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

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
27 August 2015 (27.08.2015)

WIPO | PCT

(10) International Publication Number
WO 2015/124668 A1

(51) International Patent Classification:
G06F 17/30 (2006.01) *G06F 9/50* (2006.01)

(21) International Application Number:
PCT/EP2015/053498

(22) International Filing Date:
19 February 2015 (19.02.2015)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
14155970.8 20 February 2014 (20.02.2014) EP

(71) Applicant: **DEUTSCHE TELEKOM AG** [DE/DE]; Friedrich-Ebert-Allee 140, 53113 Bonn (DE).

(72) Inventors: **LI, Weikai**; Hong Kong University of Science and Technology, Computer Science and Engineering, Clear Water Bay, Kowloon, Hong Kong (CN). **HUI, Pan**; Hong Kong University of Science and Technology, Computer Science and Engineering, Clear Water Bay, Kowloon, Hong Kong (CN). **WU, Ting**; Hong Kong University of Science and Technology, Computer Science and Engineering, Clear Water Bay, Kowloon, Hong Kong (CN). **PEYLO, Christoph**; Nordweg 14, 49401 Damme (DE).

(74) Agent: **VOSSIUS & PARTNER**; Siebertstraße 3, 81675 München (DE).

(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, 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.

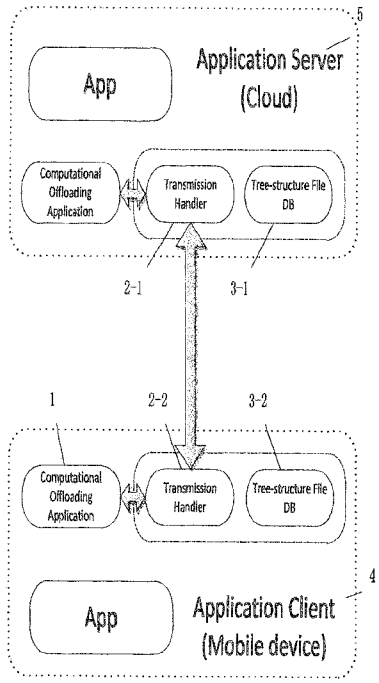
(84) Designated States *(unless otherwise indicated, for every kind of regional protection available)*: ARIPO (BW, GH, GM, KE, LR, LS, MW, MZ, NA, RW, SD, SL, ST, SZ, TZ, UG, ZM, ZW), Eurasian (AM, AZ, BY, KG, KZ, RU,

[Continued on next page]

(54) Title: TREE-STRUCTURE STORAGE METHOD FOR MANAGING COMPUTATION OFFLOADING DATA

(57) Abstract: A method is shown for managing data during an offloading computational task to increase the computational efficiency of both servers and user devices (e.g. mobile devices). This is feasible because during the computation offloading process, the cloud generates intermediate data (e.g. execution results) that is still in processing and delivers it to the user device. The main contribution of the present invention is that a tree-structure storing approach is used to manage the intermediate data that could be reused and shared. By leveraging the tree-structure file, the local user device can effciently continue the offloading task when a disconnection occurs. For the server side, by maintaining and sharing the tree structure files within the cloud, it could reduce the processing time of similar tasks dramatically.

Figure 1



WO 2015/124668 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))*

— *before the expiration of the time limit for amending the claims and to be republished in the event of receipt of amendments (Rule 48.2(h))*

Tree-structure storage method for managing computation offloading data

The present invention relates to managing intermediate data which is being generated by a cloud while processing an offloading task assigned by a user device (e.g. mobile device), and more particularly, to managing such intermediate data in a tree-structure method.

In an environment where devices equipped with varying computational capabilities and battery lifetime, developers and users have to offload some local computation intensive tasks for remote execution. Recently, lots of works have been done to construct computational network and develop applications in order to send tasks to the cloud for processing, such as MAUI [1], ThinkAir [2] and CloneCloud [3]. These approaches greatly reduce computational pressure on user devices (in particular mobile devices) and save the battery lifetime.

[1] Cuervo E, Balasubramanian A, Cho D, et al. MAUI: making smartphones last longer with code offload[C]//Proceedings of the 8th international conference on Mobile systems, applications, and services. ACM, 2010: 49-62.

[2] Sokol Kosta, Andrius Aucinas, Pan Hui, Richard Mortier, and Xinwen Zhang ThinkAir: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading//In Proceeding of the 31st Annual IEEE International Conference on Computer Communications (IEEE INFOCOM 2012), Orlando, Florida USA, March 2012

[3] Chun B G, Ihm S, Maniatis P, et al. Clonecloud: elastic execution between mobile device and cloud[C]//Proceedings of the sixth conference on Computer systems. ACM, 2011: 301-314.

For an offloading task the user device sends at least some raw data to the cloud for processing. These raw data are processed in a series of procedures and delivered to the user device as an execution result of the offloading task. For example, in a face-detection task, the face-detection method program (codes) is first sent to the

cloud. Then, when the user device captures an image (raw data), it sends this image to the cloud for processing. The codes (program) may be prepared at the cloud, instead of the codes being sent from the user device to the clouds.

Traditional offloading applications such as those indicated above only care about the execution result. We propose a useful method of managing intermediate data which is being generated during the processing, which will be explained hereinafter. In the specification, the term "intermediate data" refers to data which is being generated in a middle step of the offloading task and is still in processing (i.e. time-varying).

With the growing in storage space of portable computing devices, the devices can store more data. This opens up the possibility that a task (e.g. 3D-modeling), the result of which requires a relatively large storage capability, may be executed on the portable computing device.

We design a method to manage and utilize the intermediate data during the offloading using a tree-structure to manage the intermediate data; the tree-structure has good advantages in inserting and deleting a node as well as in maintaining the tree. This method also takes advantage of an increment data transmission approach, in order to improve the offloading performance in mobile computation environments.

One aspect of the present invention is to provide a method of managing intermediate data that is being generated and still in processing during an offloading computational task to a cloud, in which a user device connects to the cloud, and the cloud generates a set of data that will be delivered to the user device and form a result of the offloading task, the method comprising:

(a) a transmission handler on the user device delivers the offloading task to the cloud;

(b) the cloud analyzes whether the type of the offloading task is the one to which a tree-structure method is applicable, and in the affirmative, manages said intermediate data in the tree-structure method and, a transmission handler on the cloud transmits said intermediate data to the user device in increment data;

(c) the cloud creates a tree-structure file and a log file, the tree-structure file storing said intermediate data in the form of nodes of a tree, the log file including information on said tree-structure file, the tree-structure file and the log file both being updated during the offloading task;

(d) the transmission handler on the cloud transmits said log file to the user device; and

(e) the user device creates a tree-structure file on the user device so as to synchronize it to the tree-structure file on the cloud, based on the received intermediate data and the received log file.

In the specification the term "cloud" refers to the one generally used in computer science, and includes one server or a plurality of remote servers and storage means. The servers are networked to allow centralized data storage and online access to computer services or resources.

When producing intermediate data during the remote execution, the cloud stores the intermediate data in nodes and inserts the nodes into the tree. In the present specification, we call this storing method a tree-structure method; intermediate data are stored in a tree-structure file in the forms of nodes of the tree. It costs very few computational resources and time to build and maintain a tree as well as to manage intermediate data. After constructing such a tree-structure file, it will be maintained at both the server (cloud) side and the user device (e.g. mobile device) side simultaneously. The tree-structure file will help users efficiently review the offloading task on the user device and reuse some data in the future.

We design two kinds of child nodes to store the intermediate data in an offloading task, and a method to create the tree-structure and decide the location of a new node in the tree-structure. When the tree is created, the user can get not only the final execution result, but also some intermediate data and sub-data. Thus when a similar offloading task is being executed, the user can take advantage of previous processed data that is stored in a tree. This greatly reduces the duplicated computational task.

The user device receives a log file that is created by the cloud. The tree-structure file and the log file will improve the offloading performance because the user device synchronizes its tree-structure file statue of the tree-structure file on the cloud, based on the received intermediate data and the received log file. For example, it will allow the user to check the rate of processing progress (e.g. 70% of the offloading task is completed) and/or review the (completed) offloading task or (already finished) different execution stages on the user device. When a disconnection occurs, the user device can efficiently judge the processing status on the cloud at the time of disconnection, and continue the unfinished part of the offloading task via the tree-structure file and the log file stored on the user device.

BRIEF DESCRIPTION OF THE DRAWINGS

Figure 1 illustrates a schematic framework of a system that is adapted to use a managing method according to one embodiment of the present invention.

Figure 2 illustrates a flow diagram of one embodiment of the present invention.

Figure 3 illustrates a flow diagram that illustrates a transmission procedure by the transmission handler on the cloud.

Figure 4 illustrates an example of the tree-structure file.

Figure 5 illustrates an example of 3D-cow-model-processing task using the tree-structure file.

Figure 6 illustrates a flow diagram of constructing an original result file from the tree-structure file at the side of the mobile device.

Detailed description of the preferred embodiments

A managing method according to preferred embodiments of the present invention will be described with reference to the accompanying drawings.

Figure 1 is a schematic framework of a system using a managing method according to an embodiment of the present invention. The system comprises an application client (mobile device, broadly user device) 4 and an application server (cloud) 5 for connection (in a wired or wireless manner) with each other. The cloud 5 includes

storage means (not shown). As will be appreciated by the skilled person, the cloud 5 may include a group of remote servers that are networked to allow centralized data storage.

The mobile device 4 is provided with a module 1, an independent computational offloading application (e.g. MAUI [1] or ThinkAir [2]) for deciding the offloading task, i.e. deciding which task should be offloaded to the cloud 5 for remote execution. The mobile device 4 is further provided with a module 2-2, transmission handler for delivering the task to the cloud 5.

When the task is under processing, the cloud 5 will keep creating intermediate data, and transmitting, via a module 2-1, another transmission handler on the cloud 5, the intermediate data to the mobile device 4. As will be explained below, the transmission handler 2-1 is adapted to transmit intermediate data as increment data. These two transmission handlers will connect each other and exchange data between the mobile device 4 and the cloud 5.

The cloud 5 is adapted to create and store a tree-structure file and a log file which are being generated (i.e. updated) during the processing. The tree-structure file stores intermediate data using a tree-structure method, i.e. in the form of nodes of a tree (which will be explained below in more detail). In one embodiment, the tree structure file and the log file are stored in a module 3-1, tree-structure file database. The log file may be stored in any other storage area on the cloud 5. The log file will be transmitted to the mobile device 4.

The mobile device 4 is adapted to create a tree-structure file locally so as to synchronize it to the tree-structure file on the cloud 5 based on the received log file and the received intermediate data. In one embodiment, the created tree-structure file is stored in a module 3-2, tree-structure database. The received log file may be stored in the tree-structure database 3-2 or any other storage area on the mobile device 4.

Figure 2 is a flow diagram that illustrates a managing method that is carried out by the system shown in Fig. 1, according to one embodiment of the present invention.

First in step 6, the independent computational offloading application 1 on the mobile device 4 decides the offloading task. Then, the transmission handler 2-2 on the mobile device 4 delivers the task to the cloud 5. This step (corresponding to step (a) of claim 1) is performed in the same manner as the conventional methods [1]-[3]. That is, for an offloading task the mobile device 4 sends at least some raw data to the cloud 5 for processing. Codes (program) for processing the raw data to execute the offloading task may be provided in advance from the mobile device 4 or prepared at the cloud 5. As will be appreciated by the skilled person, data and codes may be delivered, for example, by means of network sockets.

The process proceeds to step 7, in which the cloud 5 receives the offloading task, and analyzes the task in order to estimate an execution time (which will explained below) on the cloud 5, and the process proceeds to step 8.

In step 8, the cloud 5 decides whether an increment data transmission approach and the tree-structure method is applicable or not to the offloading task. In the method according to the present invention, the transmission handler 2-1 will transmit the generated intermediate data back to the mobile device 4 in an increment data transmission approach. In the specification, the term "increment data transmission approach" refers to an approach that transmits only a portion of (time-varying) intermediate data that has changed since a prior transmission. This data designates "increment data" which is an increment between two intermediate data created on the cloud in different time.

The present invention uses a tree-structure method to manage intermediate data during the offloading task. The intermediate data will be stored in a tree-structure file, i.e. in nodes to create the tree. As mentioned above, a synchronized tree-structure file is created locally on the mobile device 4. Therefore, in one embodiment, the tree-structure file may be used to allow a user to review the whole offloading task

and/or take the tree-structure file for further process. Since the tree structure has great advantages in inserting, deleting and maintaining, it costs a little to build and maintain the tree structure file during the offloading.

However, not all the offloading task can use the tree-structure method. The tree-structure method is applicable to offloading task with the following property: The intermediate data that is being created during the offloading task is dividable into small sections. These subsections can be aggregated again. Each subsection of the intermediate data is stored in a node of the tree. Therefore, during the offloading task the tree will change its form by addition of nodes, each of which stores a sub-section which has not yet stored in any of the existing node(s).

When the offloading task has this property, the increment data can also be obtained for the increment data transmission approach.

For example, a data file of 3D model can be divided in a set of meshes. The meshes can reconstruct the 3D models. In this scenario, each subsection of intermediate data may be a sub-set of these meshes. The sub-set of meshes will be stored in a node, and the node is inserted into the tree. In one embodiment, increment data is equal to each subsection of intermediate data (that is, the subsection is transmitted at a time). However, when each subsection of intermediate data is further dividable, increment data for the increment data transmission approach may be one or more divisions of each subsection (e.g. increment data is equal to one mesh only). So for this offloading task of processing 3D model, the tree-structure method and the increment data transmission approach are available.

As a preferable property of the offloading task, the task may be processed in a series of separate procedures in the sense that the generated data of each procedure is stored as a node of the tree (i.e. to build a single node one procedure is required). As an example to explanation, a 3D cow model may be processed in three procedures: adding detailed meshes of a head of the cow, adding detailed meshes of a tail of the cow, and adding a texture of cow. The generated data of

these three procedures will be stored in three nodes, which will then be inserted to the tree.

Referring back to Fig, 2, if the increment data transmission approach and the tree-structure method are applicable, the process proceeds to step 9. The cloud 5 will start to run the offloading task and deliver the current execution result (current intermediate data) to the transmission handler 2-1, in a pre-set period (which will be explained below). The transmission handler 2-1 compares each delivered data (i.e. compares the current intermediate data with previous intermediate data) to get increment data, and transmits the increment data to the mobile device 4 in step 9.

In step 10, the cloud 5 manages these transmitted data in the tree-structure method. The cloud 5 stores a tree-structure file (storing the intermediate data in the form of nodes in a tree) in the database 3-1. Then the process proceeds to step 11, the cloud 5 creates a log file including information on the tree-structure file. In one embodiment, the log file records the information of the transmission including transmission data ID, location of the increment data (i.e. subsection of intermediate data) in the tree and/or the operations to the tree. The operations to the tree include when and where to insert a new node. In one example, the transmission handler 2-1 on the cloud 5 may create the log file. In another example, (other component of) the cloud 5 may create the log file based on information on the operations that the component itself has created as well as on transmission data ID and location of the increment data supplied from the transmission handler 2-1.

The log file is transmitted to the mobile device 4.

In step 12, if the task is finished, the process ends. Otherwise, then process proceeds to step 9, the transmission handler 2-1 on the cloud 5 keeps receiving the current execution result and transmitting the increment data.

In step 8, if the increment data transmission approach and the tree-structure method are not applicable to the offloading task, the process proceeds to step 13. For

example, if the offloading task only requires data (for instance, date and time (of e.g. Germany), photo) that exists on the cloud 5 (but are not available to the mobile device 4) or running codes on the cloud 5, the increment data transmission approach and the tree-structure method are not available for this offloading task. The cloud 5 will run the offloading task and transmit the result to the mobile device in a normal approach, such as only delivering the final execution result to the mobile device 4. Then the process proceeds to an end. The same applies when intermediate data is not dividable. That is, when intermediate data is not dividable (e.g. an offloading task of creating a graph), only the final execution result is delivered to the mobile device 4.

In one embodiment, after the cloud 4 confirms each processing procedure (in other words, the cloud 4 confirms how the offloading task should be done) and initiates the procedure, the transmission handler 2-1 on the cloud 4 may deliver increment data of each processing procedure in several batches, instead of one batch, which corresponds to a case where increment data is equal to a subsection (e.g. texture of the cow in case of creating a 3D cow model) of intermediate data. This will reduce bandwidth usage.

As the pre-set transmission period, the cloud may set, for example, 1 second as default. Depending on the varying offloading task, the cloud may adjust the transmission period to a proper one. The transmission handler 2-1 on the cloud 4 acquires a current procedure's estimated execution time in step 7. The frequency of transmission (i.e. times to transmit the data in one procedure) may preferably be at least 10 times, and at most 100 times. Then, the procedure's estimated execution time is divided by the frequency of transmission to get the transmission period. The cloud 4 will adjust (increase or decrease) the frequency of transmission to make the transmission period close to 1 second. However, if the frequency of transmission is out of the above-mentioned preferable range, the cloud 4 may increase and decrease the transmission period to make the frequency of transmission acceptable. The transmission handler 2-1 on the cloud 4 will keep transmitting the data in this pre-set period until finishing the current procedure.

Figure 3 is a flow diagram illustrating how the transmission handler 2-1 on the cloud 4 works in one embodiment of the present invention. First in step 14, the transmission handler 2-1 obtains the current execution result and execution status (whether or not the procedure is new) in a pre-set period, and the process proceeds to step 15. In step 15, if the handler 2-1 receives some data (execution result) from this procedure before (i.e. the procedure is not new; this corresponds to delivering increment data of one processing procedure in several batches), the process proceeds to step 16. In step 16, the current execution result is compared with the former result that is stored in a temporary memory (not shown). The transmission handler 2-1 uses a subtraction method to get increment data.

For example, when a 3D cow model is processed on the cloud, the first procedure may be creating skeleton meshes of the 3D cow. Then, the skeleton meshes of the 3D cow are stored in the temporary memory. After that, the next procedure may be to create detailed meshes of the cow's head. When this procedure is finished, the current execution result is compared to the former result in order to get increment data which is the meshes of the cow's head. Then, this 3D cow with a detail head will be stored in the temporary memory instead of the former one. After this step, the process proceeds to step 17. In step 15, if the handler 2-1 does not receive any data (execution result) from this procedure before, the process also proceeds to step 17. In step 17, the temporary memory stores the current execution result. The result in the temporary memory will be kept until the memory is updated. Finally in step 18, the transmission handler 2-1 transmits the increment data to the mobile device 4 and the process proceeds to an end.

Figure 4 shows an example of a tree-structure file. A node 19 is a root to initialize the tree, and the root will not store any data. When the cloud 5 finishes one procedure of the offloading task, if the subsection data (of intermediate data) of this procedure enriches intermediate data in comparison to an existing node, this subsection data will be stored in a node 20 as the existing node's child node or leaf node. In the specification the node 20 is referred to as Detailed Element Node.

If the subsection data (of intermediate data) of the procedure is independent, the tree will create a node 21 which stores the subsection data. In the specification the node 21 is referred to as Additional Element Node. An additional element node 21 will be linked to the root 19. In other words, the child node of the root will necessarily be an additional element node. However, in some cases, another element node may be linked to an additional element node which may or may not be a child node of the root (e.g. grandfather, parent and child additional element nodes may be linked to each other while the grandfather additional element node is linked to the root). Specifically, the child additional element node may be independent of its parent additional element node in the sense that the child additional element node does not enrich intermediate data in comparison to the parent additional element node, but is associated with the parent additional element node. The location of the child additional element node in the tree describes the relationship to its parent additional element node. In summary, an additional element node can only be linked to other additional element node or a root as its child node.

Figure 5 shows an example of a 3D-cow-model-processing offloading task (i.e. 3D rendering) to illustrate the tree-structure file. A root is created to initialize the tree. And the cloud will start to run this offloading task.

First, when the procedure of meshes of the skeleton is finished, the tree will create an additional element node to store these meshes, because the subsection data is the meshes of the skeleton and independent (there is only a root). Second, when the cloud finishes the procedure of detailed meshes of cow's head, this subsection data (meshes of cow's head) will be stored in a detailed element node which links to the node of the skeleton, because this subsection data enriches the detailed meshes of the skeleton of the cow. Third, when the cloud creates a white texture of the cow, the data of the white texture will be also stored in a detailed element node which is linked to the node of the detailed head. This is because the data of the while texture enriches current intermediate data (skeleton + head) in comparison to the node of the head of cow. However, the detailed element node may be directly

linked to the node of the skeleton (that is an additional element node) because the data of the while texture enriches intermediate data at the time of creating the skeleton.

Forth, if a model of a tree near the cow is additionally created, because the tree has little or no relationship to the cow, the meshes of tree will be stored in an additional element node. This additional element node will be linked to the root. Finally, if a bird is created on the tree, the meshes of the bird will be stored in another additional element node. Because the bird is independent of the tree in the sense that the meshes of bird do not enrich current intermediate data (skeleton + head + texture + tree) in comparison to the node of tree, but the bird belongs to the tree (in other words, the bird is associated with the tree), this node may be linked to the parent node of the tree to describe this relationship. The benefit is that if a user wants to delete the tree, any element associated with the tree (in the present case, bird) will be deleted with the tree. However, the node of bird may be directly linked to the root.

It is to be noted that in the specification the term "enrich" means improving the quality of intermediate data and/or improving the quality of information (features) of raw data (i.e. original data to be processed by the offloading task) if any. For example, the addition of a bird does not improve the quality of intermediate data. On the other hand, the addition of a head of the cow improves the quality of intermediate data because the cow will become more specific. It is noted that as will be explained below in an example of sound data creation, improving the quality of intermediate data or raw data may not necessarily be performed by adding some data.

This tree-structure file is easy to modify and reuse. If other offloading task requests a black cow, the cloud just needs to change the white texture which is stored in one detailed element node to the black texture.

A picture-processing offloading task will be explained to illustrate the tree-structure file. A picture is sent from the mobile device to the cloud for processing. First, a root

is created to initialize the tree, and an additional element node is created to store the original picture. Then, tone of the picture is changed, and these changed data (i.e. subsection data of intermediate data) will be stored in a detailed element node which links to the additional element node. Then, if an image filter is used to process the picture, these processed data (i.e. subsection data of intermediate data) will also be stored in a detailed element node. Finally, when a border or text on the picture is added, it will be stored in a new additional element node which links to the additional element node of the original picture because the child additional element node (border or text) does not enrich current intermediate data (original picture + tone + filter) in comparison to the parent additional element node (original picture).

As will be appreciated by the skilled person in the art, the tree-structure method is applicable to various offloading tasks (e.g. sound data creation, marker detection, face recognition), other than those explained above, as long as intermediate data that is being created during the offloading task is dividable.

For an example of sound data creation, assume that a song is produced as an offloading task. The user device (mobile device 4) records one or more phases or sentences of a user (i.e. raw data) and sends it to the cloud for processing in order to improve the quality and add background music. One way may be to first detect a sound wave with only noise and delete it. Original voice data is stored in an additional element node directly linked to the root. The noise sound wave location (time coordinate) is stored in a detailed element node linked to the additional element node. Instead, the total sound wave without noise may be stored in the detailed element node. Data stored in detailed element nodes will have specific features that the original voice data does not have. The background music is stored in an additional element node directly linked to the root. It can be said that noise sound wave location (or sound wave without noise) stored in the detailed element node “enriches”, i.e. improves the quality of (information or features of) original voice data stored in its parent node.

For a marker-detection task, an original image (raw data) to be processed is supplied from the user device (mobile device 4) to the cloud. The original image is stored in an additional element node directly linked to the root. At the cloud several procedures are executed, including converting the original image to a gray-scale image, extracting contours from the gray-scale image, filtering out useless contours, and recognizing the marker. In each procedure, the user device will receive intermediate data (i.e. intermediate results), for example the gray-scale image and contours' coordinates. These data "enrich" (i.e. improve the quality of) information or features of the original image in the meaning that these data include specific features which the original image does not have; they are stored in detailed element nodes. In one example, the gray-scale image is stored in a first detailed element node which has a parent node in which the original data is stored; the contours' coordinates are stored in a second detailed element node, the parent node of which is the first detailed element node. The second detailed element node stores data that "enriches" data stored in the first detailed element node.

A face-detection task and an associated creation of a tree-structure file are performed in a similar manner as the marker-detection task.

In one embodiment, an additional element node may have one or more branches which contain only detailed element nodes to enrich its details (in case of the 3D cow model, the node of white texture is linked to the node of skeleton). Any new detailed element nodes to this additional element node will be linked to these branches as their leaf nodes, or linked to this additional element node to create new branches. The term "branch" refers to a sub-tree. The term "leaf node" refers to a node without child node.

Figure 6 shows a flow diagram illustrating a depth first search method that is carried out by the mobile device 4 for reconstructing and/or reviewing the tree-structure file. The depth first search method is a recursion which is the process of repeating items in a self-similar way. The method starts to search the tree from the root. In step 22, the data in a current node is added to a result file and the process proceeds to step 23. If the current node has detailed element nodes in its sub-tree, the process

proceeds to step 24. In step 24, all the detailed element nodes will be enumerated, and the process proceeds to step 25. In step 25, the recursion starts from one selected detailed element node. After finishing the recursion, the process proceeds to step 26. In step 23, if the current node does not have any other detailed element node, the process also proceeds to step 26.

In step 26, if the current node has additional element nodes in its sub-tree, the process proceeds to step 27. All the additional element nodes will be enumerated in step 27. And in step 28, a recursion is started from one selected additional element node. After finishing the recursion, the process proceeds to an end. In step 26, if the current node does not have any other additional element node, the process also proceeds to an end.

As will be appreciated by the skilled person in the art, reconstructing and/or reviewing tree-structure result file is not limited to the method explained above. For example, the method may first search additional element nodes, instead of detailed element nodes.

In one embodiment, an additional element node and its sub-tree may construct a complete sub-file. For example, in figure 5, the whole tree may construct a white cow with a detailed head and a tree with birds. The right two nodes construct a tree with birds, which represents a sub-file.

As mentioned above, the log file that is created/updated by the cloud 5 and delivered to the mobile device 4 includes information on the tree-structure file. According to the present invention, the log file records information on the subsection data's location in the tree and on the change (i.e. update) of the tree-structure file during the offloading including when and where to insert a new node in order that the user device (mobile device 4) reconstructs the tree-structure file. In one embodiment, the log file further records the transmission information including the sequence number of the transmission and/or file type (e.g. image, sound) of the transmitted (increment) data.

In case where increment data is equal to each subsection, the log file on the cloud records information on the subsection data's location in the tree as the increment data's location in the tree. In case where a subsection is further dividable and increment data is one or more divisions of such a subsection, the log file on the cloud records not only information on the subsection data's location in the tree but also information on which subsection such increment data belongs to. This relationship between increment data that is supplied to the user device (mobile device 4) on the one hand and the subsection on the other hand guarantees that the user device reconstructs a tree-structure file based on the received increment data and the received log file.

As explained above, the tree-structure file and the log file may be saved in a single tree-structure file database 3-1. The tree-structure file database may have two tables. Table 1 stores the tree-structure file, with the start date and end date of the remote execution of the offloading task, transmission times, log file ID (foreign key), nodes number and a tree-structure file ID as primary key. Table 2 stores log file ID as primary key and the log file.

Applications can be developed based on the tree-structure files to improve the offloading performance. For example, an application may check the current running rate and decide the next running task on the local mobile device. When the mobile device offloads a model-processing task to the cloud, the mobile device keeps receiving the increment data to update the tree-structure file and display the current result on the mobile device. The mobile device will show a simple model becoming complex. The user of the mobile device may stop the offloading task which is being executed by the cloud at any time to review (already finished) different execution stages, and change the model processing priority via changing the view of the model in order to see a more detailed model under the current view.

When a task is offloaded to the cloud, the cloud may search the tree-structure file database. If a similar task is already done, the cloud may request the mobile device to directly find the result on the local mobile device.

When a disconnection occurs, the mobile device may search the tree stored in the tree-structure file of its own, and find the last created node in order to get the current state and the break point. Then, the data related to the offloading task that is saved in the tree-structure file of the mobile device will be sent to a local mobile application of the mobile device (see Fig. 1) to continue the unfinished part of the task.

At the same time, the cloud may keep running the task. In this embodiment, after connection recoveries, the local mobile device receives current running status of the offloading task on the cloud so as to synchronize its current running status to the cloud. The cloud then continues the offloading task.

The present invention is defined by the following claims, this section has been provided merely by way of detailed description. The subject-matter defined in the appended claims is not necessary limited to the specific features described above.

Claims

1. Method of managing intermediate data that is being generated and still in processing during an offloading computational task to a cloud, in which a user device connects to the cloud, and the cloud generates a set of data that will be delivered to the user device and form a result of the offloading task; the method comprising:

(a) a transmission handler on the user device delivers the offloading task to the cloud;

(b) the cloud analyzes whether the type of the offloading task is the one to which a tree-structure method is applicable, and in the affirmative, manages said intermediate data in the tree-structure method and, a transmission handler on the cloud transmits said intermediate data to the user device in increment data;

(c) the cloud creates a tree-structure file and a log file, the tree-structure file storing said intermediate data in the form of nodes of a tree, the log file including information on said tree-structure file, the tree-structure file and the log file both being updated during the offloading task;

(d) the transmission handler on the cloud transmits said log file to the user device; and

(e) the user device creates a tree-structure file on the user device so as to synchronize it to the tree-structure file on the cloud, based on the received intermediate data and the received log file.

2.Method according to claim 1, wherein an independent offloading application on the user device decides the offloading task.

3.Method according to claim 1, wherein the log file records information on the change of the tree-structure file including when and where to insert a new node.

4. Method according to claim 1, wherein the transmission handler on the cloud compares current intermediate data with previous intermediate data to obtain the increment data.

5. Method according to claim 1, wherein the tree-structure file has three kinds of node comprising:

(a) root, which initiates the tree;

(b) additional element node, which stores data that is independent of its parent node; and

(c) detailed element node, which stores data that enriches intermediate data in comparison to its parent node.

6. Method according to claim 5, wherein a parent node of the additional element node is (i) an addition element node or (ii) the root, and, in case of (i), the child additional element node is independent of its parent additional element node in the sense that the child additional element node does not enrich intermediate data in comparison to the parent additional element node, but is associated with the parent additional element node.

7. Method according to claim 1 to 6, wherein when a disconnection occurs, the user device judges processing status of the offloading task on the cloud at the time of disconnection based on the log file and the tree-structure file stored in the user device, and continues an unfinished part of the offloading task on the user device.

8. Method according to claim 7, wherein the cloud keeps running the offloading task after the disconnection occurs, and, after connection recovery, the user device receives current running status of the offloading task on the cloud so as to synchronize current running status of the offloading task on the user device to the received current running status of the offloading task on the cloud.

9. Method according to claim 1 to 6, wherein based on the log file and the tree-structure file stored in the user device, an application function is designed to check, on the user device, a current execution stage of the offloading task by the cloud.

10. Method according to claim 1 to 6, wherein based on the log file and the tree-structure file stored in the user device, an application function is designed to review, on the user device, different execution stages of the offloading task by the cloud.

Figure 1

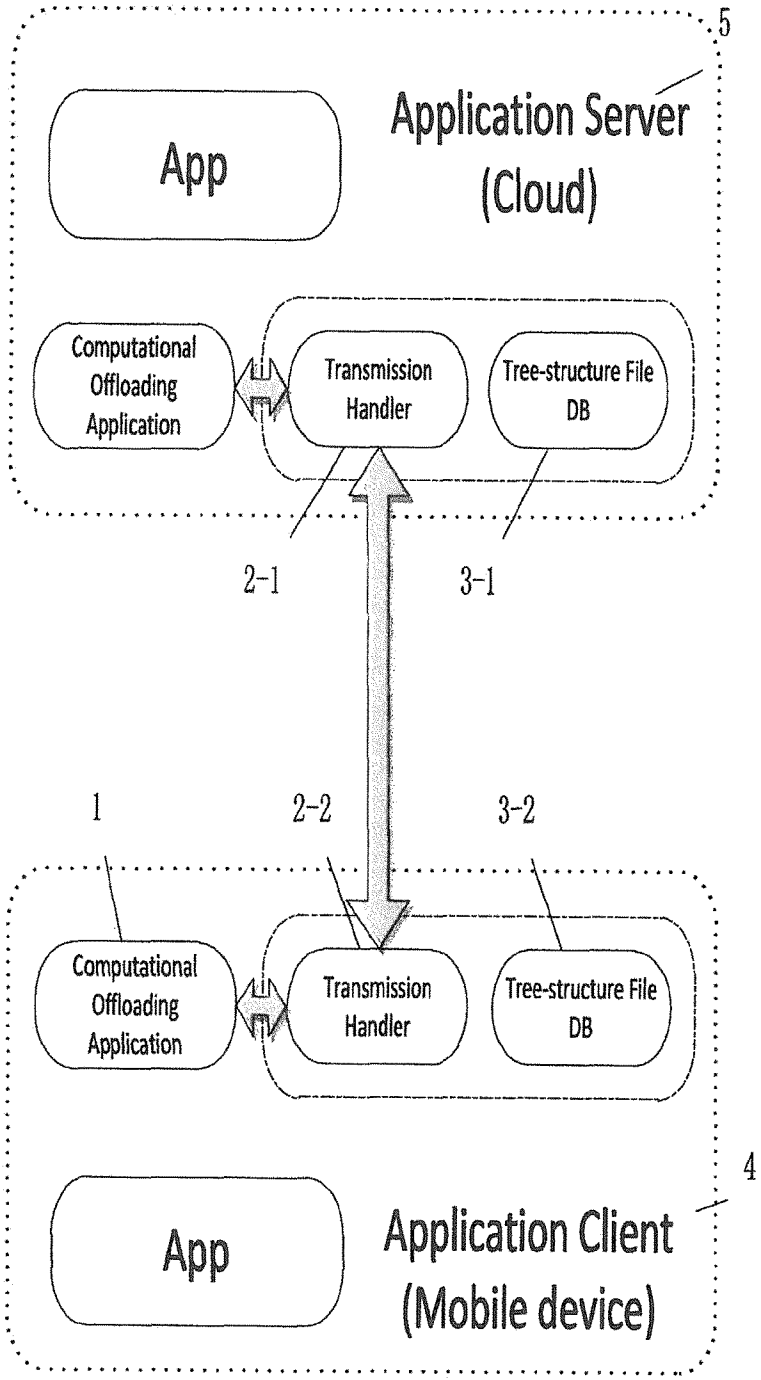

5
App
Application Server
(Cloud)
Computational
Offloading
Application
Transmission
Handler
Tree-structure File
DB
2-1
3-1
1
2-2
3-2
Computational
Offloading
Application
Transmission
Handler
Tree-structure File
DB
4
App
Application Client
(Mobile device)

Figure 2

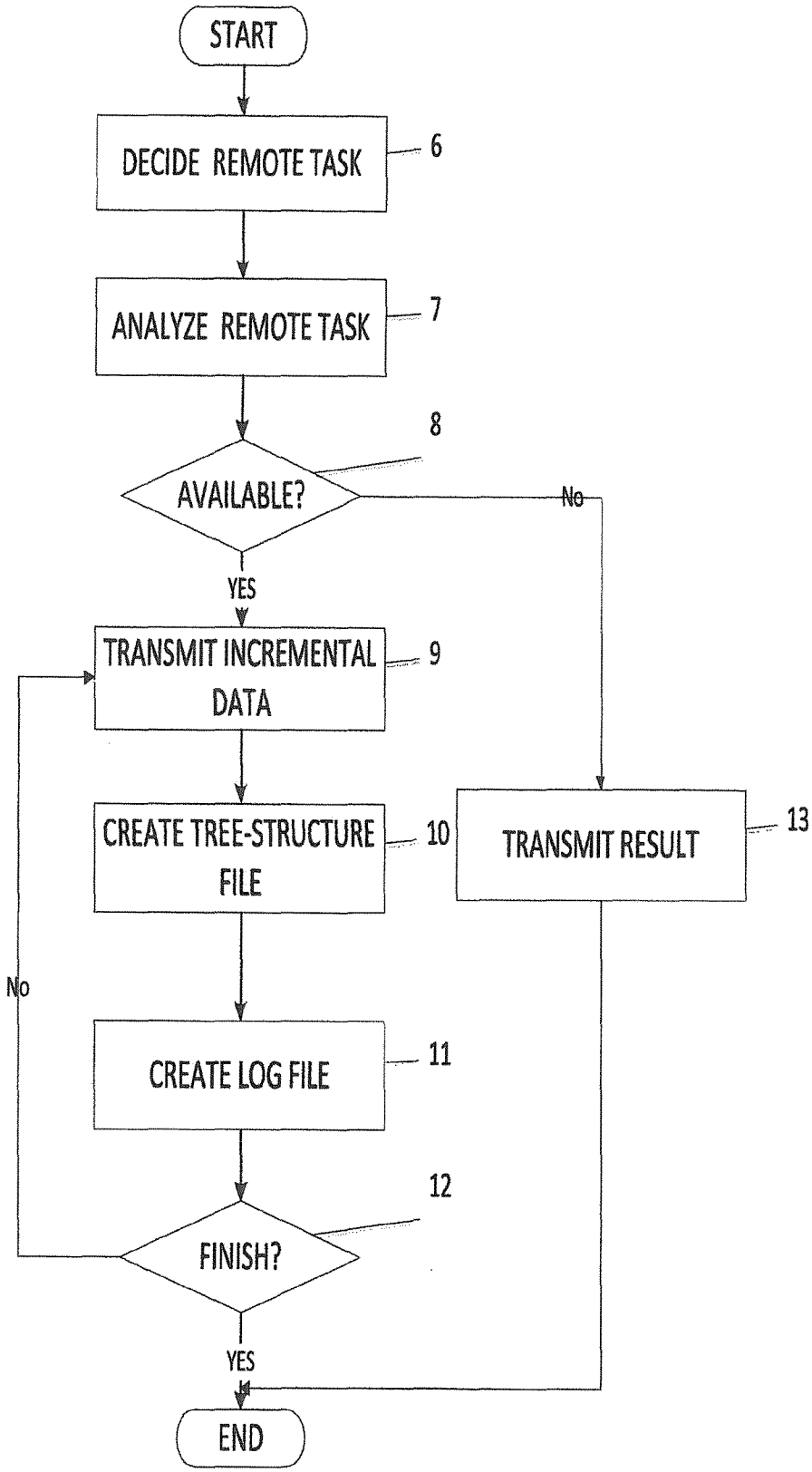

START
DECIDE REMOTE TASK
6
ANALYZE REMOTE TASK
7
8
AVAILABLE?
No
YES
TRANSMIT INCREMENTAL DATA
9
CREATE TREE-STRUCTURE FILE
10
TRANSMIT RESULT
13
No
CREATE LOG FILE
11
12
FINISH?
YES
END

Figure 3

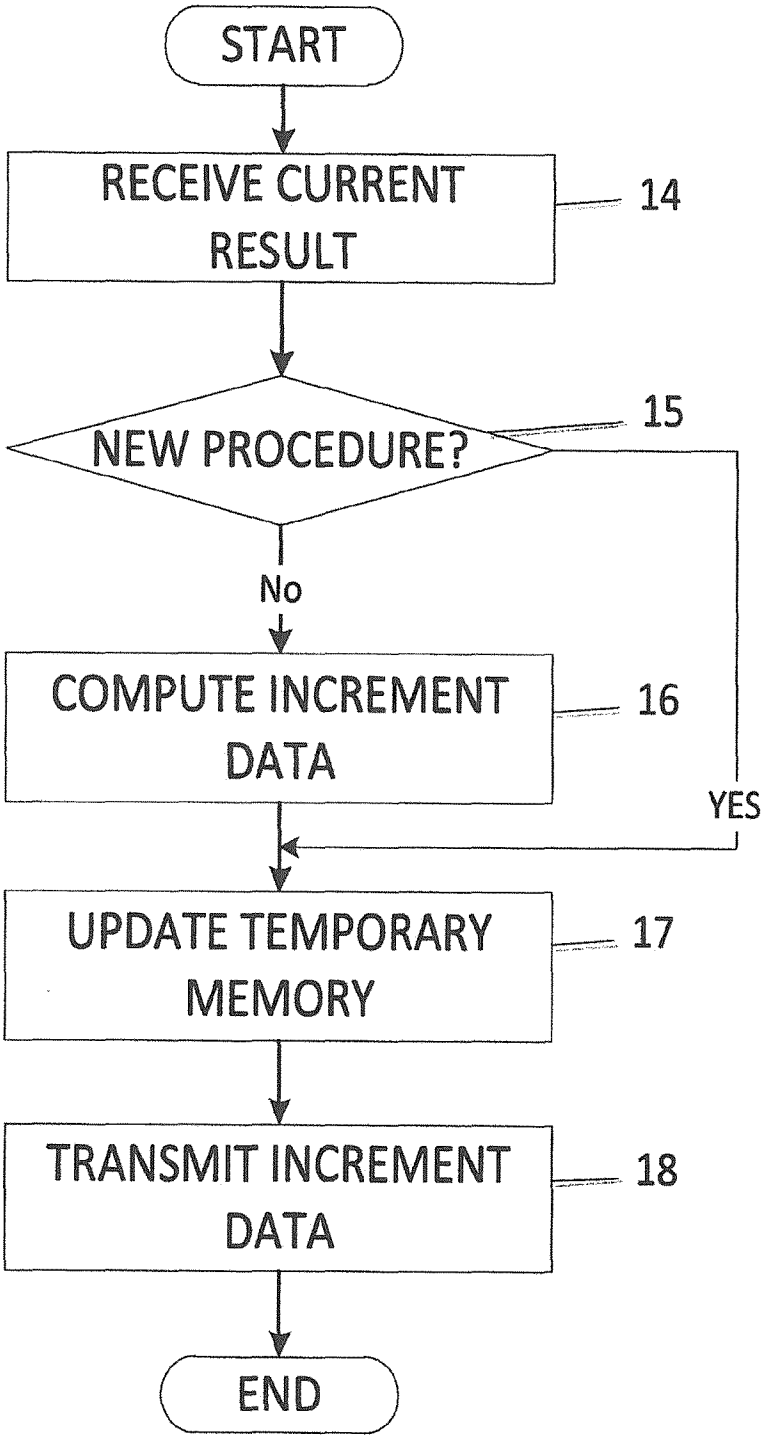

START
RECEIVE CURRENT RESULT
14
NEW PROCEDURE?
15
No
YES
COMPUTE INCREMENT DATA
16
UPDATE TEMPORARY MEMORY
17
TRANSMIT INCREMENT DATA
18
END

Figure 4

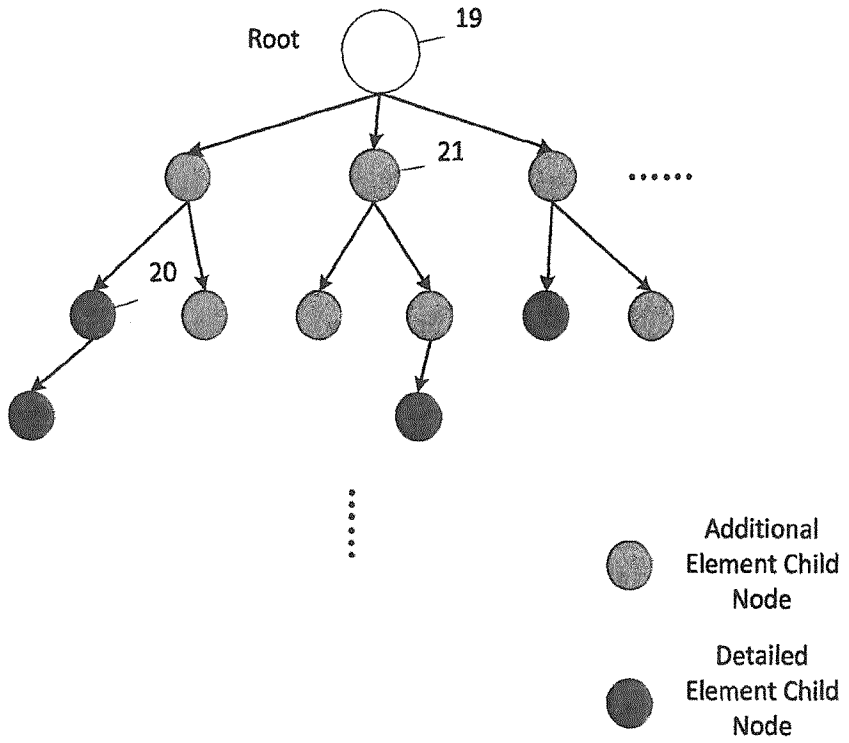


Figure 5

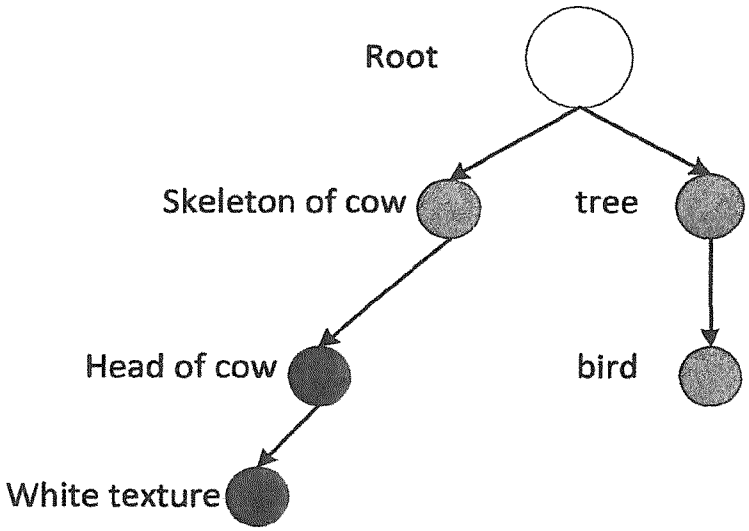
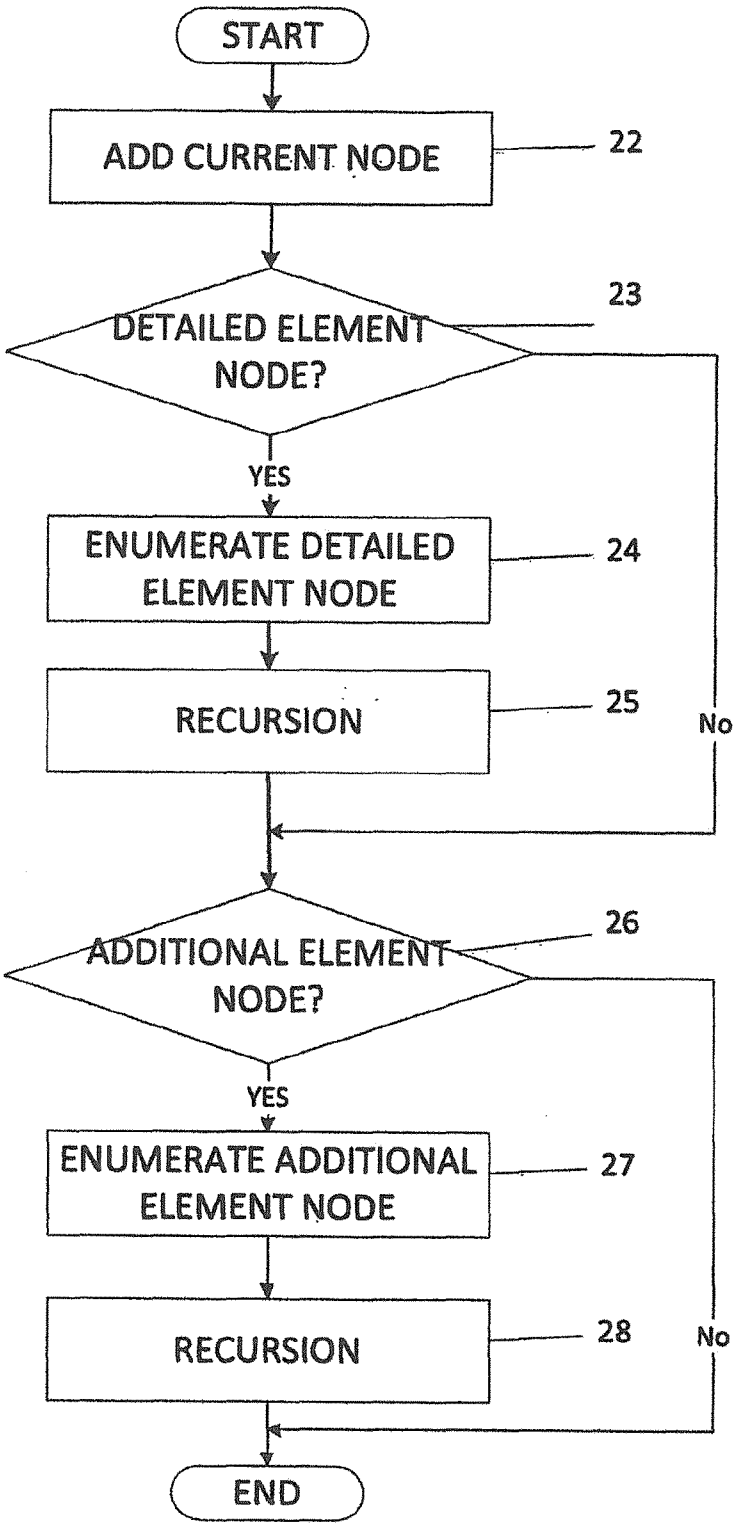


Figure 6


START
ADD CURRENT NODE
22
DETAILED ELEMENT NODE?
23
YES
ENUMERATE DETAILED ELEMENT NODE
24
RECURSION
25
No
ADDITIONAL ELEMENT NODE?
26
YES
ENUMERATE ADDITIONAL ELEMENT NODE
27
RECURSION
28
No
END

INTERNATIONAL SEARCH REPORT

International application No
PCT/EP2015/053498

A. CLASSIFICATION OF SUBJECT MATTER
INV. G06F17/30 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

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	KWON YOUNG-WOO ET AL: "Reducing the Energy Consumption of Mobile Applications Behind the Scenes", 2013 IEEE INTERNATIONAL CONFERENCE ON SOFTWARE MAINTENANCE, IEEE, 22 September 2013 (2013-09-22), pages 170-179, XP032528475, ISSN: 1063-6773, DOI: 10.1109/ICSM.2013.28 [retrieved on 2013-11-27] abstract section "I. Introduction"; page 1 section "A. Approach Overview"; page 4 section "3) Cloud Offloading Engine",; page 6; figure 7 section "B. Case Study"; page 7 ----- -/--	1-10

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

[X] See patent family annex.

* Special categories of cited documents :

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

"E" earlier 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
2 June 2015	12/06/2015

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	Gafita, Cristinel

1

Form PCT/ISA/210 (second sheet) (April 2005)

C(Continuation). DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	YOUNG-WOO KWON ET AL: "Energy-Efficient and Fault-Tolerant Distributed Mobile Execution", DISTRIBUTED COMPUTING SYSTEMS (ICDCS), 2012 IEEE 32ND INTERNATIONAL CONFERENCE ON, IEEE, 18 June 2012 (2012-06-18), pages 586-595, XP032217901, DOI: 10.1109/ICDCS.2012.75 ISBN: 978-1-4577-0295-2 abstract section "I. Introduction"; page 1 section "II. Motivating example"; page 2; figure 1 section "IV. Offloading Energy Intensive Functionality"; page 3	1-10
X	US 2012/260157 A1 (ZHU BIN BENJAMIN [US] ET AL) 11 October 2012 (2012-10-11) paragraph [0003] - paragraph [0018] paragraph [0019] - paragraphs [0029], [0047]; figures 1,2 paragraph [0057] - paragraph [0072]; figures 4,5	1-10
A	US 6 151 028 A (KUMAGAI ATSUSHI [JP] ET AL) 21 November 2000 (2000-11-21) the whole document	1-10
A	EDUARDO CUERVO ET AL: "MAUI: Making Smartphones Last Longer with Code Offload", PROCEEDINGS OF THE 8TH INTERNATIONAL CONFERENCE ON MOBILE SYSTEMS, APPLICATIONS, AND SERVICES, MOBISYS '10, 1 January 2010 (2010-01-01), page 49, XP055125674, New York, New York, USA DOI: 10.1145/1814433.1814441 ISBN: 978-1-60-558985-5 cited in the application the whole document	1-10
A	SOKOL KOSTA ET AL: "ThinkAir: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading", INFOCOM, 2012 PROCEEDINGS IEEE, IEEE, 25 March 2012 (2012-03-25), pages 945-953, XP032179206, DOI: 10.1109/INFCOM.2012.6195845 ISBN: 978-1-4673-0773-4 cited in the application the whole document	1-10

-/--

C(Continuation). DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	BYUNG-GON CHIN ET AL: "CloneCloud: Elastic Execution between Mobile Device and Cloud", EUROSYS'11 : PROCEEDINGS OF THE EUROSYS 2011 CONFERENCE ; APRIL 10 - APRIL 13, 2011, SALZBURG, AUSTRIA, ACM, NEW YORK, NY , 10 April 2011 (2011-04-10), pages 301-314, XP002692201, ISBN: 978-1-4503-0634-8 Retrieved from the Internet: URL:http://eurosys2011.cs.uni-salzburg.at/pdf/eurosys2011-chun.pdf [retrieved on 2013-02-15] cited in the application the whole document -----	1-10

INTERNATIONAL SEARCH REPORT

Information on patent family members

International application No
PCT/EP2015/053498

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
US 2012260157 A1	11-10-2012	NONE	
US 6151028 A	21-11-2000	DE 69819950 D1	08-01-2004
		DE 69819950 T2	02-09-2004
		EP 0865001 A2	16-09-1998
		JP H10255069 A	25-09-1998
		US 6151028 A	21-11-2000