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

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
5 March 2015 (05.03.2015)

WIPO | PCT

(10) International Publication Number
WO 2015/028538 A1

(51) International Patent Classification:
G06F 9/445 (2006.01)

(21) International Application Number:
PCT/EP2014/068228

(22) International Filing Date:
28 August 2014 (28.08.2014)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
13182601.8 2 September 2013 (02.09.2013) EP

(71) Applicant: **AGFA HEALTHCARE** [BE/BE]; IP Department 3802, Septestraat 27, B-Mortsel 2640 (BE).

(72) Inventor: **BOEL, Nikolas**; c/o AGFA HEALTHCARE, IP Department 3802, Septestraat 27, B-2640 Mortsel (BE).

(74) Agent: **VERBRUGGHE, Anne-Marie**; IP Department 3622, Septestraat 27, B-2640 Mortsel (BE).

(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, 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, KN, KP, KR, KZ, LA, LC, LK, LR, LS, LT, 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, 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))*

(54) Title: A METHOD AND SYSTEM FOR UPGRADING SOFTWARE

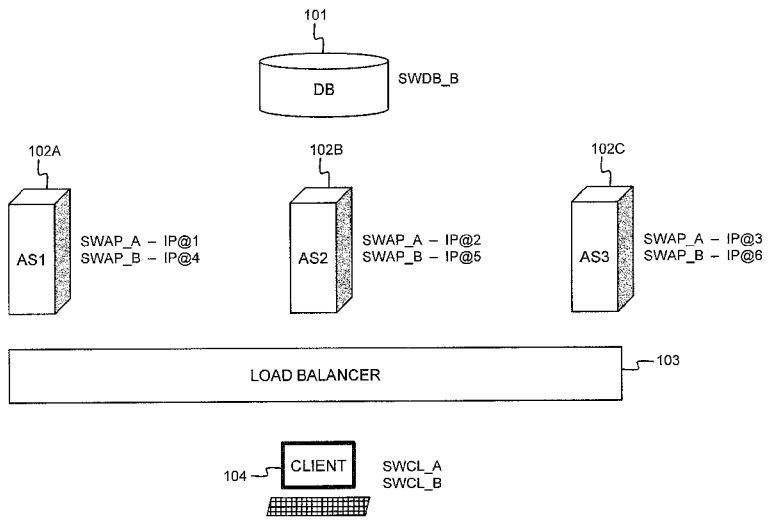

Fig. 4

(57) Abstract: In order to upgrade software having a client component (SWCL), application server component (SWAP) and database component (SWDB) from an old version (SWCL_A, SWAP_A, SWDB_A) to a new version (SWCL_B, SWAP_B, SWDB_B), the new version (SWDB_B) of the database component (SWDB) is installed on the database servers (101 A, 101 B; 101) without interrupting client service in a first step. In a second step, the new version (SWAP_B) of the application server component (SWAP) is installed on the application servers (102A, 102B, 102C) without interrupting client service. In a third step, the new version (SWAP_B) of the application server component (SWAP) is registered with the load balancer (103) using an IP address (IP@4, IP@5, IP@6) that is different from the IP address (IP@1, IP@2, IP@3) used to register the old version (SWAP_A) of the application server component (SWAP). In a fourth step, the new version (SWCL_B) of the client component (SWCL) is installed with respective ones of the clients (104A, 104B, 104C; 104) when

respective users of these clients (104A, 104B, 104C; 104) desire

WO 2015/028538 A1

A METHOD AND SYSTEM FOR UPGRADING SOFTWARE

Field of the Invention

[01] The present invention generally relates to upgrading software. The invention in particular relates to upgrading client-server application software that typically contains a database component, an application server component, and a client component. The invention aims at upgrading such software, in particular in critical domains like healthcare, with limited or no downtime for the clients.

Background of the Invention

[02] In the past, software upgrades of client-server applications were executed at a slow pace, typically once a year. These software upgrades however involved downtime of the system as a result of which the system was temporarily unavailable for users.

[03] Technologies like scrum have reduced development cycles of new versions of software. New features of a client-server software application are introduced within the next three months. As a result, software upgrades come at a faster rhythm and downtime of the system for customers increases substantially.

[04] Service upgrades and hot bug fixes further tend to trigger additional upgrades of client-server software applications, as a consequence of which the customer experiences even more frequent downtimes of the system.

[05] In particular in critical environments like healthcare where a patient's life can depend on the availability of for instance a system for consulting images, a need exists to reduce or even avoid downtime resulting from upgrades of client-server software applications.

[06] It is consequently an objective of the present invention to disclose a method for upgrading software with reduced downtime for the customer.

Summary of the Invention

[07] According to the present invention, the above-identified objective is achieved by the computer-implemented method for upgrading software having a client component, application server component and database component from an old version to a new version in a system with clients executing the client component, application servers executing the application server component, a load balancer directing a client out of these clients to a respective one of the application servers, and database servers executing the database component, the method being defined by claim 1 and comprising:

- in a first step, installing the new version of the database component on the database servers without interrupting application server service;
- in a second step subsequent to the first step, installing the new version of the application server component on the application servers without interrupting client service;
- in a third step subsequent to the second step, registering the new version of the application server component with the load balancer, using an IP address that is different from an IP address used to register the old version of the application server component; and
- in a fourth step subsequent to the third step, installing the new version of the client component with respective ones of the clients when respective users of the clients desire so.

[08] Thus, in line with the present invention, the database is first upgraded causing no downtime for the clients and application servers. The database then supports two versions of the database component in case the new version of the database component is not backwards compatible with the old version of the database component, or it supports only the new version of the database component in case this new version of the database component is backwards compatible with the old version of the database component. Backwards compatibility implies that the old

version of the application server component can be executed with the new version of the database component. Thereafter, the application servers are upgraded without service interruption for the clients. The different software versions installed on the application servers are registered with different IP addresses towards the load balancer. The clients pass the software version they support with every request to the load balancer. The load balancer shall detect the version information and direct a client to an application server supporting the version mentioned in the client's request. Consequently, as long as the new version of the software is not installed on a client, the new version of the application server component shall not be used. This allows for instance to test the new version of the application server extensively in order to guarantee successful installation thereof while the clients stay ignorant on the existence of the new version of the software. The new version of the client component of the software can be downloaded while the old version of the client component is still running. Clients and their respective users are then informed on the availability of the new version of the software. Only when the user desires so, e.g. when the user approves upgrading the client, when the user is locked out, or when inactivity is detected by the system for some time, the new version of the client component will be installed on the user's client device. This is done with minimum downtime for the user, typically a few minutes or even less.

[09] The software upgrade process according to the invention has the additional advantage that performance decrease of the application servers during the upgrade is limited. All application servers are continuously up and running during the upgrade causing the total application performance to stay stable during the upgrade.

[10] In accordance with an advantageous aspect of the present invention, defined by claim 2, the new version of the database component is backwards compatible with the old version of the database component.

[11] Indeed, in case the new version of the database component is backwards compatible with the old version of the database component, the old version of the application server component can be executed with the new version of the database component. This way, it is avoided that the old version and new version of the database component have to coexist on the database. In case of absence of

backwards compatibility between the new version and old version of the database component, the two versions have to be maintained on the database until all application servers and all clients have been upgraded successfully.

[12] Optionally, as defined by claim 3, the computer-implemented method for upgrading software according to the present invention applies versioning of at least important application objects in the first step.

[13] Indeed, to facilitate compatibility between the new version and old version of the database component, versioning of important objects such as workflows, templates, etc. can be applied. Versioning ensures that the new version of the database component can introduce new objects, e.g. workflows, while existing objects' behaviour remains unaltered. As a result thereof, the old version of the application server component can run with the new version of the database component of the software.

[14] According to an optional aspect of the computer-implemented method for upgrading software according to the present invention, the second step further comprises testing the new version of the application server component on the application servers without interrupting client service in order to verify successful installation.

[15] First, start-up of the application server whereon the new version of the application server component is installed can be tested. This way, unsuccessful start-up of the new version of the application server component on an application server can be detected and the next steps in the software upgrade process can be stopped without influencing the running version of the application server component. In addition, smoke testing can be performed to even better guarantee successful installation of the application server component while the clients remain ignorant of the ongoing software upgrade.

[16] Following another optional aspect of the computer-implemented method for upgrading software according to the present invention, defined by claim 5, the load balancer receives a request from a client out of the clients, the request specifying

which version of the client component is executed by the client, and re-directs the client to a respective one of the application servers executing a version of the application server component that corresponds with the version of the client component specified in the request.

[17] Indeed, preferably, the clients notify in each request to the load balancer which version of the client component they actually support. The load balancer extracts and interprets this information upon receipt of the request, and directs the client to an application server that has the corresponding version of the application server component running. This is made possible by using different IP addresses for different versions of the application server component of the software.

[18] In an embodiment of the computer-implemented method for upgrading software according to the present invention, defined by claim 6, the fourth step comprises downloading binaries of the new version of the client component to the clients without interrupting client service, and installing the new version of the client component with a respective client of the clients either in lock time or at a point in time when a user of the client allows interruption of his service.

[19] Thus, by downloading the binaries of the client component already to the client while the client maintains service, the downtime for the user is further reduced. Moreover, by enabling the user to decide when the upgrade of the client component shall take place, impact on the user of the small downtime needed to install the client component, is minimized or even reduced to zero in case the user elects to install the client component for instance during a period of inactivity or after logout.

[20] Optionally, as defined by claim 7, the computer-implemented method for upgrading software according to the present invention further comprises:

- cleaning the old version of the application component from the application servers as soon as all clients have installed the new version of the client component; and

- de-registering the old version of the client application component with the load balancer to thereby release IP addresses used for the old version of the client application component.

[21] Hence, as soon as all active clients have been upgraded towards the new version of the client component, the old version of the application server component may be de-registered at the load balancer, and the old version of the application server component may be deleted from the application servers in order to free-up memory space on the application servers.

[22] According to an advantageous aspect of the present invention, defined by claim 8, the computer-implemented method for upgrading software comprises pausing the deletion of the old version of the client component.

[23] Thanks to the pausing, the old version of the software and the new version of the software can be maintained on the client device, thus delaying the restart of the client device which is needed for the new version of the client component to become active. Pausing in other words gives further control to the user on the moment in time whereon the small service interruption shall take place.

[24] In addition to a computer-implemented method for upgrading software as defined by claim 1, the present invention also relates to a corresponding data processing system as defined by claim 9, comprising means for carrying out this method.

[25] The present invention further also relates to a corresponding computer program as defined by claim 10, comprising software code adapted to perform the method, and to a computer readable storage medium as defined by claim 11, comprising the computer program according to the present invention.

Brief Description of the Drawings

[26] Fig. 1 shows a system wherein an embodiment of the software upgrading method according to the present invention is used;

[27] Fig. 2 illustrates the first step in an embodiment of the software upgrading method according to the present invention;

[28] Fig. 3 illustrates the second and third steps in an embodiment of the software upgrading method according to the present invention;

[29] Fig. 4 illustrates the fourth step in an embodiment of the software upgrading method according to the present invention; and

[30] Fig. 5 illustrates the fifth step in an embodiment of the software upgrading method according to the present invention.

Detailed Description of Embodiment(s)

[31] Fig. 1 shows the architectural composition of a system wherein a client-server based software application is used. Such system consists of a database server cluster with several database servers, DB1 or 101A and DB2 or 101B, an application server cluster with several application servers, AS1 or 102A and AS2 or 102B and AS3 or 102C, a load balancer 103 and several clients, CLIENT1 or 104A and CLIENT2 or 104B and CLIENT 3 or 104C. Each one of the application servers 102A, 102B and 102C connects to one or more of the database servers 101A and 101B to persist the application data. The load balancer 103 is connected to each one of the application servers 102A, 102B and 102C, and ensures proper spreading of the load, i.e. the requests received from the clients 104A, 104B and 104C, over the different application servers 102A, 102B and 102C. The clients 104A, 104B and 104C all connect with the load balancer 103.

[32] A client-server based software application typically consists of a database component SWDB that is installed on the database servers 101A and 101B, an application server component SWAP that is installed on the application servers 102A, 102B and 102C, and a client component SWCL that is installed on the clients 104A, 104B and 104C. The load balancer registers the IP addresses of the various installations of the application server component SWAP, receives and interprets

requests from the clients, and directs these requests to the application servers taking into account their respective loads.

[33] As explained here above, various situations require software upgrades. Such a software upgrade usually impacts each of the levels in the system architecture depicted in Fig. 1:

- the client component SWCL needs to be upgraded from an old version SWCL_A to a new version SWCL_B;
- the application server component SWAP needs to be upgraded from an old version SWAP_A to a new version SWAP_B; and
- the database component SWDB needs to be upgraded from an old database model SWDB_A to a new database model SWDB_B.

[34] The invention consists in a specific upgrade sequence described in the following paragraphs with reference to Fig. 2 - Fig. 5, supporting zero down time for the clients 104A, 104B and 104C, and limiting the testing efforts for the organization deploying the software. It is noticed that Fig. 2 - Fig. 5 show the upgrade process for a single instantiation of the database 101 and a single client 104.

[35] In a first step of the upgrade process, illustrated by Fig. 2, the database component SWDB of the software is upgraded from the old version SWDB_A to the new version SWDB_B. The upgrade of the database component SWDB is performed while the application server component SWAP is running, causing no downtime of the application. To facilitate compatibility, versioning of important database objects like workflows, templates, etc., is utilized. This ensures that the new version can introduce new objects, e.g. workflows, while the existing object's behavior remains the same. In case the old version SWAP_A of the application server component SWAP is able to run against the new database model, i.e. in case the new version SWDB_B of the database component SWDB is backwards compatible, the old version SWDB_A of the database component SWDB can be deleted from the database server 101.

[36] In a second step of the software upgrade process illustrated by Fig. 3, the new version SWAP_B of the application server component SWAP is deployed and started

on all application servers, 102A, 102B and 102C. Unsuccessful startup of one of the application server nodes after installation of the new version SWAP_B of the application server component SWAP can be detected as a result of which next steps in the installation process can be stopped without influencing the running application. Additional smoke testing of the new version SWAP_B of the application server component SWAP increases the guaranty of a successful installation while the client component SWCL remains unaware of the upgrade ongoing.

[37] When successfully started and tested, the newly deployed application server components SWAP_B on the different application servers 102A, 102B and 102C are registered towards the load balancer 103 with IP-addresses IP@4, IP@5 and IP@6 that differ from the IP addresses IP@1, IP@2 and IP@3 at which the old version SWAP_A of the application server component SWAP installed on the different application servers 102A, 102B and 102C was registered . The load balancer 103 is thus aware which IP addresses support which version of the application server component of the software.

[38] The client component SWCL passes the software version, i.e. SWCL_A in the situation depicted in Fig. 3, with every call or request sent to the load balancer 103. The load balancer 103 detects the software version passed by the client 104 and directs the call or request from the client 104 to one of the application servers 102A, 102B or 102C supporting the corresponding version SWAP_A of the application server component. A call from a client with the old version SWCL_A of the client component will thus never be redirected to an application server running only the new version SWAP_B of the application server component. This is an important condition. By enforcing this condition, the software upgrade process according to the present invention ensures that a client will only need to be tested and will only run against the application server component version with which it was deployed.

[39] In a third step of the software upgared process, illustrated by Fig. 4, executed after successfully installing the new version SWAP_B on the application servers 102A, 102B and 102C, the clients are informed of the new version SWCL_B of the client component SWCL, allowing them to finish any ongoing server calls and informing the users of these clients of the newly available version. When the user has

the time to upgrade, or when inactivity of the system is detected, e.g. because the user is locked out after a period of inactivity, upgrade of the client 104 towards the new version SWCL_B of the client component SWCL of the software is triggered.

[40] In a fifth step of the software upgrade process, illustrated by Fig. 5, after all active clients have been upgraded towards the new version SWCL_B of the client component SWCL, a cleanup operation is performed. The old version SWCL_A of the client component SWCL is deleted from the clients and the old version SWAP_A of the application server component SWAP is deregistered from the load balancer 103. As a result, support and maintenance services for the old version of the software can be stopped.

[41] The side by side activation of different runtime versions of the software that is inherent to the software upgrade method according to the present invention, allows first trouble shooting analysis thus reducing risk of unsuccessful upgrades.

[42] Since the load balancer is aware of the different software versions, client calls are always channeled to the correct software version.

[43] The software upgrade method according to the invention further enforces client-server connections not allowing an old client component version to connect to a new application server component. This limits testing effort on version compatibility and allows frequent updates of the software while maintaining compliancy with high standard regulations as for instance applicable in healthcare.

[44] The method according to the invention shall typically be computer-implemented to run on a data processing system or computing device. A data processing system or computing device that is operated according to the present invention can include a workstation, a server, a laptop, a desktop, a hand-held device, a mobile device, a tablet computer, or other computing device, as would be understood by those of skill in the art.

[45] The data processing system or computing device can include a bus or network for connectivity between several components, directly or indirectly, a memory or

database, one or more processors, input/output ports, a power supply, etc. One of skill in the art will appreciate that the bus or network can include one or more busses, such as an address bus, a data bus, or any combination thereof, or can include one or more network links. One of skill in the art additionally will appreciate that, depending on the intended applications and uses of a particular embodiment, multiple of these components can be implemented by a single device. Similarly, in some instances, a single component can be implemented by multiple devices.

[46] The data processing system or computing device can include or interact with a variety of computer-readable media. For example, computer-readable media can include Random Access Memory (RAM), Read Only Memory (ROM), Electronically Erasable Programmable Read Only Memory (EEPROM), flash memory or other memory technologies, CDROM, digital versatile disks (DVD) or other optical or holographic media, magnetic cassettes, magnetic tape, magnetic disk storage or other magnetic storage devices that can be used to encode information and can be accessed by the data processing system or computing device.

[47] The memory can include computer-storage media in the form of volatile and/or nonvolatile memory. The memory may be removable, non-removable, or any combination thereof. Exemplary hardware devices are devices such as hard drives, solid-state memory, optical-disc drives, or the like. The data processing system or computing device can include one or more processors that read data from components such as the memory, the various I/O components, etc.

[48] The I/O ports can allow the data processing system or computing device to be logically coupled to other devices, such as I/O components. Some of the I/O components can be built into the computing device. Examples of such I/O components include a microphone, joystick, recording device, game pad, satellite dish, scanner, printer, wireless device, networking device, or the like.

[49] Although the present invention has been illustrated by reference to specific embodiments, it will be apparent to those skilled in the art that the invention is not limited to the details of the foregoing illustrative embodiments, and that the present invention may be embodied with various changes and modifications without

departing from the scope thereof. The present embodiments are therefore to be considered in all respects as illustrative and not restrictive, the scope of the invention being indicated by the appended claims rather than by the foregoing description, and all changes which come within the meaning and range of equivalency of the claims are therefore intended to be embraced therein. In other words, it is contemplated to cover any and all modifications, variations or equivalents that fall within the scope of the basic underlying principles and whose essential attributes are claimed in this patent application. It will furthermore be understood by the reader of this patent application that the words "comprising" or "comprise" do not exclude other elements or steps, that the words "a" or "an" do not exclude a plurality, and that a single element, such as a computer system, a processor, or another integrated unit may fulfil the functions of several means recited in the claims. Any reference signs in the claims shall not be construed as limiting the respective claims concerned.

CLAIMS

1. A computer-implemented method for upgrading software having a client component (SWCL), application server component (SWAP) and database component (SWDB) from an old version (SWCL_A, SWAP_A, SWDB_A) to a new version (SWCL_B, SWAP_B, SWDB_B) in a system with clients (104A, 104B, 104C; 104) executing said client component (SWCL), application servers (102A, 102B, 102C) executing said application server component (SWAP), a load balancer (103) directing a client out of said clients (104A, 104B, 104C; 104) to a respective one of said application servers (102A, 102B, 102C), and database servers (101A, 101B; 101) executing said database component (SWDB), said method comprising:

- in a first step, installing said new version (SWDB_B) of said database component (SWDB) on said database servers (101A, 101B; 101) without interrupting application server service;

- in a second step subsequent to said first step, installing said new version (SWAP_B) of said application server component (SWAP) on said application servers (102A, 102B, 102C) without interrupting client service;

- in a third step subsequent to said second step, registering said new version (SWAP_B) of said application server component (SWAP) with said load balancer (103), using an IP address (IP@4, IP@5, IP@6) that is different from an IP address (IP@1, IP@2, IP@3) used to register said old version (SWAP_A) of said application server component (SWAP); and

- in a fourth step subsequent to said third step, installing said new version (SWCL_B) of said client component (SWCL) with respective ones of said clients (104A, 104B, 104C; 104) when respective users of said clients (104A, 104B, 104C; 104) desire so.

2. A computer-implemented method for upgrading software according to claim 1,

wherein said new version (SWDB_B) of said database component (SWDB) is backwards compatible with said old version (SWDB_A) of said database component (SWDB).

3. A computer-implemented method for upgrading software according to claim 2,

wherein versioning of at least important application objects is applied in said first step.

4. A computer-implemented method for upgrading software according to claim 1,

wherein said second step further comprises testing said new version (SWAP_B) of said application server component (SWAP) on said application servers (102A, 102B, 102C) without interrupting client service in order to verify successful installation.

5. A computer-implemented method for upgrading software according to claim 1,

wherein said load balancer (103) receives a request from a client out of said clients (104A, 104B, 104C; 104), said request specifying which version of said client component (SWCL) is executed by said client, and re-directs said client to a respective one of said application servers (102A, 102B, 102C) executing a version of said application server component (SWAP) that corresponds with said version of said client component (SWCL) specified in said request.

6. A computer-implemented method for upgrading software according to claim 1,

wherein said fourth step comprises downloading binaries of said new version (SWCL_B) of said client component (SWCL) to said clients (104A, 104B, 104C; 104) without interrupting client service, and installing said new version (SWCL_B) of said client component (SWCL) with a respective client of said clients (104A, 104B, 104C; 104) either in lock time or at a point in time when a user of said client allows interruption of his service.

7. A computer-implemented method for upgrading software according to claim 1, further comprising:

- cleaning said old version (SWAP_A) of said application component (SWAP) from said application servers (102A, 102B, 102C) as soon as all clients (104A, 104B,

104C; 104) have installed said new version (SWCL_B) of said client component (SWCL); and

- de-registering said old version (SWAP_A) of said client application component (SWAP) with said load balancer (103) to thereby release IP addresses (IP@1, IP@2, IP@3) used for said old version (SWAP_A) of said client application component (SWAP).

8. A data processing system comprising means for carrying out the method of any of claims 1 to 7.

9. A computer program comprising software code adapted to perform the method of any of claims 1 to 7.

10. A computer readable storage medium comprising the computer program of claim 9.

WO 2015/028538 PCT/EP2014/068228

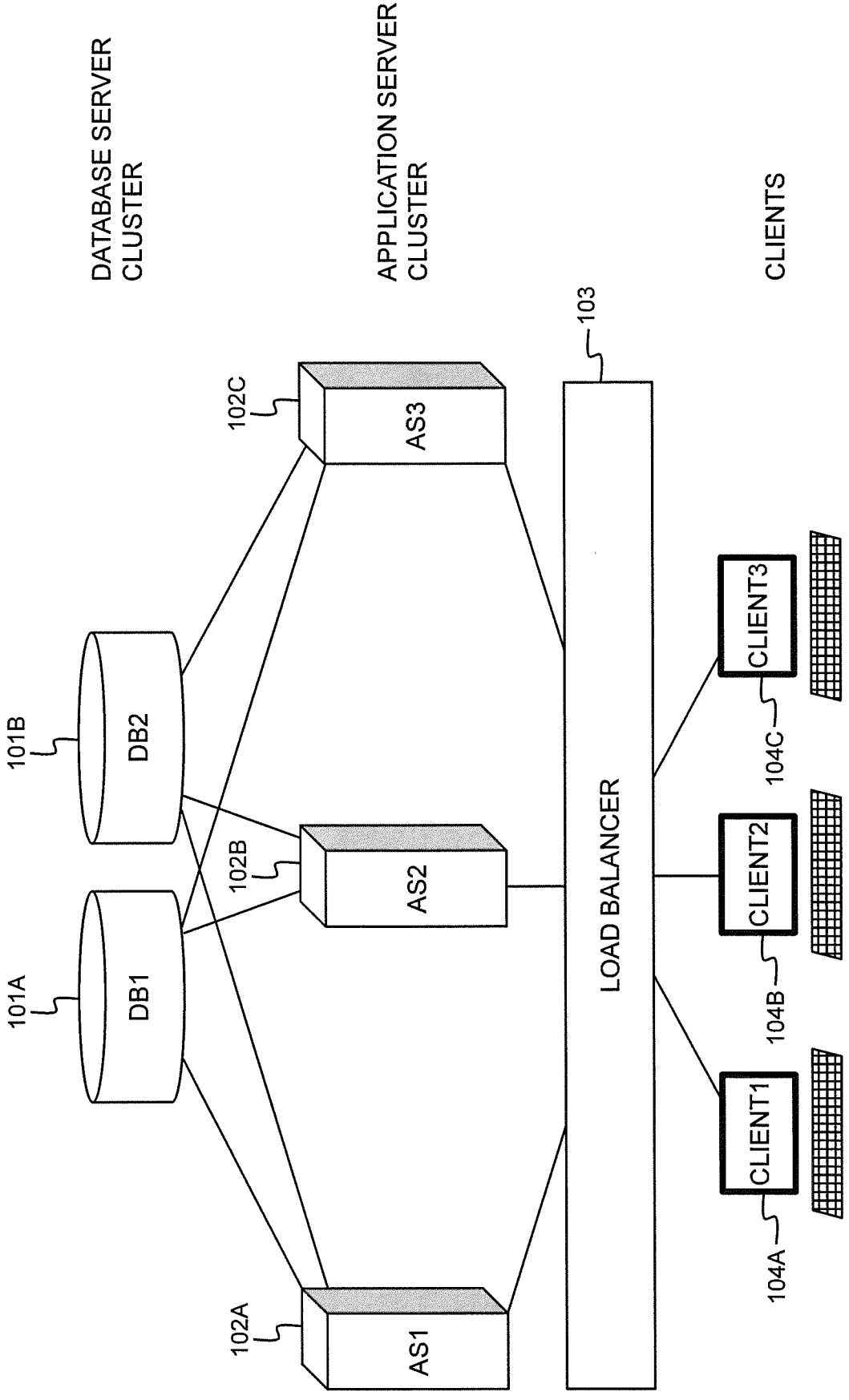

Fig. 1

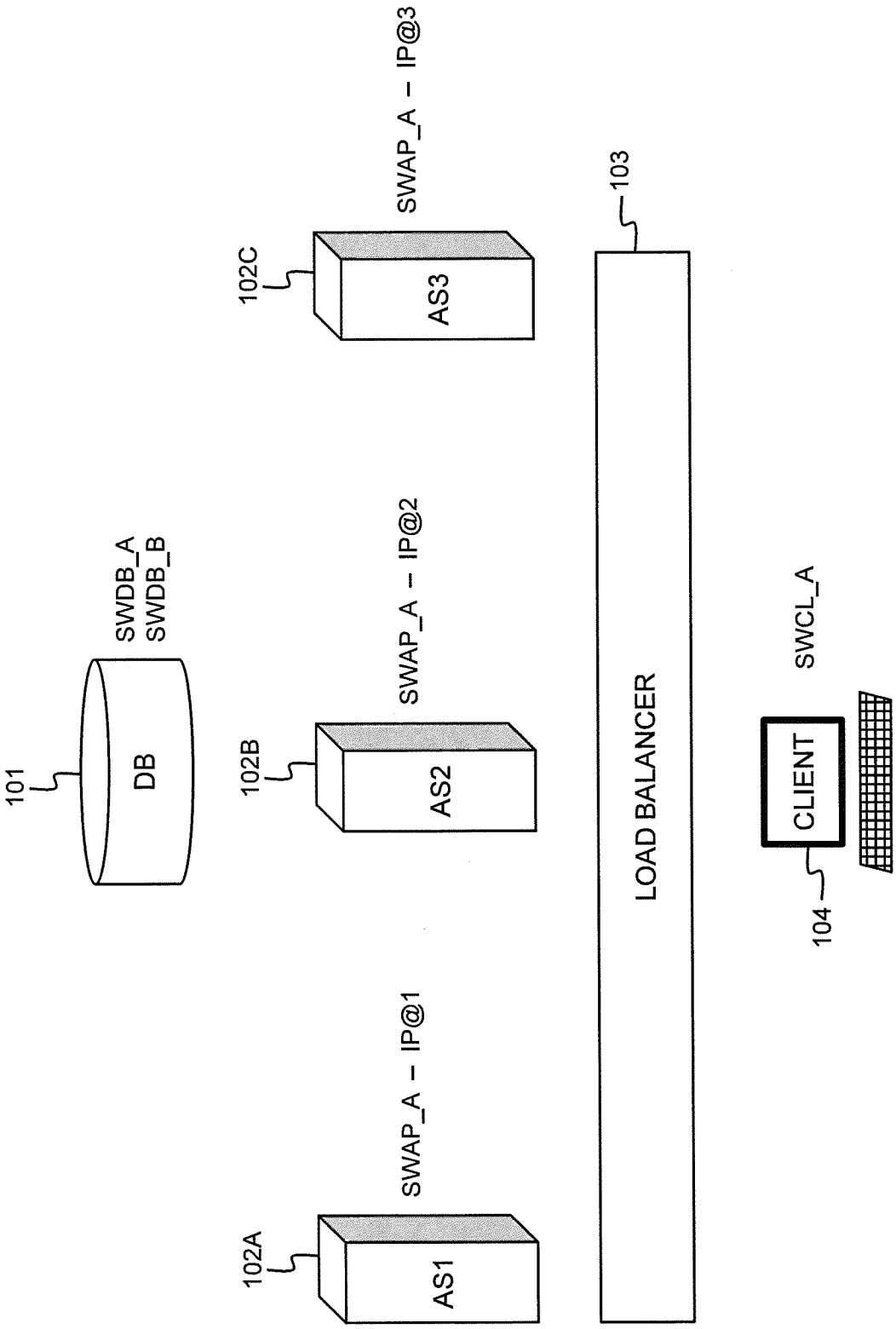
101
DB
SWDB_A
SWDB_B
102A
AS1
SWAP_A – IP@1
102B
AS2
SWAP_A – IP@2
102C
AS3
SWAP_A – IP@3
103
LOAD BALANCER
CLIENT
SWCL_A
104

Fig. 2

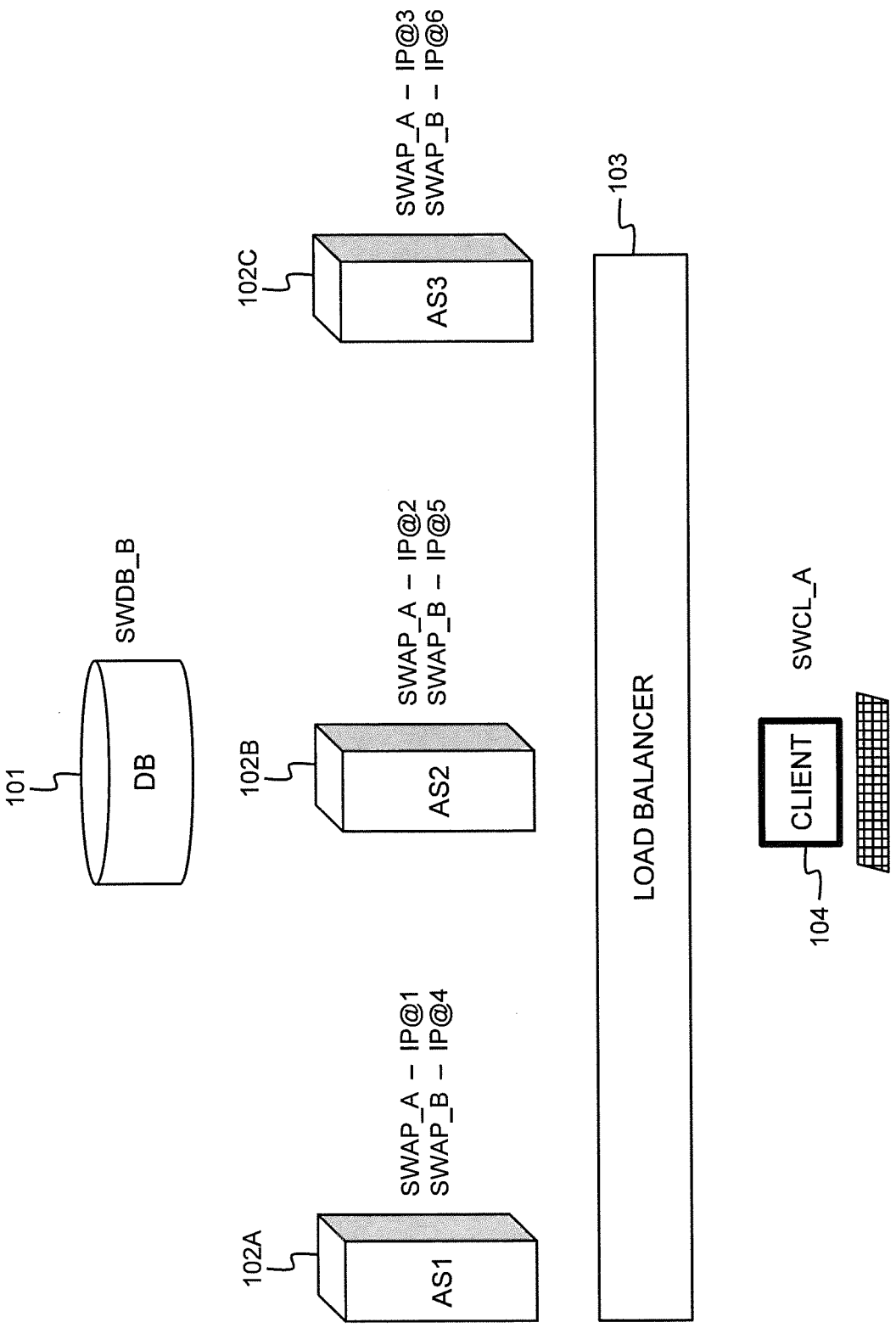
SWAP_A – IP@3
SWAP_B – IP@6
102C
AS3
103
SWDB_B
SWAP_A – IP@2
SWAP_B – IP@5
101
DB
102B
AS2
LOAD BALANCER
SWCL_A
CLIENT
104
SWAP_A – IP@1
SWAP_B – IP@4
102A
AS1

Fig. 3

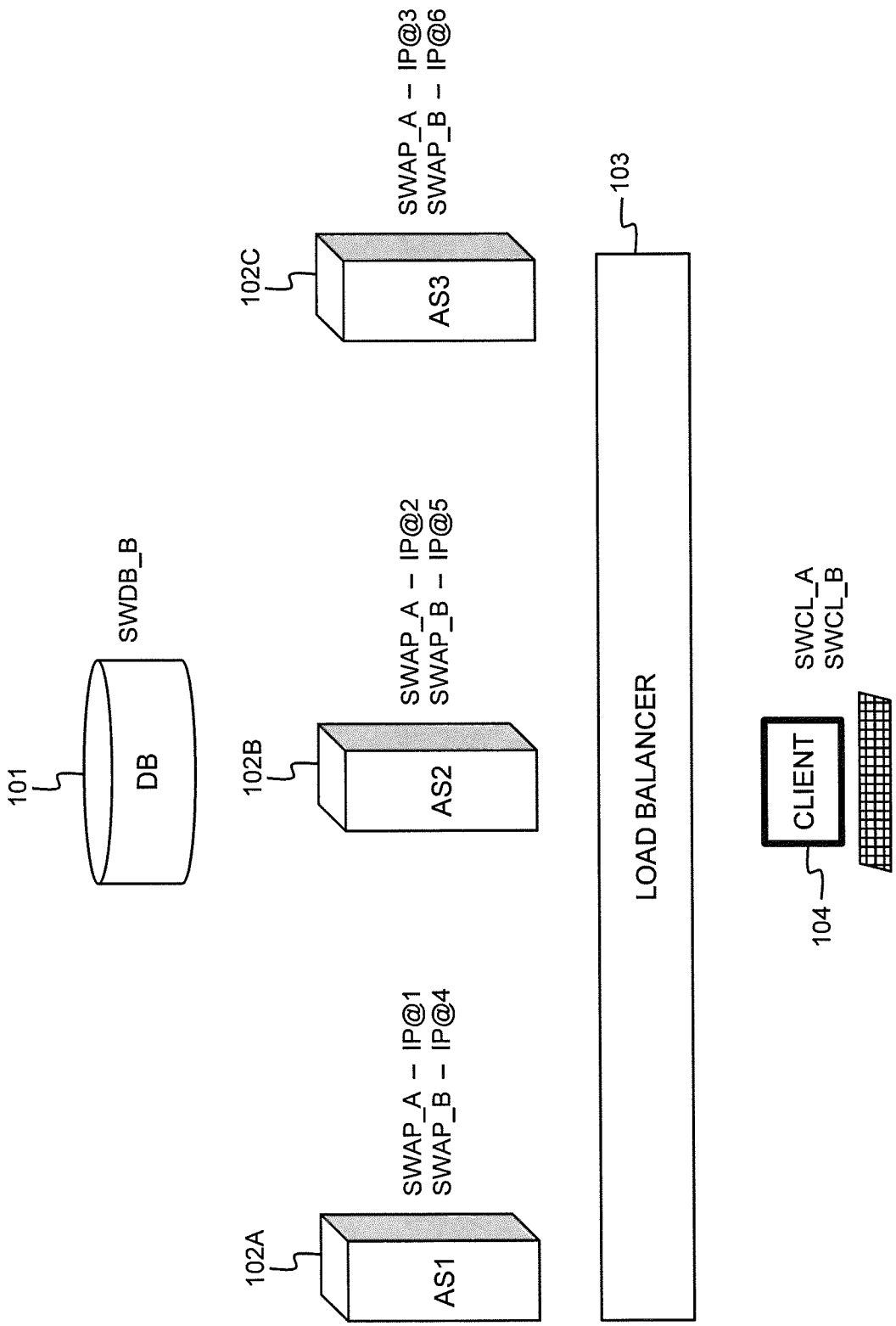
101
DB
SWDB_B
102A
AS1
SWAP_A – IP@1
SWAP_B – IP@4
102B
AS2
SWAP_A – IP@2
SWAP_B – IP@5
102C
AS3
SWAP_A – IP@3
SWAP_B – IP@6
103
LOAD BALANCER
104
CLIENT
SWCL_A
SWCL_B

Fig. 4

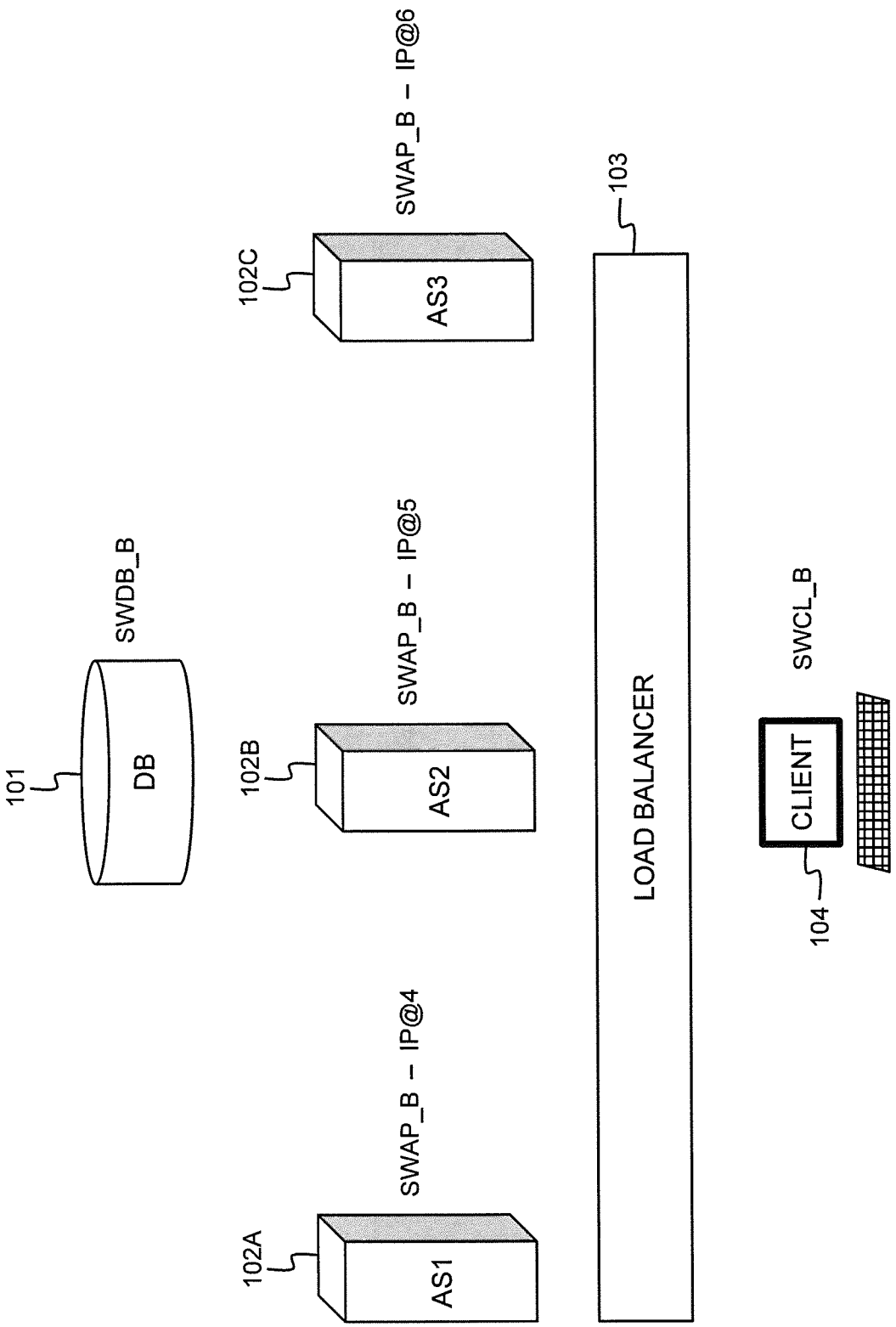
SWAP_B – IP@6
102C
AS3
103
SWDB_B
101
DB
SWAP_B – IP@5
102B
AS2
LOAD BALANCER
SWCL_B
CLIENT
104
SWAP_B – IP@4
102A
AS1

Fig. 5

INTERNATIONAL SEARCH REPORT

International application No
PCT/EP2014/068228

A. CLASSIFICATION OF SUBJECT MATTER
INV. G06F9/445
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

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	GB 2 411 995 A (SUN MICROSYSTEMS INC [US]) 14 September 2005 (2005-09-14) abstract claim 1; figure 1	1-10
A	US 8 200 842 B1 (LAU PRISCILLA [US]) 12 June 2012 (2012-06-12) abstract	1-10

☐ 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
22 September 2014	01/10/2014

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	Müller, Tobias

1

INTERNATIONAL SEARCH REPORT

Information on patent family members

International application No
PCT/EP2014/068228

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
GB 2411995 A	14-09-2005	NONE	
US 8200842 B1	12-06-2012	NONE	