CPU is $\overrightarrow{v_{cpu}} = (4,2,4,3,0,5)$ and that for memory is $\overrightarrow{v_{mem}} = (0,4,1)$, and suppose $t_{cpu,base} = 2$ and $t_{mem,base} = 1$. The first thing is changing $\overrightarrow{v_{mem}}$ to $\overrightarrow{v_{mem}} = (0,4,1,+\infty,+\infty,+\infty)$. Afterwards, applying the t_i formula above for every i gives a vector of $(1,4,2,5,2,7)$, which represents the absolute times to add the corresponding VMs.

This procedure is visualized in figure 6, which is a schematic representation of conflict resolution according to an example, where for ease of presentation, $t_{m,base}$ is already merged into $\overrightarrow{v_m}$.

As can be seen in figure 6, every VM is provided at the earliest time as required by any of its resources, and so, for the sake of argument, as resource 1 is scaled at time 1 for memory, the corresponding CPU scaling for this resource at time 6 is removed, and so on.

To resolve the second type of conflict, where some resources require adding a VM and others require removing this VM, the present method takes the following approach. For VM i, all of the absolute time values are collected in a set or a vector as $\{t_{m,base} + \overrightarrow{v_{m\,i}}\}_{m\in \mathrm{M}}$, where each element has also been associated with the property of either "adding the VM" or "removing the VM". The approach sorts the vector elements in ascending order. For every group of consecutive elements with the same property, we only keep one element and delete the other elements in the following manner. If a group of consecutive elements have the property of adding a VM, we keep the first element and delete the others in the same group. If a group of consecutive elements have the property of removing a VM, we keep the last element and delete the others in the same group. This approach guarantees that a VM is added at the earliest time when it is needed by any of its resources and removed at the latest time when it is not needed by any of its resources..

Figure 7 is a schematic representation of conflict resolution according to an example. Suppose four metrics are used: CPU, memory, storage and network utilization. The former two are the same as in figure 6 and the latter two are captured by $\overrightarrow{v_{stor}} = (8, +\infty, +\infty, +\infty, 3, +\infty)$ and $\overrightarrow{v_{net}} = (3, +\infty, +\infty, +\infty, 6, +\infty)$.

We also assume that CPU and storage require adding VMs and that memory and network require removing VMs. All four vectors are drawn in the top part of figure 7, and the approach described above will produce that shown the bottom subfigure. This means, for example, adding one more VM 5 as required by CPU at time 2, and removing it as required by network at time 6; although storage also requires adding VM 5 at time 3, $\overrightarrow{v_{stor\,5}}$ is eliminated because VM 5 is already added at time 2.

The final part of the process operates the service by adding or removing VMs in the case of horizontal scaling or increasing or decreasing the amount of resources in the case of vertical scaling, at the time lags specified by the result after conflict resolution as described above. In a real-world cloud system, one approach to store and process this result can be by using a set of key-value pairs where each entry is in the format of {RESOURCE_ID, TIME_LAG} for example.

Only a minimum amount of information is needed as input. The first is the absolute time of when to scale the one resource that is being monitored, denoted as

$t_{m,base}$ previously. In order to get this, existing methods can be used, such as rule-based ones, e.g., the absolute time is when the average utilization of existing resources is above 90% for 5 minutes for example. Having this information, it is then possible to tell the absolute times of when to scale every other resource/VM, as shown above.

The second issue is the number of VMs or the amount of resources that need to be added or removed per scaling operation. This information can be pre-specified. Note that, given these two inputs, the focus is the following: firstly, when one resource needs to be scaled at some time, at what time relative to it the "other related" resources also need to be scaled and which (of multiple VMs in system for example) these "other related" resources are. Secondly, only one resource needs to be monitored (and which this one resource is).

The necessary mechanisms to ensure concurrency and handle the scaling delays are therefore provided. The scaling operations of any two resources with the same time lag can always be executed concurrently. Furthermore, the time duration for executing a scaling operation may also need to be taken into account so that one may desire to start to scale the resource at some time before the time that is specified by the time lag so that the scaling operation can be completed on time. The method described is proactive compared with existing reactive ones, in the sense that the resources can be ready before the time that is specified by the time lag, because the time lag is calculated in advance.

Thus, a general, efficient, and proactive method that only requires a minimum and almost zero manual intervention in order to scale a wide range of cloud services, either horizontally or vertically is provided. It is fast, scalable, and consumes negligible resource for execution. It involves only local VM metrics which are out of the data path of the service. Second, it requires extremely moderate manual touch or pre-specified settings, and afterwards calculates everything needed automatically. Last, it is proactive, predictive, and can execute online to calculate when future scaling operations need to occur so that these operations can start in advance and complete in time.

Figure 8 is a schematic representation of an apparatus 800 such as orchestration engine or server 204. The apparatus 800 includes processor 810, data storage 811,

and I/O interface 830. The processor 810 controls the operation of the apparatus 800. The processor 810 cooperates with the data storage 811.

The data storage 811 stores appropriate ones of programs 820 executable by the processor 810. Data storage 811 may also optionally store program data such as threshold values or the like as appropriate. The processor-executable programs 820 may include an input gathering program 821, aggregated metric generation program 823, directed graph generation program 825, edge/node removal Program 827, and conflict resolution program 829. Other programs may be provided to perform other processes as appropriate and as described above.

Processor 810 cooperates with processor-executable programs 820. The I/O interface 830 cooperates with processor 810 and an I/O interface program to support communications over communication channels 207, 209 of FIG. 1 as appropriate and as described above.

In an example, the information gathering program 823 can perform the step 101 of FIG. 1 as appropriate and as described above; the aggregated metric generation program 825 can generate respective sets of aggregated metrics for respective ones of multiple virtual machines as appropriate and as described above; the directed graph generation program 825 can generate a directed graph as appropriate and as described above; the edge/node removal program 827 can remove edges of the directed graph whose weight falls below a threshold measure and remove nodes whose incident edges have been removed; and the Conflict resolution program 829 can resolve conflicts in the case of horizontal scaling for the application or service component by adding resources of a virtual machine at the earliest time required by any of the resources or by removing resources of a virtual machine at the latest time required by any of the resources, or by adding or removing a virtual machine instance for a resource. A scaling program 831 can be provided to scale the weight of remaining edges of the directed graph relative to a selected node of the graph.

Programs 820 can include a program 833 to generate a resource request for the service or application component which is operable to provide an instruction to add or remove one or more virtual machine instances or increase or decrease the amount of resources available to one or more virtual machine instances at times selected according to the time lag values specified in a time-lag vector.

In some embodiments, the processor 810 may include resources such as processors / CPU cores, the I/O interface 830 may include any suitable network interfaces, or the data storage 811 may include memory or storage devices. Moreover the apparatus 800 may be any suitable physical hardware configuration such as: one or more server(s), blades consisting of components such as processor, memory, network interfaces or storage devices. In some of these embodiments, the apparatus 800 may include cloud network resources that are remote from each other.

In some embodiments, the apparatus 800 may be virtual machine. In some of these embodiments, the virtual machine may include components from different machines or be geographically dispersed. For example, the data storage 811 and the processor 810 may be in two different physical machines.

When processor-executable programs 820 are implemented on a processor 810, the program code segments combine with the processor to provide a unique device that operates analogously to specific logic circuits.

Although depicted and described herein with respect to embodiments in which, for example, programs and logic are stored within the data storage and the memory is communicatively connected to the processor, it should be appreciated that such information may be stored in any other suitable manner (e.g., using any suitable number of memories, storages or databases); using any suitable arrangement of memories, storages or databases communicatively connected to any suitable arrangement of devices; storing information in any suitable combination of memory(s), storage(s) or internal or external database(s); or using any suitable number of accessible external memories, storages or databases. As such, the term data storage referred to herein is meant to encompass all suitable combinations of memory(s), storage(s), and database(s).

The present inventions can be embodied in other specific apparatus and/or methods. The described embodiments are to be considered in all respects as illustrative and not restrictive. In particular, the scope of the invention is indicated by the appended claims rather than by the description and figures herein. All changes that come within the meaning and range of equivalency of the claims are to be embraced within their scope.

CLAIMS

1. A method of determining a set of virtual machine instances in a virtualized system to be scaled in response to a variation in workload of a service or application component, the method comprising:

generating respective sets of aggregated metrics for respective ones of multiple virtual machines;

generating a directed graph in which nodes representing a resource of one or more of the multiple virtual machines are linked using one or more edges, the weight of each of the edges representing a time-lagged partial correlation value and a time lag value representing a measure of when to scale one resource relative to another;

determining the set of virtual machine instances to be scaled by removing edges of the directed graph whose weight falls below a threshold measure and removing nodes whose incident edges have been removed; and

scaling the weight of remaining edges of the directed graph relative to a selected node of the graph.

2. A method as claimed in claim 1, wherein generating an aggregated metric includes:

determining, at periodic intervals, utilisation of one or more resources by respective ones of the multiple virtual machines; and

generating an aggregated vector sum of the resource utilisations for respective ones of the multiple virtual machines.

3. A method as claimed in claim 1 or 2, wherein a node of the directed graph represents a resource of a virtual machine or an aggregated resource measure of a set of virtual machines that execute the same service or application component.

4. A method as claimed in any preceding claim, further comprising:

for respective ones of the nodes of the directed graph, calculating a single-source shortest path towards every node in the directed graph and generating a vector of the calculated lengths, each vector comprising components representing the time lag of each node in the graph relative to a source node for the vector in question, whereby to provide a serialised set of time lags for the directed graph.

5. A method as claimed in claim 4, further comprising:

selecting a vector from the set with a minimum norm value;

in the event that the selected vector includes negative valued components, adding the absolute value of the minimum valued component to components of the selected vector; and

determining a resource to monitor based on the component of the selected vector with the minimum value.

6. A method as claimed in claim 4 or 5, further comprising:

resolving conflicts in the case of horizontal scaling for the application or service component by adding resources of a virtual machine at the earliest time required by any of the resources or removing resources of a virtual machine at the latest time required by any of the resources, or by adding or removing a virtual machine instance for a resource.

7. A system including multiple virtual machines in program execution on physical computing hardware supported by a hypervisor, the multiple virtual machines being operable to execute a service or application component in a cloud infrastructure, the system including:

an orchestration engine to:

generate respective sets of aggregated metrics for respective ones of multiple virtual machines;

generate a directed graph in which nodes representing a resource of one or more of the multiple virtual machines are linked using one or more edges, the weight of each of the edges representing a time-lagged partial correlation value and a time lag value representing a measure of when to scale one resource relative to another;

determine the set of virtual machine instances to be scaled by removing edges of the directed graph whose weight falls below a threshold measure and removing nodes whose incident edges have been removed; and

scale the weight of remaining edges of the directed graph relative to a selected node of the graph.

8. A system as claimed in claim 7, wherein the orchestration engine is operable to:

determine, at periodic intervals, utilisation of one or more resources by respective ones of the multiple virtual machines; and

generate an aggregated vector sum of the resource utilisations for respective ones of the multiple virtual machines.

9. A system as claimed in claim 7 or 8, wherein a node of the directed graph represents a resource of a virtual machine or an aggregated resource measure of a set of virtual machines that execute the same service or application component.

10. A system as claimed in any claims 7 to 9, wherein the orchestration engine is operable to:

for respective ones of the nodes of the directed graph, calculate a single-source shortest path towards every node in the directed graph and generate a time-lag vector of the calculated lengths, each vector comprising components representing the time lag of each node in the graph relative to a source node for the vector in question, whereby to provide a serialised set of time lags for the directed graph.

11. A system as claimed in claim 10, wherein the orchestration engine is operable to:

select a vector from the set with a minimum norm value;

in the event that the selected vector includes negative valued components, add the absolute value of the minimum valued component to components of the selected vector; and

determine a resource to monitor based on the component of the selected vector with the minimum value.

12. A system as claimed in claim 10 or 11, wherein the orchestration engine is operable to:

resolve conflicts in the case of horizontal scaling for the application or service component by adding resources of a virtual machine at the earliest time required by any of the resources or removing resources of a virtual machine at the latest time required by any of the resources, or by adding or removing a virtual machine instance for a resource.

13. A system as claimed in any of claims 7 to 12, wherein the orchestration engine is operable to:

generate a resource request for the service or application component providing an instruction to add or remove one or more virtual machine instances or increase or decrease the amount of resources available to one or more virtual machine instances at times selected according to the time lag values specified in a time-lag vector.

14. A computer program product, comprising a computer usable medium having computer readable program code embodied therein, said computer readable program code adapted to be executed to implement a method of determining a set of virtual machine instances in a virtualized system to be scaled in response to a variation in workload of a service or application component as claimed in any of claims 1 to 6.

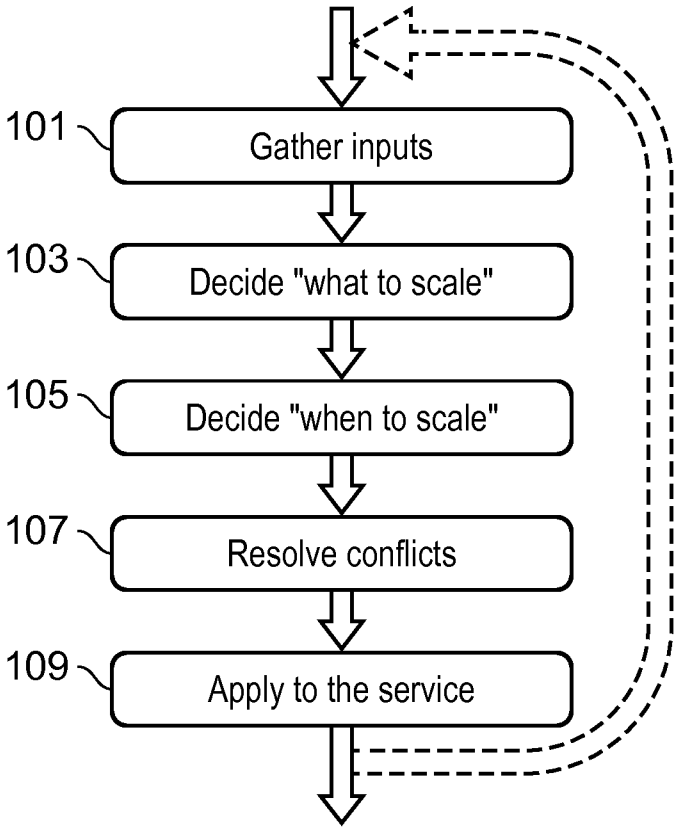

101
Gather inputs
103
Decide "what to scale"
105
Decide "when to scale"
107
Resolve conflicts
109
Apply to the service

FIG. 1

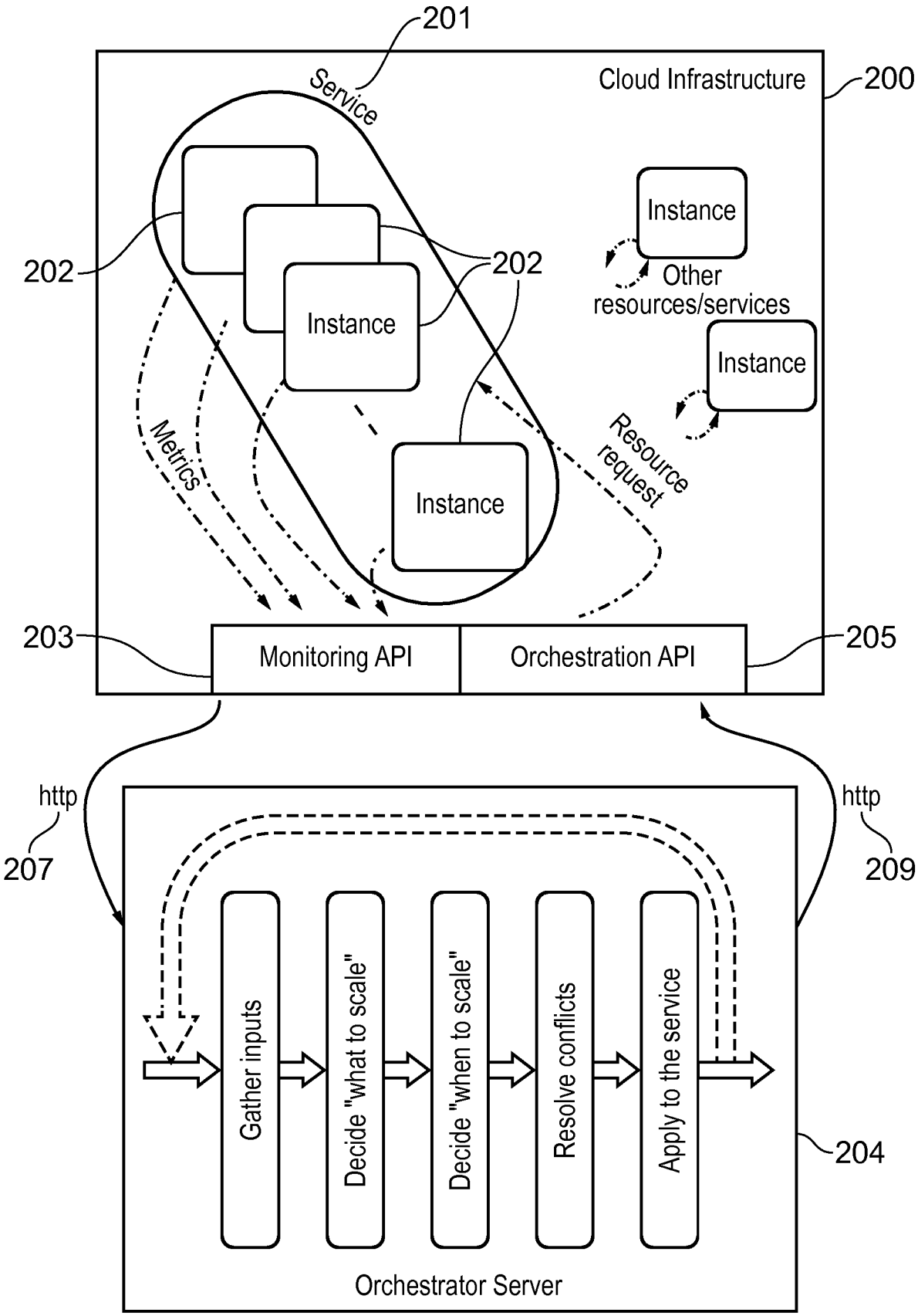

201
Service
Cloud Infrastructure
200
Instance
202
202
Other
resources/services
Instance
Instance
Metrics
Resource
request
Instance
203
Monitoring API
Orchestration API
205
http
207
http
209
Gather inputs
Decide "what to scale"
Decide "when to scale"
Resolve conflicts
Apply to the service
204
Orchestrator Server

FIG. 2

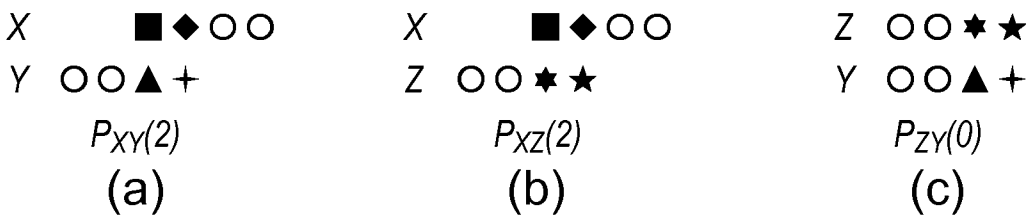


FIG. 3

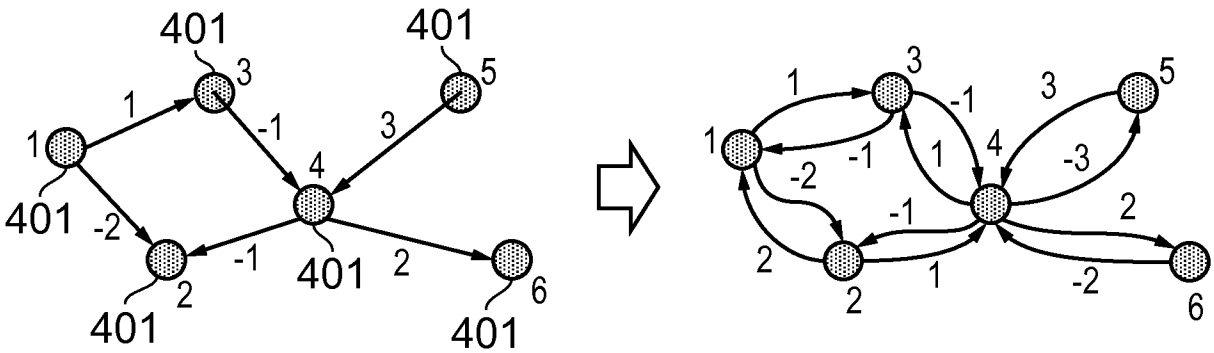
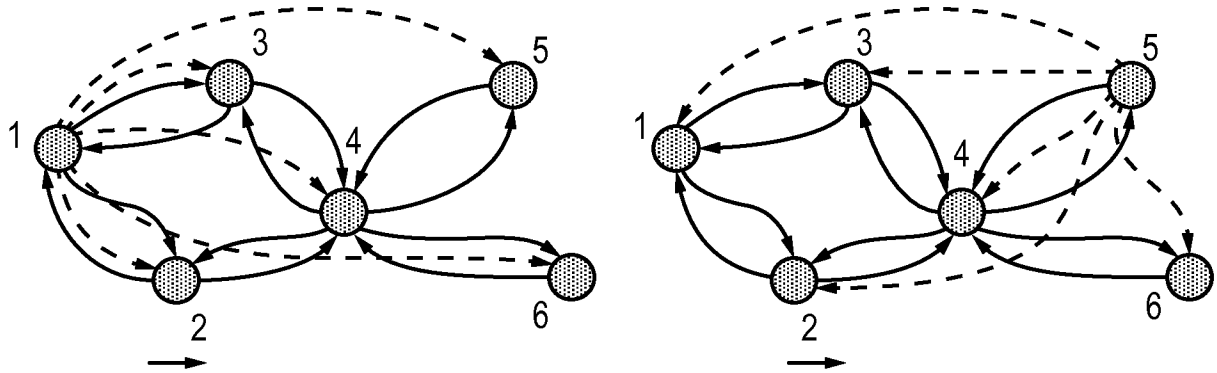


FIG. 4



$V_1 = (0, -2, 0, -1, -4, 1)$ (a)

$V_5 = (3, 1, 4, 3, 0, 5)$ (b)

FIG. 5

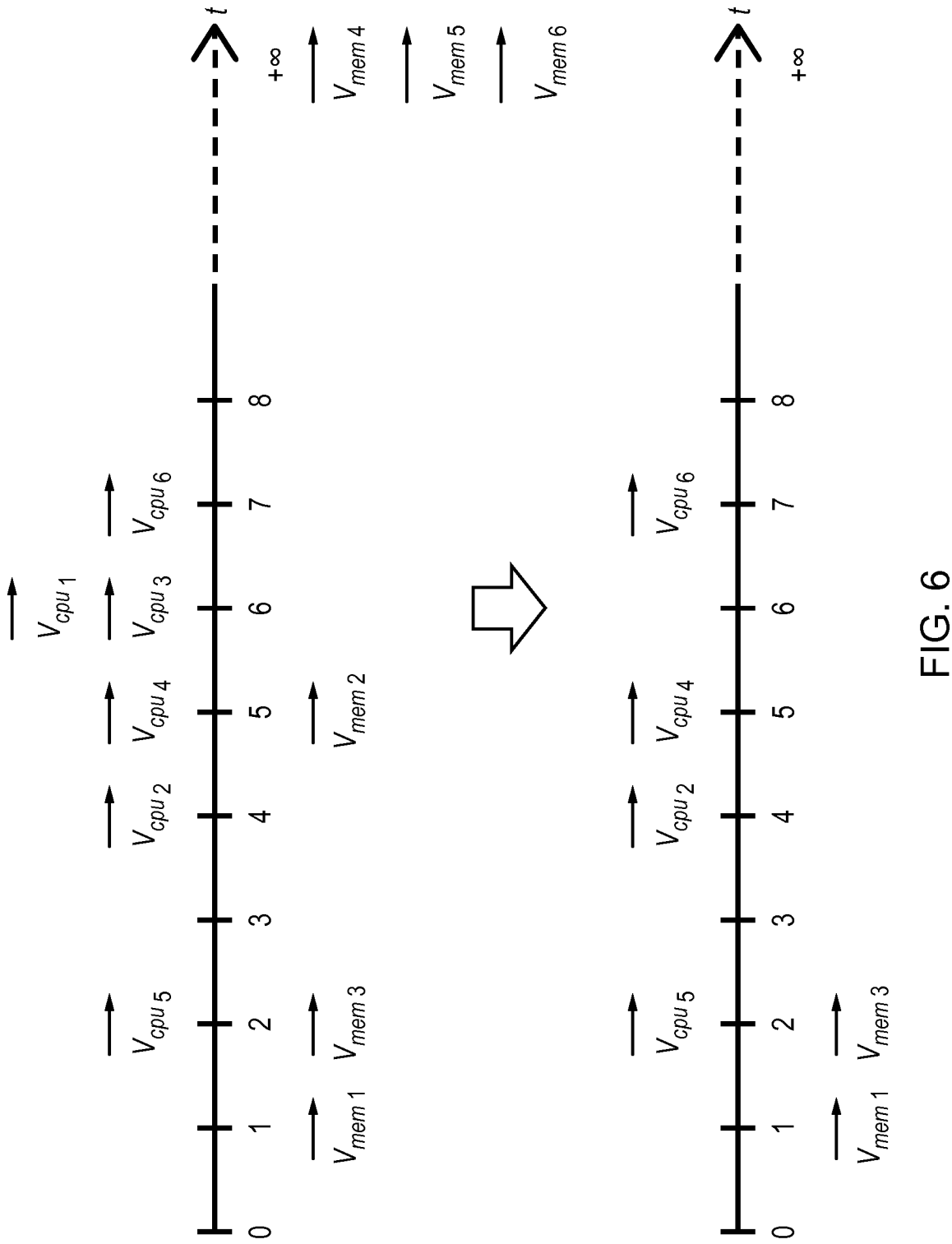

t
+∞
0
1
2
3
4
5
6
7
8
$V_{cpu\,1}$
$V_{cpu\,2}$
$V_{cpu\,3}$
$V_{cpu\,4}$
$V_{cpu\,5}$
$V_{cpu\,6}$
$V_{mem\,1}$
$V_{mem\,2}$
$V_{mem\,3}$
$V_{mem\,4}$
$V_{mem\,5}$
$V_{mem\,6}$

FIG. 6

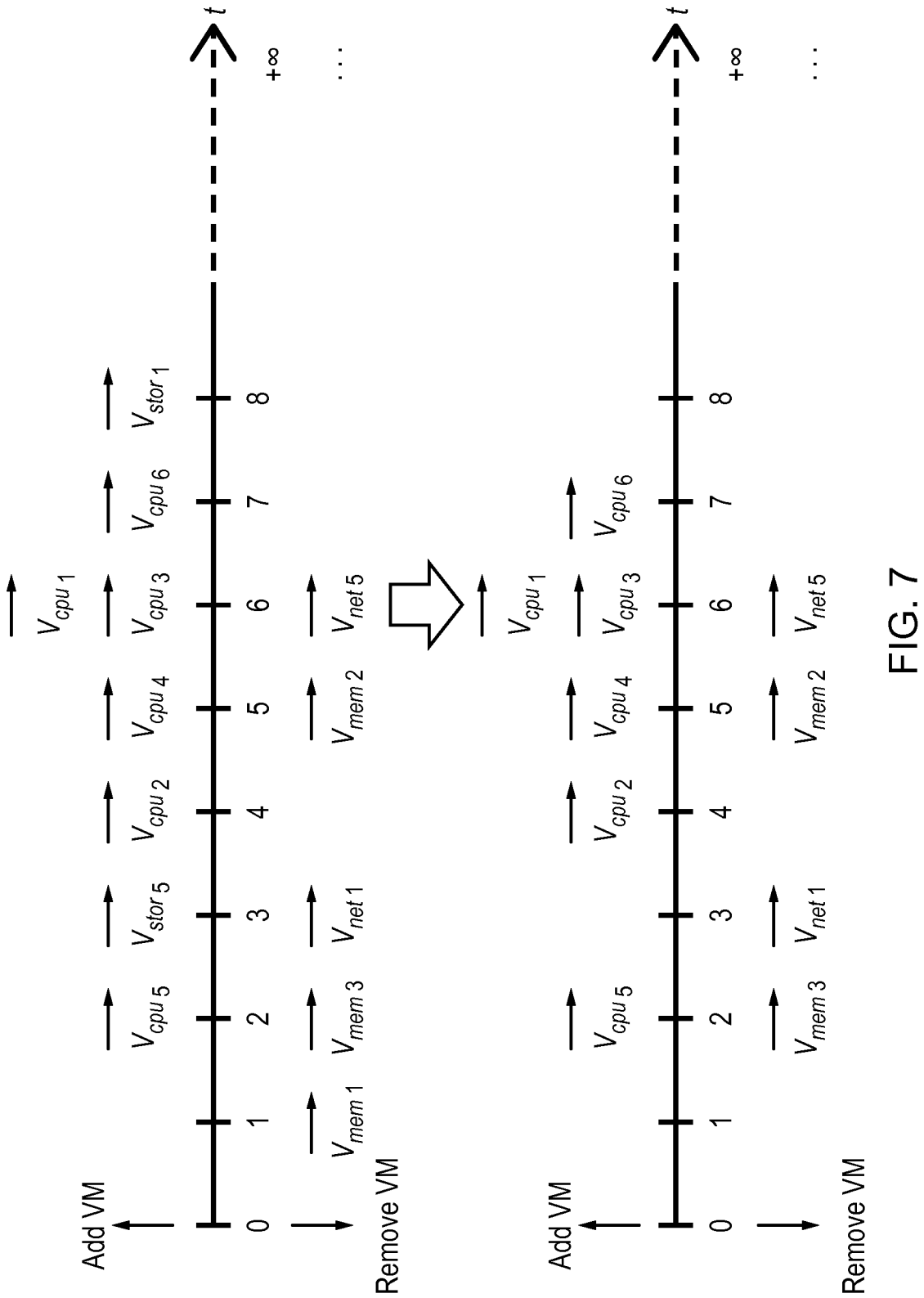

t
+∞
...
Add VM
Remove VM
0
1
2
3
4
5
6
7
8
$V_{cpu\,1}$
$V_{cpu\,5}$
$V_{stor\,5}$
$V_{cpu\,2}$
$V_{cpu\,4}$
$V_{cpu\,3}$
$V_{cpu\,6}$
$V_{stor\,1}$
$V_{mem\,1}$
$V_{mem\,3}$
$V_{net\,1}$
$V_{mem\,2}$
$V_{net\,5}$

FIG. 7

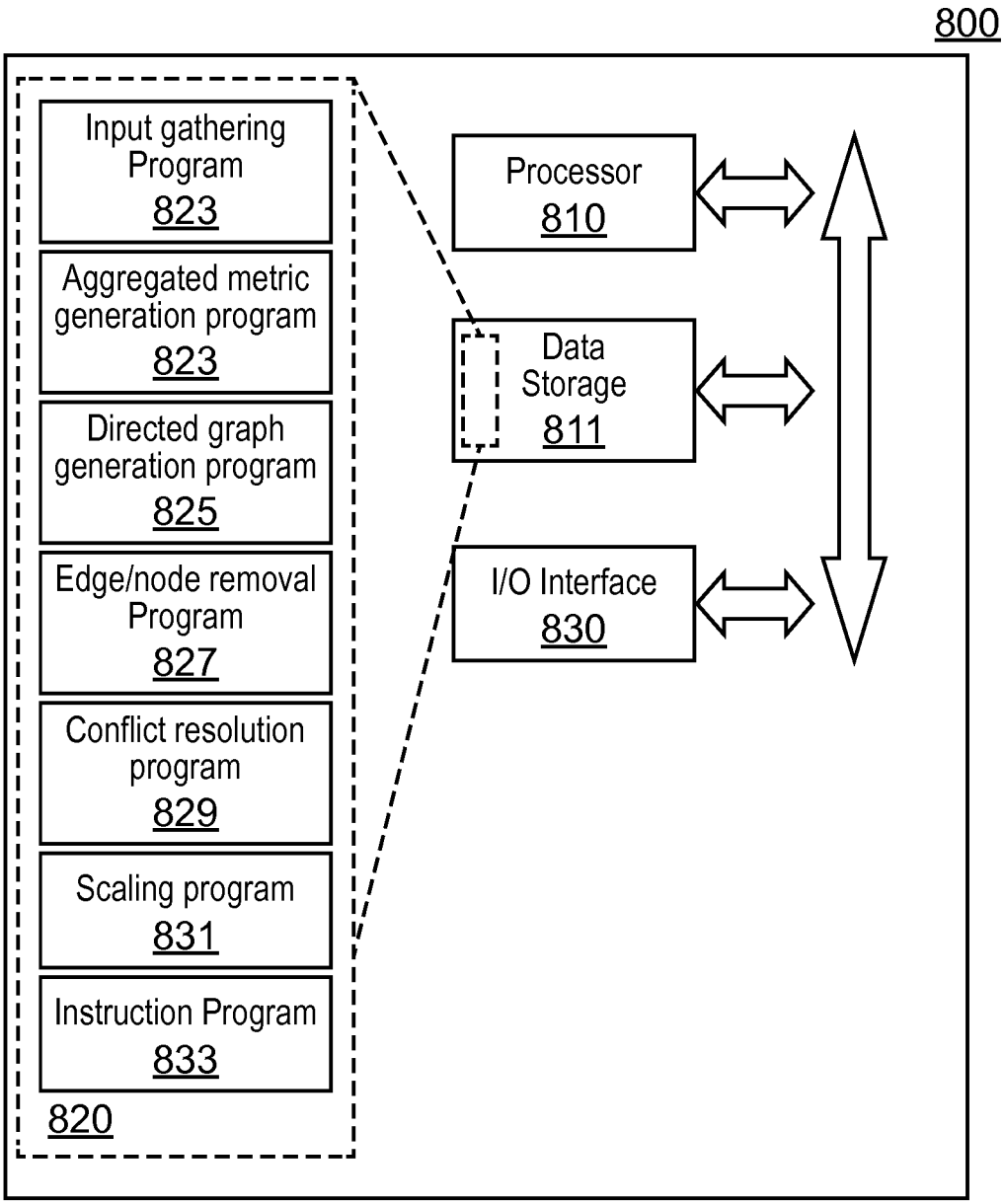

800
Input gathering Program 823
Aggregated metric generation program 823
Directed graph generation program 825
Edge/node removal Program 827
Conflict resolution program 829
Scaling program 831
Instruction Program 833
820
Processor 810
Data Storage 811
I/O Interface 830

FIG. 8

INTERNATIONAL SEARCH REPORT

International application No
PCT/EP2017/057185

A. CLASSIFICATION OF SUBJECT MATTER
INV. G06F9/50
ADD.

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

B. FIELDS SEARCHED

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

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

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

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	US 2014/143773 A1 (CIANO GIUSEPPE [IT] ET AL) 22 May 2014 (2014-05-22) paragraphs [0030], [0035] -----	1-14

☐ Further documents are listed in the continuation of Box C.

☒ See patent family annex.

* Special categories of cited documents :

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

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

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

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

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

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

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

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

"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
10 April 2017	19/04/2017

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, Fax: (+31-70) 340-3016	Mühlenbrock, Martin

1

INTERNATIONAL SEARCH REPORT

Information on patent family members

International application No
PCT/EP2017/057185

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
US 2014143773 A1	22-05-2014	GB 2508161 A	28-05-2014
		US 2014143773 A1	22-05-2014