

COMPUTATIONAL RESOURCE ALLOCATION

CROSS-REFERENCE TO RELATED APPLICATIONS

This application claims priority to U.S. Application Serial No. 62/431,505, filed on December 8, 2016.

5

TECHNICAL FIELD

This description relates to allocation of computational resources in a computing system.

BACKGROUND

Applications that run on computing systems require a portion of the computing system's computational resources to do so. The computing system must therefore manage allocation of its resources to applications running thereon. Some examples of resources that are allocated to applications include access to a portion of the computing system's memory, access to file data, and access to a required amount of processing power.

15 In distributed computing systems, computational resources (including data storage and processing resources) are distributed among a number of servers included in one or more clusters that work together to run data processing applications. In some examples, distributed computing systems use a centralized resource manager, which both schedules execution of applications on the computing system and
20 manages allocation of the computing system's distributed resources to the applications. One such resource manager is "Hadoop YARN."

SUMMARY

In a general aspect 1, a method is provided for requesting, from a computational resource manager, computational resources for executing a computer
25 program, having a plurality of program portions, in a computing system, the method including: storing information specifying target quantities of computational resources associated with respective ones of the program portions, where each program portion uses granted computational resources granted by the computational resource manager enabling that program portion to be executed in the computing system; and computing
30 requests, the requests being for computational resources for executing the computer program in the computing system, for submission to the computational resource

manager as a plurality of sets of requests, where each set of requests, except for an initial set of requests, would become processed by the computational resource manager only if the previously submitted one or more sets of the plurality of sets of requests were granted by the computational resource manager for use by the plurality of program portions, the computing including: forming the initial set of requests, and forming one or more additional sets of requests for processing by the computational resource manager after the initial set of requests, wherein at least one set of requests of the additional sets of requests includes a partial request for at least one program portion of the plurality of program portions that is in a least-fulfilled set of one or more program portions, wherein each program portion of the plurality of program portions is associated with a respective fulfillment ratio of its quantity of previously requested computational resources divided by its associated target quantity of computational resources, wherein a program portion is a member of the least-fulfilled set if its associated fulfillment ratio, over all of the plurality of program portions, is the minimum of the associated fulfillment ratios, and wherein the partial request requests computational resources of a quantity such that, after the partial request is granted, each program portion in the least-fulfilled set will have a fulfillment ratio that is greater than the fulfillment ratio had been for each of the one or more program portions in the least-fulfilled set before the partial request had been granted, and is still smaller than one.

Aspect 2 according to aspect 1, wherein the partial request requests a quantity of computational resources that is less than or equal to a quantity such that, after the partial request is granted, each program portion in the least-fulfilled set will have a fulfillment ratio that is still smaller than a fulfillment ratio of at least one program portion that is not a member of the least-fulfilled set.

Aspect 3 according to aspect 1 or 2, wherein the partial request requests a quantity of computational resources that is greater than or equal to a quantity such that, after the partial request is granted, each program portion in the least-fulfilled set will have a fulfillment ratio that is at least as large as a fulfillment ratio of at least one program portion that is not a member of the least-fulfilled set.

Aspect 4 according to any one of aspects 1 to 3, wherein the partial request requests a quantity of computational resources that is greater than or equal to a quantity such that, after the partial request is granted, each program portion in the

least-fulfilled set will have a fulfillment ratio that is at least as large as a fulfillment ratio of at least one program portion that is not a member of the least-fulfilled set.

Aspect 5 according to any one of aspects 1 to 4, wherein the initial set of requests includes a minimum request for each program portion, where each minimum request requests a minimum quantity of computational resources needed to execute the corresponding portion of the program.

Aspect 6 according to any one of aspects 1 to 5, wherein at least one of the program portions is associated with a fixed target quantity of computational resources and at least one of the program portions is associated with a variable target quantity of computational resources, and the minimum request for each program portion associated with a fixed target quantity of computational resources is a full request corresponding to the complete target quantity of computational resources, and the minimum request for each program portion associated with a variable target quantity of computational resources is a partial request corresponding to less than the complete target quantity of computational resources.

Aspect 7 according to any one of aspects 1 to 6, wherein the minimum request for each program portion associated with a variable target quantity of computational resources is a partial request corresponding to a single unit of computational resources, where each variable target quantity of computational resources is defined in terms of a granularity of computational resources that divides the complete target quantity of computational resources into a plurality of units of equal size.

Aspect 8 according to any one of aspects 1 to 7, wherein at least some requests in at least one set of requests correspond to quantities of computational resources of different sizes from each other.

Aspect 9 according to any one of aspects 1 to 8, wherein storing the information specifying target quantities of computational resources associated with respective ones of the plurality of program portions includes, for at least one program portion, determining its associated target quantity of computational resources based at least in part on user input.

Aspect 10 according to any one of aspects 1 to 9, wherein the computational resources are distributed over a plurality of computing nodes.

Aspect 11 according to any one of aspects 1 to 10, wherein at least some of the program portions require that their granted computational resources are obtained from a specific one or more computing nodes of the plurality of computing nodes.

Aspect 12 according to any one of aspects 1 to 11, wherein computational resources associated with each request of each set of requests of the plurality of sets of requests must be obtained from a single computing node of the plurality of computing nodes.

5 Aspect 13 according to any one of aspects 1 to 12, wherein at least some of the program portions are configured to process data items from one or more data sources and the method further includes determining the target quantities of computational resources for each of the program portions based at least in part on the data items from the one or more data sources.

10 Aspect 14 according to any one of aspects 1 to 13, wherein determining the target quantities of computational resources for each of the program portions based at least in part on the data items from the one or more data sources includes, for each program portion, adding computational resources associated with a single instance of the program portion to the target quantity of computational resources associated with
15 the program portion for each data item to be processed by the program portion.

 Aspect 15 according to any one of aspects 1 to 14, wherein each request, of each set of requests of the plurality of sets of requests, requests computational resources required by a single instance of a particular program portion of the plurality of program portions.

20 Aspect 16 according to any one of aspects 1 to 15, wherein a first program portion of the plurality of program portions is associated with a plurality of requests from the plurality of sets of requests.

 Aspect 17 according to any one of aspects 1 to 16, further including receiving, in response to the initial set of requests, an indication that one or more requests of the
25 initial set of requests was not fulfilled and aborting execution of the program based on the response.

 In a general aspect 18, a computer-readable medium stores software for requesting, from a computational resource manager, computational resources for executing a computer program, having a plurality of program portions, in a computing
30 system, the software including instructions for causing a computing system to perform the method of any of aspects 1 to 17.

 In a general aspect 19, a computing system is provided for requesting, from a computational resource manager, computational resources for executing a program, having a plurality of program portions, in the computing system, the computing

system including: a data storage system storing information specifying target quantities of computational resources associated with respective ones of the program portions; and at least one processor configured to perform the method of any of aspects 1 to 17.

5 The above aspects and below described clauses can include one or more of the following advantages.

 Among other advantages, aspects allocate computational resources of a computing system to program portions, such as data processing components, of a computer program, such as a data processing graph, such that no one of the data
10 processing components is allocated more computational resources than it needs while another program portion is starved of computational resources. Thus, the techniques described herein reduce break down of computational resources of the computing system that may be overloaded by data processing tasks while other computational resources of the computing system are not fully engaged with data processing tasks.

15 These techniques hence contribute to more efficient usage of the computational resources of the computing system executing the computer program, such as the data processing graph, and therefore enhance and ensure the proper internal functioning of the computing system. Another technical effect of the techniques described herein is the effect on the computer program, such as the data processing graph, being executed
20 on the computing system. For example, by facilitating overall balanced allocation of the computational resources of the computing system, a greater number of computer programs, such as data processing graphs, may be able to execute concurrently, and/or some of the computer programs, such as data processing graphs, may be able to start sooner by not having to wait as long to acquire the necessary computational resources
25 of the computing system needed to execute.

 Other features and advantages of the invention will become apparent from the following description, figures, and from the claims.

 This Summary is provided to introduce a selection of concepts in a simplified form that are further described below in the Detailed Description. This Summary is
30 not intended to identify key features or essential features of the claimed subject matter.

DESCRIPTION OF DRAWINGS

FIG. 1 is a block diagram of a system for allocation of distributed computational resources.

FIG. 2A is a detailed block diagram of the system for allocation of distributed
5 computational resources.

FIG. 2B is diagram of an example of a data processing graph.

FIG. 3 is a flowchart of a distributed computational resource allocation algorithm.

FIG. 4 is the block diagram of FIG. 2 after completion of the first step of the
10 computational resource allocation algorithm.

FIG. 5 is the block diagram of FIG. 2 after completion of the second step of the computational resource allocation algorithm.

FIG. 6 is the block diagram of FIG. 2 during the third step of the computational resource allocation algorithm.

FIG. 7 is the block diagram of FIG. 2 during the fifth step of the computational resource allocation algorithm.

FIG. 8 is the block diagram of FIG. 2 during a first part of the seventh step of the computational resource allocation algorithm.

FIG. 9 is the block diagram of FIG. 2 during a second part of the seventh step
20 of the computational resource allocation algorithm.

FIG. 10 is the block diagram of FIG. 2 during a third part of the seventh step of the computational resource allocation algorithm.

FIG. 11 is the block diagram of FIG. 2 during a first iteration of the ninth step of the computational resource allocation algorithm.

FIG. 12 is the block diagram of FIG. 2 during a second iteration of the ninth
25 step of the computational resource allocation algorithm.

FIG. 13 is the block diagram of FIG. 2 during a third iteration of the ninth step of the computational resource allocation algorithm.

FIG. 14 is the block diagram of FIG. 2 during a fourth iteration of the ninth
30 step of the computational resource allocation algorithm.

FIG. 15 illustrates a process flow, in reference to numbered clauses described herein, according to which computational resources can be allocated.

DESCRIPTION

FIG. 1 shows an example of a computing system 100 or data processing system 100 in which the computational resource allocation techniques can be used.

The system 100 includes a data source 102 that may include one or more
5 sources of data such as storage devices or connections to online data streams, each of which may store or provide data in any of a variety of formats (e.g., database tables, spreadsheet files, flat text files, or a native format used by a mainframe). An execution environment 104 includes an execution module 112, a resource manager 120, and a resource requesting module 122. The execution environment 104 may be
10 hosted, for example, on one or more general-purpose computers (each having a central processing unit, CPU) under the control of a suitable operating system, such as a version of the UNIX operating system. For example, the execution environment 104 may be a multiple-node parallel computing environment being hosted on a configuration of computer systems using multiple central processing units (CPUs) or
15 processor cores, either local (e.g., multiprocessor systems such as symmetric multi-processing (SMP) computers), or locally distributed (e.g., multiple processors coupled as clusters or massively parallel processing (MPP) systems, or remote, or remotely distributed (e.g., multiple processors coupled via a local area network (LAN) and/or wide-area network (WAN)), or any combination thereof.

20 Some examples of the term “computational resources”, which are allocated to one or more computer programs (such as one or more data processing graphs), as used throughout this application, include one or more of access to a portion of the computing system’s memory (such as primary memory), access to data storage (such as file data stored on a disc) of the computing system 100, and access to a required
25 amount of processing power (such as access to a CPU) of the computing system 100. Throughout this application, the term “computational resources”, which are allocated to computer programs (such as data processing graphs), can more specifically include one or more of access to a portion of memory (such as primary memory) accessible by the execution environment 104, access to data storage (such as data source 102 or
30 data storage 116) accessible by the execution environment 104, and access to a required amount of processing power of the execution environment 104 (such as access to CPU or virtual cores accessible by the execution environment 104).

A data storage system 116 is accessible to the execution environment 104 and to an optional development environment 118. The development environment 118 is a

system for developing the computer programs, such as the data processing graphs mentioned herein. The data storage system 116 may be local to the execution environment 104, for example, being stored on a storage medium connected to a computer hosting the execution environment 104 (e.g., hard drive), or may be remote to the execution environment 104, for example, being hosted on a remote system (e.g., mainframe) in communication with a computer hosting the execution environment 104, over a remote connection (e.g., provided by a cloud computing infrastructure).

For example, such an environment for developing graph-based computations is described in more detail in U.S. Publication No. 2007/0011668, titled “Managing Parameters for Graph-Based Applications,” incorporated herein by reference. A system for executing such graph-based computations is described in U.S. Patent 5,966,072, titled “EXECUTING COMPUTATIONS EXPRESSED AS GRAPHS,” incorporated herein by reference. Data processing graphs, e.g. made in accordance with this system, provide methods for getting information into and out of individual processes represented by graph components, for moving information between the processes, and for defining a running order for the processes. This system may include algorithms that choose interprocess communication methods from any available methods (for example, communication paths according to the links of the graph can use TCP/IP or UNIX domain sockets, or use shared memory to pass data between the processes).

The execution module 112 processes data from the data source 102 according to the one or more computer programs 114, such as the one or more data processing graphs 114, using computational resources allocated by the resource manager 120, to generate output data which may be stored back in the data source 102, or stored in the data storage system 116, or otherwise used. Storage devices providing the data source 102 may be local to the execution environment 104, for example, being stored on a storage medium connected to a computer hosting the execution environment 104 (e.g., hard drive 108), or may be remote to the execution environment 104, for example, being hosted on a remote system (e.g., mainframe 110) in communication with a computer hosting the execution environment 104, over a remote connection (e.g., provided by a cloud computing infrastructure). In some examples, the data source 102, includes one or more (e.g. different) forms of database systems including data that may be organized as records having values for respective fields (also called “attributes” or “columns”), including possibly null values.

The resource manager 120 schedules execution of the one or more computer programs 114, such as the one or more data processing graphs 114, on the execution environment 104 and manages allocation of the computing system's, or more specifically of the execution environment's, computational resources to the computer programs or data processing graphs 114. As is described in greater detail below, for computer programs that include program portions, such as data processing graphs that include a number data processing components, the resource requesting module 122 interacts with the resource manager 120 to allocate computational resources to the program portions, such as data processing components, such that no one program portion or data processing component is allocated more computational resources than it needs while another program portion component is starved of computational resources.

The one or more computer programs 114, such as the one or more data processing graphs 114, can be configured or programmed in a variety of ways such that different portions thereof (e.g., program portions such as data processing components of the data processing graph) are associated with specific (e.g. different) target quantities of computational resources to be allocated for use at runtime. The target quantities may be determined based on the minimum required quantity of computational resources that are needed by the respective program portion to execute all its one or more instances properly or fully, e.g., to fully process (without failure) a given amount of input data associated with or to be processed by the respective program portion and/or to execute on a given input data all data processing tasks associated with the instances, implemented by the program portion and associated with the input data without execution failure. The minimum required quantity of computational resources may be a minimum number of CPUs, virtual cores and/or portions of primary memory associated with the program portion that are required to be consumed in parallel when the instance's program portion is to be executed. These computer programs 114, as used throughout the present application, may be data processing programs that process data during runtime, such as data received from the data source 102. One example of such a data processing program is the data processing graph 114 as mentioned throughout the present application (which can also be called "dataflow graph") that can be represented by a directed graph that includes vertices (representing the data processing components or datasets) connected by directed links or data flow connections (representing flows of work elements, i.e.,

data) between the vertices. An example 225 of the data processing graph 114 is illustrated in FIG. 2B. In FIG. 2B, the data processing components are C1 to C5 (also labelled as 226, 228, 230, 232 and 234). The links or data flow connections are labelled in FIG. 2B as 226a, 228a, 228b, 230a and 232a. The program portions of the computer program as described throughout this application can also be connected by dataflow connections which provide flow of data between the program portions. The number of data processing components or program portions and the number of links or data flow connections is chosen in this application for illustrative purposes only and can each be any other number. In addition to these links or data flow connections, some data processing graphs may also have control flow connections for determining and/or adjusting of data flow between data processing components. In the data processing graphs described throughout this application, the data processing components can be the program portions and can be interrelated or interdependent according to their links or data flow connections. The data processing components can be separately granted computing resources for being executed. In other examples of the computer program, the program portions are sub-modules or other data processing entities within the computer program that are separately granted computing resources for being executed. The program portions, such as data processing components, as described throughout this application are considered “interrelated or interdependent” to the extent that the ability of the overall computer program to which they belong, such as the data processing graph, to be executed depends on the abilities of the individual program portions to be executed. In an example, if one of the program portions cannot execute, the overall computer program to which they belong to cannot execute. Such interrelated or interdependent program portions may also be dependent on each other for execution. For example, one program portion may receive data from or provide data to another program portion. Also, while the program portions, such as data processing components, are separately granted computing resources of the computing system, they may overlap, be interrelated or be interdependent in various other ways (e.g., competing for a limited supply of computing resources). The term “program portion” (or “data processing component”) as used throughout this application may be a piece of software code (also called “programming code”), wherein the computer program (or dataflow graph) may be formed by its program portions (or data processing components).

Referring to FIG. 2A, a more specific representation of the computing system 100 / data processing system 100 described above in context of FIG. 1 includes the execution environment 104, the data source 102, and the computer program or data processing graph 224 for processing data from the data source 102 using
5 computational resources of the execution environment 104.

The execution environment 104 includes the resource requesting module 122, the resource manager 120, and the execution module 112. Among other features, the execution module 112 includes the computational resources which may be distributed across and/or allocated to multiple hosts (e.g., computing clusters such as servers). In
10 FIG. 2A, there are three hosts: a first host H1 236, a second host, H2 238, and a third host, H3 240. Each host includes a finite amount or number of computational resources which taken together include the total computational resources of the execution module 112. Examples of the computational resources being managed and allocated by the resource manager 120 may include any of: a usage share of a host's
15 processor (e.g., specified as virtual cores that map to physical cores of one or more multi-core processors), a portion of volatile memory of a host (e.g., specified a quantity of the host's main memory space), a portion of non-volatile memory of a host (e.g., specified as a quantity of the host's hard disk drive storage space), or a usage share of a communication channel (e.g., a fraction of the bandwidth of a host's
20 Ethernet interface). A single unit 237 of computational resources may include one or more of multiple types of computational resources, such as a specified number of CPUs or virtual cores and a specified amount of main/primary memory or number of (virtual) main/primary memory portions. The CPU may be a processor.

In FIG. 2A, for the sake of simplifying the explanation of the computational
25 resource allocation approaches described herein, the entirety of computational resources of the hosts is formed by one or more computational resource units 237 (illustrated as squares within the hosts in FIG. 2A), which are all shown as having the same granularity (i.e., the smallest size that can be granted). However, it is noted that the computational resources are not necessarily segmented into units 237 with a fixed
30 and equal granularity but can instead be segmented into units of various granularities or portioned using other, alternative approaches. Furthermore, for the sake of simplifying the explanation of the computational resource allocation approaches described herein, all of the hosts in the execution module 112 of FIG. 2A are shown as having the same number (i.e., 16) of computational resource units 237. However,

it is noted that, in general, different hosts may have different amounts of computational resources, e.g. by having different numbers of computational resource units 237.

The resource manager 120 receives requests for the computational resources and either grants or denies the requests based on an amount of available computational resources in the hosts 236, 238, 240 of the execution module 112. One example of such a resource manager 120 includes the “Hadoop YARN” resource manager which is capable of receiving a request for computational resources for executing a computer program (or program portion) and, if sufficient computational resources are available, grants a ‘container’, e.g. one container for each instance of the program portion, with some number of units of the computational resources for use by the computer program. A container can be implemented as any suitable data structure for containing a particular quantity of computational resources, or containing any information that identifies a particular quantity of computational resources, or any combination thereof. The computer program may then execute using the computational resources in the granted container. In some examples, the computer program can request multiple containers of computational resources at one time (e.g., a number of containers for running concurrent instances 239 of a portion of the program or for running multiple portions of the program) from the resource manager 120. If sufficient resources are available for the resource manager 120 to grant all of the requested multiple containers to the computer program, it will do so. Otherwise, based on the available resources, the resource manager 120 may grant only some of the requested containers (i.e., an integer number of containers less than the total number of containers requested), or the resource manager 120 may not grant any of the requested containers. In some implementations, all of the computational resources associated with a given container are derived from a single host. Alternatively, in other implementations, a given container’s resources may be derived from multiple hosts.

As is described in greater detail below, the resource requesting module 122 interacts with the resource manager 120 in a way that ensures that a number of constraints imposed by the computer program, such as the data processing graph 224, are satisfied, and/or are satisfied in a more balanced manner.

The data processing graph 224 is a specification or example of the computer program for processing data received from the data source 102. FIG. 2B shows an

example 225 of a graph-based specification of the data processing graph 224, which includes a number of (e.g. interrelated or interdependent) data processing components including a first data processing component, C1 226, a second data processing component, C2 228, a third data processing component, C3 230, a fourth data processing component, C4 232, and a fifth data processing component, C5 234.

In general, each data processing component of the data processing graph described throughout this application may be associated with one or more constraints. Such a constraint may be a 'layout' that specifies constraints related to parallelization of instances 239 of the data processing component. One of the constraints specified in the layout may be a 'layout type' which can take one of a number of values including a static (S) layout type, a fixed-depth dynamic layout (FDL) type, and a variable-depth dynamic layout (VDL) type. The one or more constraints, such as the different layout types, may specify a target number of and/or an execution location for component instances of the data processing component that are executable or to be executed in parallel during execution of the data processing component. For each data processing component, the target number of component instances may be determined based on the associated number of data elements to be processed by the data processing component.

That also means that these constraints, such as the layouts, are directly determining a target quantity of computational resources (e.g., virtual cores or amount of primary memory) that are required to be consumed in parallel when the target number of component instances are to be executed. In other words, each of these component instances will consume computational resources, so that the target quantity of computational resources needed for a data processing component with a particular layout is directly determined by or correlated with the corresponding target number of component instances for that component. The term "target quantities of computational resources", as used throughout this application, for a particular program portion, such as a particular data processing component, may be equal to the product of the target number of instances of the program portion times the quantity of computational resources needed by each of the instances of the program portion.

Very generally, if one of the data processing components described throughout this application has a static layout type then this data processing component has a predefined, fixed target number of component instances 239 that are required to execute on the execution module 112 for the data processing graph 224 to

successfully run (e.g. execute all associated data processing tasks without failure).

The data processing component with a static layout type is also required to execute its component instances 239 on a predefined set of one or more hosts 236, 238, 240 of the execution module 112.

5 A potential advantage of the static layout type is locality of the computation. Locality refers to the ability to locate certain computational resources for the data processing component (e.g., a core of a host on which an instance will execute) on the same host as other computational resources (e.g., memory of the host in which data for the instance is stored). A dynamic layout type (FDL or VDL) may assign an
10 instance to execute on a different host from the host that stores data to be operated on by that instance, which may provide increased flexibility, but only by trading off reduced locality. Though, if locality is not critical to a particular computation, this trade-off may be worthwhile.

 If one of the data processing components described throughout this application
15 has an FDL type then this data processing component has a predefined, fixed target number of component instances 239 that are required to execute on the execution module 112 for the data processing graph 224 to successfully run (e.g. execute all associated data processing tasks without failure). There is no restriction as to where (i.e., on which hosts 236, 238, 240 of the execution module 112) the component
20 instances of the data processing component with the FDL type execute.

 If one of the data processing components described throughout this application has a VDL type then this data processing component has a dynamically determined target number of component instances 239 (which may be determined at runtime), where execution of at least one component instance on the execution module 112 is
25 required for the data processing graph 224 to successfully run (e.g. execute all associated data processing tasks without failure). There is no restriction as to where (i.e., on which hosts 236, 238, 240 of the execution module 112) the component instances of the data processing component with the VDL type execute.

 A potential advantage of the dynamic layout type (FDL or VDL) is the
30 flexibility of a computation being able to start even if there are no computational resources available on a particular (e.g., local) host 236, 238, 240, as long as there are computational resources available on some host 236, 238, 240 in a cluster. Another potential advantage is, if a computation fails due to failure of one or more hosts 236,

238, 240, the computation may be able to restart on a different set of hosts 236, 238, 240.

A potential advantage of the VDL type over the FDL type is, even if there is not as high a quantity of computational resources as desired, the computation is still
5 able to start. A potential advantage of the FDL type over the VDL type is the ability to fix the degree of parallelism (i.e., number of component instances) in order to limit the startup time delay that would be incurred by waiting for requests for a larger degree of parallelism to be fulfilled.

In the exemplary data processing graph 224, 225 of FIGs. 2A and 2B, the
10 exemplary first component 226 has a static layout type and is required to execute two component instances 239 (executed in the same or different computational resource units 237) on the first host 236 of the execution module 112. The second component 228 exemplary has an FDL type and is required to run three component instances 239 on the execution module 112 with no restriction as to where (i.e., on which hosts 236,
15 238, 240 of the execution module 112) the component instances are run. The third component 230, the fourth component 232, and the fifth component 234 all exemplary have VDL types and therefore are only required to execute a single component instance on the execution module 112 for the data processing graph to execute, though additional component instances may be executed if sufficient computational resources
20 are available on the execution module 112. There is no restriction as to where (i.e., on which hosts 236, 238, 240 of the execution module 112) the component instances of the third component 230, the fourth component 232, and the fifth component 234 execute.

Each of the data processing components 226, 228, 230, 232, 234 in the data
25 processing graph 224, 225 is configured to process data from the data source 102, such as data elements (e.g., blocks, files, or records) from datasets in the data source 102. In some examples, a block is a portion of data in a file that includes one or more records. In FIG. 2A, the first component 226 processes first data elements 252 from a first dataset 242, the second component, C2 228 processes second data elements 254
30 from a second dataset 244, the third component, C3 230 processes third data elements 256 from a third dataset 246, the fourth component, C4 232 processes fourth data elements 258 from a fourth dataset 248, and the fifth component, C5 234 processes fifth data elements 260 from a fifth dataset 250. In an example described below, a portion of the system 100 (e.g., the resource requesting module 122) analyzes the

datasets associated with each component to determine a target number of instances for different components, and consequently a target quantity of computational resources, based on the number of data elements. But, in other examples, the target quantity of computational resources is not solely based on a quantity of data in an associated dataset, but may instead (or also) depend on other factors, such as explicit or implicit user requirements, historical or estimated performance data, or combinations thereof. It is noted that there is no requirement that each data processing component 226, 228, 230, 232, 234 is associated with a different dataset. For example, a number of data processing components 226, 228, 230, 232, 234 may all process data from one or more of the same dataset(s).

In other words and in summary of the computing system 100 described above in context of FIGS. 1 and 2A, a computing system 100 comprises: a data source 102 that includes one or more sources of data; an execution environment 104 that includes an execution module 112, a resource manager 120, and a resource requesting module 122, wherein the execution environment 104 is hosted on one or more computers, wherein the execution module 112 is configured to process data from the data source 102 according to one or more computer programs 114, such as one or more data processing graphs 224, using computational resources of the execution environment 104 allocated by the resource manager 120 to the one or more computer programs 114, to generate output data, wherein the computer program 114 includes a plurality of (e.g. interrelated or interdependent) program portions, such as a plurality of (e.g. interrelated or interdependent) data processing components 226, 228, 230, 232, 234 of the data processing graph 224, which implement data processing tasks performed on data flowing from the data source 102 through the plurality of program portions to generate the output data, a data storage 116 that is accessible to the execution environment 104, wherein the computer program 114 and its plurality of program portions are specified by programming code and/or data structures stored in the data storage 116, wherein the plurality of program portions are connected by one or more data flow connections for data flow, such as links 226a, 228a, 228b, 230a, 232a, between the program portions, the data flow connections being specified by the programming code and/or the data structures, wherein the execution module 112 is configured to read the stored programming code and/or data structures, wherein the resource manager 120 is configured to receive requests for computational resources from the resource requesting module 122 and to either grant or deny the received

requests based on an amount of computational resources available in the execution environment 104, and optionally wherein the resource manager 120 is further configured to schedule and monitor execution of the one or more computer programs 114 by the execution module 112, and wherein the resource manager 120 is further
5 configured to allocate the computational resources of the execution environment, which are associated with the requested and granted requests, to the plurality of program portions and the resource requesting module 122 and the resource manager 120 are configured to interact with each other such that any one of the methods described in this application is executed, for example, such that each program portion
10 of the plurality of program portions has an instance of the program portion that is executed on a portion of the data from the data source 102 without execution failure to generate the output data, such that all data processing tasks implemented by the plurality of program portions are executed on the data from the data source 102 without execution failure to generated the output data, or such that a given amount of
15 data from the data source 102 to be processed by the computer program is fully processed (e.g., without execution failure) by the computer program to generated the output data.

A “component instance” of a data processing component, or more generally an “instance” of a program portion, as used herein throughout this application is an
20 executable copy of the data processing component, or program portion, currently consuming computational resources, such as being in a portion of the primary memory.

1 Resource Allocation

Given the layout constraints associated with the components in the data
25 processing graph 224, the exemplary data processing graph 224 can only successfully run or run at all if two component instances of the first data processing component 226 are executed on the first host 236 of the execution module 112, three component instances of the second data processing component 228 are executed on the execution module 112, and at least one component instance of each of the third data processing
30 component 230, the fourth data processing component 232, and the fifth data processing component 234 are executed on the execution module 112. A successful run of the data processing graph may be characterized in that all component instances

are executed on the data from the data source 102 without execution failure to generate the output data.

If the resource requesting module 122 has knowledge of the computational resources required by a total number of component instances of the data processing components of the data processing graph 224 and the computational resources available in the execution module 112, then the task of allocating the computational resources for the data processing graph 224 via the resource manager 120 could be performed using a solution to a packing optimization problem.

However, it is generally not the case that resource requesting module 122 has knowledge of the computational resources available in the execution module 112. Instead, the resource requesting module 122 requests computational resources from the resource manager 120 and waits to see if the requested computational resources are granted. Due to the lack of knowledge of the computational resources available in the execution module 112, there is a potential that the resource requesting module 122 will over-allocate the resource requirements of certain VDL type data processing components of the data processing graph 224 and in doing so, exhaust the computational resources available in the execution module 112 before any computational resources are allocated to other VDL type data processing components of the data processing graph 224 (i.e., starvation), or waste computational resources of over-allocated data processing components by creating a bottleneck with under-allocated data processing components (i.e., unbalanced execution).

1.1 Computational Resource Starvation

For example, in one approach to allocating computational resources for the data processing graph 224, the resource requesting module 122 first requests the computing resources required for the first, static layout type component 226 and the second, FDL type component 228. In this example, the resource manager 120 is able to allocate the required computational resources for the first component 226 and the second component 228.

The resource requesting module 122 then requests computational resources for executing the target number of instances (4 instances) of the third, VDL type component 230 on the execution module 112 (rather than only requesting enough computational resources to execute the required, single instance of the third component 230 on the execution module 112). The resource manager 120 is able to

allocate the requested computational resources for the 4 instances of the third component 230. Next, the resource requesting module 122 requests the computational resources required to execute a single instance of the fourth component 232. In this example, the resource manager 120 indicates that there are insufficient resources for allocating the computational resources required to execute a single instance of the fourth component 232 on the execution module 112.

In the above example, by over-allocating the optional target computational resources for the third component 230, the resource requesting module 122 has essentially starved the fourth component 232 and the fifth component 234 of the data processing graph 224 of any computational resources. As a result, the fourth component 232 and the fifth component 234 are unable to execute even a single component instance on the execution module 112. Since the data processing graph 224 requires execution of at least one instance of all of its VDL type components on the computing system 112 to successfully run, the data processing graph is unable to successfully run. Whereas, if the third component 230 were only allocated computational resources for a single instance, there would have been enough for a single instance of each of the fourth component 232 and fifth component 234. In this sense, the third component 230 was over-allocated computational resources.

1.2 Computational Resource Requesting Algorithm

Referring to FIG. 3, a resource requesting algorithm is implemented by the computing system 100, such as by the resource requesting module 122, and is described below for the data processing graph as an example of the computer program and for the data processing components as examples of the program portions. The algorithm avoids the above-described starvation of data processing components, and fairly and properly allocates computational resources to VDL type components in a balanced manner. The algorithm accepts the data processing components 362 of the data processing graph 224 and the datasets 364 from the data source 102 as input and processes the input to allocate computational resources to the data processing components 362 for processing data from the datasets 364, including fairly and properly allocating computational resources to VDL type components 230, 232, 234 and avoiding starving VDL type components 230, 232, 234 of computational resources. One or more pre-processing steps before a first step 370 may optionally be included in the algorithm and may comprise identifying (e.g., based on metadata

associated with each of the data processing components) any static, FDL and VDL data processing components in the accepted data processing components 362 of the data processing graph 224.

In the first step 370, each of the data processing components 362 is analyzed
5 to determine an amount of computational resources 372 required to execute a component instance of the data processing component on the execution module 112. That is, each of the data processing components 362 is analyzed to determine the amount of computational resources 372 required by the programming code and/or the data structures defining the data processing component for (full) execution of one
10 component instance 239 of the data processing component. For each static layout type data processing component and each FDL type data processing component, the target number of component instances to be executed or executable during execution of the data processing component may be determined which yields the target quantity of computational resources required for execution of each of these data processing
15 components. In a second step 366, for each VDL type data processing component (e.g., data processing components 230, 232, 234 in FIGS. 2A, 2B), the amount of data or number of data items of the dataset(s) associated with or to be processed by the data processing component is used to determine a target number of component instances 239, 368 (sometimes referred to as 'plys') associated with or required by the
20 data processing component, the target number of component instances to be executed or executable during execution of the data processing component may be determined, which yields the target quantity of computational resources required for execution of each of these data processing components. In some examples, the target number of instances 368 is the same as the number of data items of the datasets(s) associated
25 with the component.

In a third step 374, for each required component instance of each static layout type component 226 among the data processing components 362, the resource requesting module 122 requests a 'container' including the required computational resources from the required host(s) H1-H3 for the component instance from the
30 resource manager 120. If the requests are successful or fulfilled, the resource manager 120 responds to each of the requests with a container including the required computational resources. At a fourth step 376, if any of the requests sent to the resource manager 120 in the third step 374 are not fulfilled (e.g., if the resource manager 120 does not respond within a predetermined time interval), the algorithm

aborts because there are insufficient resources available in the execution module 112 for executing the data processing graph 224. That is, if any of the requests is not fulfilled, then the data processing graph cannot run. If all of the requests sent to the resource manager 120 in the third step 374 are fulfilled, the algorithm proceeds to a
5 fifth step 378.

In the fifth step 378, for each required component instance of each FDL type component 228 among the data processing components 362, the resource requesting module 122 requests a container including the required computational resources for the component instance from the resource manager 120. As is noted above, FDL type
10 components can execute using computational resources from any of the hosts of the execution module 112. In a sixth step 380, if any of the requests sent to the resource manager 120 in the fifth step 378 are not fulfilled, the algorithm aborts because there are insufficient computational resources available in the execution module 112 for executing the data processing graph 224. That is, if any of the requests is not fulfilled,
15 then the data processing graph cannot run. If all of the requests sent to the resource manager 120 in the fifth step 378 are fulfilled, the algorithm proceeds to a seventh step 382.

In the seventh step 382, for each VDL type component 230, 232, 234 among the data processing components 362, the resource requesting module 122 requests a
20 container including the required computational resources for a single component instance of the component from the resource manager 120. In an eighth step 384 if any of the requests sent to the resource manager 120 in the seventh step 382 are not fulfilled, the algorithm aborts because there are insufficient computational resources available in the execution module 112 for executing the data processing graph 224. If
25 all of the requests sent to the resource manager 120 in the seventh step 382 are fulfilled, sufficient computational resources are allocated to ensure that the data processing graph 224 can execute. In particular, upon successful completion of the seventh step 382, the algorithm has ensured that the required computational resources for any static layout type components and any FDL type components are allocated,
30 and that at least some computational resources are allocated to each of the VDL type components. The algorithm then proceeds to a ninth step 386.

In the ninth step 386, all of the VDL type components 230, 232, 234 among the data processing components 362 are ranked according to their fulfillment ratio, where the term 'fulfillment ratio' is a percentage of the respective component's target

quantity of computational resources that have been allocated to the respective component. In particular, the fulfillment ratio is the quantity of previously requested (and assumed granted) computational resources ‘# granted’ divided by its associated target quantity of computational resources ‘# target’. For example, if a VDL type component has a target set to run ten component instances of itself, and has had two component instances previously requested (and granted), that component is twenty percent fulfilled, i.e. has a fulfillment ratio of twenty percent. Note that, in this example, each instance of a particular component is assumed to require the same quantity of computational resources (specified as a number of containers), but in other examples, different instances of the same component may require different quantities of computational resources. The number of containers granted then determines the number of component instances that can be executed at runtime.

Once the VDL type components are ranked according to their fulfillment ratios, the least-fulfilled VDL type component (referred to as component A) and the second least-fulfilled component (referred to as component B) are identified. For a tie in identifying the least-fulfilled component, the following can be performed for each of the least-fulfilled components. A tie in identifying the second least-fulfilled component does not matter, since only its fulfillment ratio will be used. For the least-fulfilled component, the resource requesting module 122 requests one or more containers for a number, N , of containers for the least-fulfilled component that, if granted, that ‘partial request’ would result in the least-fulfilled component being more fulfilled than the second least-fulfilled component. In some examples, N is determined according to the following equation:

$$N = \left\lceil \left(\frac{\# \text{ granted to B}}{\# \text{ target for B}} - \frac{\# \text{ granted to A}}{\# \text{ target for A}} \right) * (\# \text{ target for A}) + 1 \right\rceil$$

If a tie occurs because all components are completely fulfilled (i.e., fulfillment ratio of 1), then the algorithm is complete.

In a tenth step 388, if any of the containers requested by the resource manager 120 in the ninth step 386 are not fulfilled, the algorithm is complete. If all of the containers requested by the resource manager 120 in the ninth step 386 are fulfilled, the algorithm loops back to the ninth step 386. The ninth step 386 and the tenth step 388 repeatedly run until all of the data processing components 362 are completely

fulfilled, i.e. have a fulfillment ratio of 100%, or a request for containers for any of the VDL type components is not fulfilled.

In other exemplary implementations, some of the steps of the resource requesting algorithm can be performed in a different order or some steps can be eliminated. For example, requests of steps 374, 378, and 382 can be performed in a different order, or combined into a single request.

In some exemplary implementations, the algorithm above formulates all of the requests for containers before any of the requests are submitted to the resource manager 120. Once formulated all of the requests are submitted together, e.g. all at once, and are processed in order by the resource manager 120, according to the order in which the requests would have been submitted using the above algorithm. For example, the requests may be associated with priorities or include priority tags specifying the priorities (e.g. metadata specifying a primary priority, a secondary priority and a tertiary priority etc.), which cause that the requests are to be assessed for grant by the resource manager 120 in the order given by the priorities. In other words, the priorities may define an assessment constraint, to be accessed and used by the resource manager 120, defining the order according to which the requests are to be assessed for grant by the resource manager 120, wherein according to the assessment constraint the requests only become assessed for grant by the resource manager 120 if all higher prioritized and/or previously submitted one or more requests of the requests were granted by the computational resource manager. For example, the resource manager 120 may first access the priority tags included in all received requests. Then, the resource manager 120 may identify an order of these requests according to the priorities given by the priority tags. Then, the resource manager 120 may, request by request according to the identified order, determine for a currently considered request if all higher prioritized and/or previously submitted requests (if any) were granted by the resource manager 120 and only if so (otherwise the currently considered request is not even assessed for a grant of computational resources), assess whether computational resources are still available for allocation to the program portion associated with the currently considered request in the order. If such computational resources are still available, then this currently considered request becomes granted. However, if for the currently considered request it was assessed that no sufficient computational resources are available for allocation to the program portion associated with the currently considered request in the order, then the currently considered

request is not granted (i.e. is denied). Each program portion may be associated with a current fulfillment ratio, which is a degree of the respective program portion's determined target quantity of computational resources that has been or is so far deemed allocated to the respective program portion by a computational resource
5 manager module, wherein the fulfillment ratio may be updated every time an additional computational resource has been or is deemed allocated to the respective program portion (e.g. by the grant of the associated request by the resource manager 120).

10 1.3 Example

Referring to FIGs. 4-14, the algorithm of FIG. 3 is applied to the system 100 of FIG. 1 or 2 in preparation for executing the data processing graph 224. Referring to FIG. 4, in the first step 370 of the algorithm, the resource requesting module 122 analyzes the data processing components of the data processing graph 224 to
15 determine the computational resource requirements for each of the data processing components. As a result of the first step 370, the resource requesting module 122 determines that the each component instance of the first data processing component 226 requires five units 237 of computational resources, each component instance of the second data processing component 228 requires three units 237 of computational
20 resources, each component instance of the third data processing component 230 requires four units of computational resources, each component instance of the fourth data processing component 232 requires five units 237 of computational resources, and each component instance of the fifth data processing component 234 requires two units 237 of computational resources.

Referring to FIG. 5, in the second step 366 of the algorithm, the resource
25 requesting module 122 analyzes the datasets 246, 248, 250 associated with each VDL type component to determine a target number of instances for the component. In some examples, the number of data items included in a dataset associated with a component determines the target number of instances for the component. In FIG. 5,
30 the third dataset 246 associated with the third component 230 includes four data items 256. The resource requesting module 122 therefore determines that the target number of component instances for the third component 230 is equal to four. The fourth dataset 248 associated with the fourth component 232 includes two data items 258.

The resource requesting module 122 therefore determines that the target number of component instances for the fourth component 232 is equal to two. The fifth dataset 250 associated with the fourth component 234 includes ten data items 260. The resource requesting module 122 therefore determines that the target number of
5 component instances for the fifth component 234 is equal to ten.

Referring to FIG. 6, in the third step 374 of the algorithm, the resource requesting module 122 interacts with the resource manager 120 to allocate computational resources for the first, static layout type component 226. To do so, the resource requesting module 122 sends a request 690 to the resource manager 120 for
10 two containers (i.e., one for each required component instance of the component) from the first host 236, each with a size of five computational resource units 237. In FIG. 6, the resource manager 120 allocates ten computational resource units 237 on the first host 236 and responds with a message 692 indicating that it was able to allocate the requested computational resources and the request 690 is therefore
15 fulfilled. With two containers, each including five computational resource units allocated for the first component 226, the first component 226 is 100% fulfilled, as is required for the data processing graph 224 to execute. The fourth step 376 of the algorithm determines that the request 690 was fulfilled and therefore causes the algorithm to proceed to the fifth step 378 of the algorithm.

Referring to FIG. 7, in the fifth step 378 of the algorithm, the resource requesting module 122 interacts with the resource manager 120 to allocate computational resources for the second, FDL type component 228. To do so, the resource requesting module 122 sends a request 790 to the resource manager 120 for
20 three containers (i.e., one for each required component instance of the component) from any of the hosts 236, 238, 240, each with a size of three computational resource units. In FIG. 7, the resource manager 120 allocates three computational resource units 237 on the first host 236, three computational resource units on the second host 238, and three computational resource units on the third host 240. The resource manager 120 then responds with a message 792 indicating that it was able to allocate
25 the requested computational resources and the request 790 is therefore fulfilled. With three containers, each including three computational resource units 237 allocated for the second component 228, the second component 228 is 100% fulfilled, as is required for the data processing graph 224 to execute. The sixth step 380 of the
30

algorithm determines that the request 790 was fulfilled and therefore causes the algorithm to proceed to the seventh step 382 of the algorithm.

Referring to FIGs. 8-10, in the seventh step 382 of the algorithm, the resource requesting module 122 interacts with the resource manager 120 to allocate resources for a single component instance of each of the third VDL type component 230, the fourth VDL type component 232, and the fifth VDL type component 234.

In FIG. 8, the resource requesting module 122 requests allocation of computational resources for a single component instance of the third VDL type component 230 by sending a request 890 to the resource manager 120 for a single container with a size of four computational resource units 237. The resource manager 120 allocates four computational resource units 237 on the third host 240 and then responds with a message 892 indicating that it was able to allocate the requested computational resources and the request 890 is therefore fulfilled. With one container including four computational resource units allocated (out of a target number of four containers) for the third component 230, the third component 230 is 25% fulfilled.

In FIG. 9, the resource requesting module 122 requests allocation of computational resources for a single component instance of the fourth, VDL type component 232 by sending a request 990 to the resource manager 120 for a single container with a size of five computational resource units 237. The resource manager 120 allocates five computational resource units 237 on the second host 238 and then responds with a message 992 indicating that it was able to allocate the requested computational resources and the request 990 is therefore fulfilled. With one container including five computational resource units allocated (out of a target number of two containers) for the fourth component 232, the fourth component 232 is 50% fulfilled.

In FIG. 10, the resource requesting module 122 requests allocation of computational resources for a single component instance of the fifth VDL type component 234 by sending a request 1090 to the resource manager 120 for a single container with a size of two computational resource units 237. The resource manager 120 allocates two computational resource units 227 on the second host, 238 and then responds with a message 1092 indicating that it was able to allocate the requested computational resources and the request 1090 is therefore fulfilled. With one container including two computational resource units allocated (out of a target number of ten containers) for the fifth component 234, the fifth component 234 is 10% fulfilled.

The eighth step 384 of the algorithm determines that computational resources were successfully allocated for a single component instance of each of the VDL type components 230, 232, 234 and therefore causes the algorithm to proceed to the ninth step 386 of the algorithm.

5 Upon completion of the seventh step and eighth steps 382, 384 sufficient computational resources are allocated to the data processing components of the data processing graph 224 to ensure that each data processing component has at least one component instance that can be executed so that the data processing graph 224 can execute on the execution module 104. The ninth and tenth steps 386, 388 of the
10 algorithm are then repeatedly run to fairly and properly distribute the remaining computational resources to the third, fourth, and fifth components 230, 232, 234, as is illustrated in FIGs. 11-14.

Referring to FIG. 11, in the ninth step 386 of the algorithm, the resource requesting module 122 determines that the fifth component 234 is the least-fulfilled
15 component (at 10% fulfillment ratio) and that the third component 230 is the second least-fulfilled component (at 25% fulfillment ratio). The resource requesting module 122 then determines that the number, N of instances of the fifth component 234 that, if granted, would result in the fifth component 234 being more fulfilled than the third component 230 is:

$$20 \quad N = \left\lceil \left(\frac{1}{4} - \frac{1}{10} \right) * (10) + 1 \right\rceil = 2$$

The resource requesting module 122 then sends a request 1190 to the resource manager 120 for two the containers from any of the hosts, each with a size of two computational resource units. In FIG. 11, the resource manager 120 allocates two computational resource units on the second host 238 and two computational resource
25 units on the third host 240. The resource manager 120 then responds with a message 1192 indicating that it was able to allocate the requested computational resources and the request 1190 is therefore fulfilled. With three containers, each including two computational resource units allocated for the fifth component 234, the fifth component 234 is 30% fulfilled. The tenth step 388 determines that all of the
30 containers requested by the resource manager 120 in the ninth step 386 are fulfilled, and the algorithm therefore loops back to the ninth step 386.

Referring to FIG. 12, in a second iteration of the ninth step 386 of the algorithm, the resource requesting module 122 determines that the third component 230 is the least-fulfilled component (at 25% fulfillment ratio) and that the fifth component 234 is the second least-fulfilled component (at 30% fulfillment ratio). The resource requesting module 122 then determines that the number, N of instances of the third component 230 that, if granted, would result in the third component 230 being more fulfilled than the fifth component 234 is:

$$N = \left\lceil \left(\frac{3}{10} - \frac{1}{4} \right) * (4) + 1 \right\rceil = 1$$

The resource requesting module 122 then sends a request 1290 to the resource manager 120 for one container from any of the hosts with a size of four computational resource units. In FIG. 12, the resource manager 120 allocates four computational resource units on the second host 238. The resource manager 120 then responds with a message 1292 indicating that it was able to allocate the requested computational resources and the request 1290 is therefore fulfilled. With a single container including four computational resource units allocated for the third component 230, the third component 230 is 50% fulfilled. The tenth step 388 determines that all of the containers requested by the resource manager 120 in the ninth step 386 are fulfilled, and the algorithm therefore loops back to the ninth step 386.

Referring to FIG. 13, in a third iteration of the ninth step 386 of the algorithm, the resource requesting module 122 determines that the fifth component 234 is the least-fulfilled component (at 30% fulfillment ratio) and that the third component 230 (or the fourth component 232) is the second least-fulfilled component (at 50% fulfillment ratio). The resource requesting module 122 then determines that the number, N of instances of the fifth component 234 that, if granted, would result in the fifth component 234 being more fulfilled than the third component 230 is:

$$N = \left\lceil \left(\frac{2}{4} - \frac{3}{10} \right) * (10) + 1 \right\rceil = 3$$

The resource requesting module 122 then sends a request 1390 to the resource manager 120 for three containers from any of the hosts, each container with a size of two computational resource units. In FIG. 13, the resource manager 120 allocates two computational resource units on the first host 236, and four computational resource units on the third host 240. The resource manager 120 then responds with a message

1392 indicating that it was able to allocate the requested computational resources and the request 1390 is therefore fulfilled. With a three containers including two computational resource units allocated for the fifth component 234, the fifth component 234 is 60% fulfilled. The tenth step 388 determines that all of the
 5 containers requested by the resource manager 120 in the ninth step 386 are fulfilled, and the algorithm therefore loops back to the ninth step 386.

Referring to FIG. 14, in a fourth iteration of the ninth step 386 of the algorithm, the resource requesting module 122 determines that the fourth component 232 is the least-fulfilled component (at 50% fulfillment ratio) and that the third
 10 component 230 is the second least-fulfilled component (at 50% fulfillment ratio). The resource requesting module 122 then determines that the number, N , of component instances of the fourth component 232 that, if granted, would result in the fourth component 232 being more fulfilled than the third component 230 is:

$$N = \left\lfloor \left(\frac{2}{4} - \frac{1}{2} \right) * (2) + 1 \right\rfloor = 1$$

15 The resource requesting module 122 then sends a request 1490 to the resource manager 120 for one container from any of the hosts 236, 238, 240 with a size of five computational resource units 237. In FIG. 14, the resource manager 120 is unable to allocate five computational resource units 237 among the available computational resource units on any of the hosts and therefore responds with a message 1492
 20 indicating that it was unable to allocate the requested computational resources. The tenth step 388 determines that all of the containers requested by the resource manager 120 in the ninth step 386 are not fulfilled, and the algorithm is therefore “done” and exits.

It is noted that, in some examples, since data processing components may have
 25 different computational resource requirements, the algorithm may not exit or complete upon the first failed request for resource allocation but may instead attempt to allocate additional resources for the other VDL type components until none of the data processing components is able to allocate any additional computation resources.

In some examples, each data processing component is associated with a
 30 different layout, but in other examples, there may be multiple (or even all) components that share the same layout. In those latter examples, the same techniques described herein can be used by summing resources required by the multiple

components and requesting the resulting sum of resources for the multiple components to share.

Various aspects and embodiments have been described for the computational resource allocation. The present invention may also (alternatively) be described by the
5 below numbered clauses 1 to 23, which are also illustrated in Figure 15:

According to a general clause 1, described is a computer-implemented method (1000) for allocating computational resources for executing a computer program, having a plurality of program portions, in a computing system, the method comprising:

- 10 for each program portion of the plurality of program portions, determining (1001) target number of one or more instances of the program portion that are required to be executed by the computing system for the computer program to run;
- for each program portion of the plurality of program portions, analyzing (1002) the program portion to determine a target quantity of computational resources that are
- 15 required to execute the required one or more instances of the program portion, wherein at least one program portion of the plurality of program portions is of a first type and a different at least one program portion of the plurality of program portions is of a second type different from the first type;
- for each program portion of the plurality of program portions, associating (1003)
- 20 and/or maintaining the program portion with a current or updated fulfillment ratio, which is a degree of the respective program portion's determined target quantity of computational resources that has been or is so far deemed allocated to the respective program portion by a computational resource manager module;
- wherein a program portion of the plurality of program portions is a member of a least-
- 25 fulfilled set if its associated fulfillment ratio, over all program portions of the plurality of program portions, is equal to the minimum of the associated fulfillment ratios;
- submitting (1004) a plurality of requests for computational resources to the computational resource manager module, wherein the plurality of requests include:
- for each program portion of the first type, a first request, having an associated primary
- 30 priority, for the complete target quantity of computational resources determined for the program portion of the first type, and
- for each program portion of the second type, a second request, having an associated secondary priority, for less than the complete determined target quantity of computational resources determined for the program portion of the second type but

for at least the amount of computational resources associated with a single one of the required instances of the program portion of the second type; and
one or more third requests, each of the third requests having an associated tertiary priority and being submitted for one program portion of the second type that is a
5 member of the least-fulfilled set, and each third request being for computational resources of a quantity that, after this quantity of computational resources is allocated or is deemed allocated to the one program portion of the second type, each program portion of the plurality of program portions being a member in the least-fulfilled set will have a fulfillment ratio that is greater than the fulfillment ratio had been for the
10 one or more program portions being a member in the least-fulfilled set before this quantity of computational resources requested in the third request had been or is deemed allocated and is still smaller than one;
and
receiving (1005) allocations, or indications of deemed allocations, of computational
15 resources to the program portions of the plurality of program portions as requested in those one or more requests of the plurality of requests that were granted by the computational resource manager module based on the available computational resources, wherein the primary priority, secondary priority and tertiary priority define an assessment constraint according to which (i) the one or more first requests, the one
20 or more second requests and the one or more third requests are to be assessed for grant by the computational resource manager module in the order of first requests before second requests and second requests before third requests and (ii) the first, second and third requests only become assessed for grant by the computational resource manager module if all in the order higher prioritized and/or previously
25 submitted one or more requests of the plurality of requests were granted by the computational resource manager.

In a clause 2, the method according to clause 1 further comprises: determining a number of data elements or data items in each data set of a plurality of data sets stored in a data source, wherein each data set of the plurality of data sets is associated
30 with a program portion of the plurality of program portions.

In a clause 3, the method according to clause 2 further comprises: for each program portion of the plurality of program portions, determining, based on the associated determined number of data elements or data items, the target number of one or more instances of the program portion that are required to be executed by the

computing system for the computer program to run with the plurality of data sets as input data.

Clause 4 according to any one of clauses 1 to 3, wherein the at least one program portion of the first type is associated with a fixed target quantity of computational resources and the at least one program portion of the second type is associated with a variable or dynamically determined target quantity of computational resources.

Clause 5 according to any one of clauses 1 to 4, wherein the at least one program portion of the first type is associated with a fixed target number of instances and the at least one program portion of the second type is associated with a variable or dynamically determined target number of instances.

Clause 6 according to clause 4 or clause 5, wherein the dynamic determination is a determination at runtime of the computer program.

Clause 7 according to any one of clause 1 to close 6, wherein at least some requests in the plurality of requests correspond to quantities of computational resources of different sizes from each other.

Clause 8 according to any one of clause 1 to clause 7, wherein the one or more third requests each request a quantity of computational resources that is less than or equal to a quantity such that, after the third request is granted, each program portion in the least-fulfilled set will have a fulfillment ratio that is still smaller than a fulfillment ratio of at least one program portion that is not a member of the least-fulfilled set.

Clause 9 according to any one of clause 1 to clause 8, wherein the one or more third requests each request a quantity of computational resources that is greater than or equal to a quantity such that, after the third request is granted, each program portion in the least-fulfilled set will have a fulfillment ratio that is at least as large as a fulfillment ratio of at least one program portion that is not a member of the least-fulfilled set.

Clause 10 according to any one of clause 1 to clause 9, wherein the one or more third requests each request a quantity of computational resources that is greater than or equal to a quantity such that, after the third request is granted, each program portion in the least-fulfilled set will have a fulfillment ratio that is at least as large as a fulfillment ratio of at least one program portion that is not a member of the least-fulfilled set.

Clause 11 according to any one of clause 1 to clause 10, the method further comprising: storing information specifying target quantities of computational resources associated with respective ones of the program portions, where each program portion uses granted computational resources granted by the computational resource manager enabling that program portion to be executed in the computing system.

Clause 12 according to clause 11, wherein storing the information specifying target quantities of computational resources associated with respective ones of the plurality of program portions includes, for at least one program portion, determining its associated target quantity of computational resources based at least in part on user input.

Clause 13 according to any one of clause 1 to clause 12, wherein the computational resources are distributed over a plurality of computing nodes.

Clause 14 according to clause 13, wherein at least some of the one or more program portions of the first type require that their granted computational resources are obtained from a specific one or more computing nodes of the plurality of computing nodes.

Clause 15 according to any one of clause 1 to clause 14, wherein at least some of the program portions are configured to process data items from one or more data sources and the method further includes determining the target quantities of computational resources for each of the program portions based at least in part on the data items from the one or more data sources.

Clause 16 according to any one of clause 1 to clause 15, wherein the plurality of requests for computational resources are submitted all at once to the computational resource manager module, wherein each request may optionally include a priority tag specifying the associated primary, secondary or tertiary priority.

Clause 17 according to any one of clause 1 to clause 16, wherein the plurality of requests for computational resources are submitted in the order of first requests before second requests and second requests before third requests to the computational resource manager module.

Clause 18 according to any one of clause 2 to clause 17, further comprising determining the target quantities of computational resources for each of the program portions based at least in part on the data elements or data items, and, for each program portion, adding computational resources associated with a single instance of

the program portion to the target quantity of computational resources associated with the program portion for each data item or data element to be processed by the program portion.

Clause 19 according to any one of clause 1 to clause 18, including receiving,
5 in response to submitting the first or second requests, an indication that one or more requests of the first or second requests was not granted or fulfilled and aborting execution of the computer program based on the response.

Clause 20 according to any one of clauses 1 to 19, wherein the first type is a static (S) layout type or a fixed-depth dynamic layout (FDL) type, and when the
10 second type is a variable-depth dynamic layout (VDL) type.

Clause 21 according to any one of clauses 1 to 20, wherein the method further comprises: determining that the one or more requests of the plurality of requests were not granted by the computational resource managing module; and in response to the determining, outputting a notification indicating the one or more program portions
15 that are associated with the not granted one or more requests, wherein optionally this notification includes an indication of the current fulfillment ratio associated with the one or more program portions that are associated with the not granted one or more requests.

In a general clause 22, a computer-readable medium stores software for
20 requesting, from a computational resource manager, computational resources for executing a computer program, having a plurality of program portions, in a computing system, the software including instructions for causing a computer to perform the method of any of clauses 1 to 21.

In a general clause 23, a computing system is provided for requesting, from a
25 computational resource manager, computational resources for executing a computer program, having a plurality of program portions, in the computing system, the computing system including: a computer-readable medium storing software instructions; and at least one processor configured to perform the method of any of clauses 1 to 21 by using the software instructions.

30 **2 Implementations**

The computational resource allocation approaches described above can be implemented, for example, using a programmable computing system executing suitable software instructions or it can be implemented in suitable hardware such as a

field-programmable gate array (FPGA) or in some hybrid form. For example, in a programmed approach the software may include procedures in one or more computer programs that execute on one or more programmed or programmable computing system (which may be of various architectures such as distributed, client/server, or
5 grid) each including at least one processor, at least one data storage system (including volatile and/or non-volatile memory and/or storage elements), at least one user interface (for receiving input using at least one input device or port, and for providing output using at least one output device or port). The software may include one or more modules of a larger program, for example, that provides services related to the
10 design, configuration, and execution of data processing graphs. The modules of the program (e.g., elements of a data processing graph) can be implemented as data structures or other organized data conforming to a data model stored in a data repository.

The software may be stored in non-transitory form, such as being embodied in
15 a volatile or non-volatile storage medium, or any other non-transitory medium, using a physical property of the medium (e.g., surface pits and lands, magnetic domains, or electrical charge) for a period of time (e.g., the time between refresh periods of a dynamic memory device such as a dynamic RAM). In preparation for loading the instructions, the software may be provided on a tangible, non-transitory medium, such
20 as a CD-ROM or other computer-readable medium (e.g., readable by a general or special purpose computing system or device), or may be delivered (e.g., encoded in a propagated signal) over a communication medium of a network to a tangible, non-transitory medium of a computing system where it is executed. Some or all of the processing may be performed on a special purpose computer, or using special-purpose
25 hardware, such as coprocessors or field-programmable gate arrays (FPGAs) or dedicated, application-specific integrated circuits (ASICs). The processing may be implemented in a distributed manner in which different parts of the computation specified by the software are performed by different computing elements. Each such computer program is preferably stored on or downloaded to a computer-readable
30 storage medium (e.g., solid state memory or media, or magnetic or optical media) of a storage device accessible by a general or special purpose programmable computer, for configuring and operating the computer when the storage device medium is read by the computer to perform the processing described herein. The inventive system may also be considered to be implemented as a tangible, non-transitory medium,

configured with a computer program, where the medium so configured causes a computer to operate in a specific and predefined manner to perform one or more of the processing steps described herein.

Each of the optional features described in the specification can be combined
5 with any other one of the optional features and can also be combined alone, or in combination with the other optional features, with any one of the numbered aspects, clauses or claims described in this application.

A number of embodiments of the invention have been described.
Nevertheless, it is to be understood that the foregoing description is intended to
10 illustrate and not to limit the scope of the invention, which is defined by the scope of the following claims. Accordingly, other embodiments are also within the scope of the following claims. For example, various modifications may be made without departing from the scope of the invention. Additionally, some of the steps described above may be order independent, and thus can be performed in an order different
15 from that described.

What is claimed is:

1. A method for requesting, from a computational resource manager, computational resources for executing a program in a computing system, the method including:
 - storing information specifying target quantities of computational resources associated with respective ones of a plurality of program portions of the program, where each program portion uses computational resources granted by the computational resource manager enabling that program portion to be executed in the computing system; and
 - computing requests for submission to the computational resource manager as a plurality of sets of requests, where each set of requests, except for an initial set of requests, is processed by the computational resource manager only if the previously submitted one or more sets of the plurality of sets of requests were granted by the computational resource manager, the computing including:
 - forming the initial set of requests, and
 - forming one or more additional sets of requests for processing by the computational resource manager after the initial set of requests,
 - wherein at least one set of requests of the additional sets of requests includes a partial request for at least one program portion that is in a least-fulfilled set of one or more program portions,
 - wherein each program portion is associated with a respective fulfillment ratio of its quantity of previously requested computational resources divided by its associated target quantity of computational resources,
 - wherein a program portion is a member of the least-fulfilled set if its associated fulfillment ratio, over all of the plurality of program portions, is the minimum of the associated fulfillment ratios, and
 - wherein the partial request requests a quantity of computational resources that is small enough so that, after the partial request is granted, each program portion in the least-fulfilled set will have a fulfillment ratio that is greater than its fulfillment ratio had been before the partial request had been granted, and is still smaller than one.
2. The method of claim 1, wherein the partial request requests a quantity of computational resources that is small enough so that, after the partial request is granted, each program portion in the least-fulfilled set will have a fulfillment ratio that is still smaller than a

fulfillment ratio of at least one program portion outside of the least-fulfilled set.

3. The method of claim 2, wherein the partial request requests a quantity of computational resources that is large enough so that, after the partial request is granted, each program portion in the least-fulfilled set will have a fulfillment ratio that is at least as large as a fulfillment ratio of at least one program portion outside the least-fulfilled set.

4. The method of claim 1, wherein the partial request requests a quantity of computational resources that is large enough so that, after the partial request is granted, each program portion in the least-fulfilled set will have a fulfillment ratio that is at least as large as a fulfillment ratio of at least one program portion outside the least-fulfilled set.

5. The method of claim 1, wherein the initial set of requests includes a minimum request for each program portion, where each minimum request requests a minimum quantity of computational resources needed to execute the corresponding portion of the program.

6. The method of claim 5, wherein at least one of the program portions is associated with a fixed target quantity of computational resources and at least one of the program portions is associated with a variable target quantity of computational resources, and the minimum request for each program portion associated with a fixed target quantity of computational resources is a full request corresponding to the complete target quantity of computational resources, and the minimum request for each program portion associated with a variable target quantity of computational resources is a partial request corresponding to less than the complete target quantity of computational resources.

7. The method of claim 6, wherein the minimum request for each program portion associated with a variable target quantity of computational resources is a partial request corresponding to a single unit of computational resources, where each variable target quantity of computational resources is defined in terms of a granularity of computational resources that divides the complete target quantity of computational resources into a plurality of units of equal size.

8. The method of claim 1, wherein at least some requests in at least one set of requests correspond to quantities of computational resources of different sizes from each other.

9. The method of claim 1, wherein storing the information specifying target quantities of computational resources associated with respective ones of the plurality of program portions includes, for at least one program portion, determining its associated target quantity of computational resources based at least in part on user input.
10. The method of claim 1, wherein the computational resources are distributed over a plurality of computing nodes.
11. The method of claim 10, wherein at least some of the program portions require that their granted computational resources are obtained from a specific one or more computing nodes of the plurality of computing nodes.
12. The method of claim 10, wherein computational resources associated with each request of each set of requests of the plurality of sets of requests must be obtained from a single computing node of the plurality of computing nodes.
13. The method of claim 1, wherein at least some of the program portions are configured to process data items from one or more data sources and the method further includes determining the target quantities of computational resources for each of the program portions based at least in part on the data items from the one or more data sources.
14. The method of claim 13, wherein determining the target quantities of computational resources for each of the program portions based at least in part on the data items from the one or more data sources includes, for each program portion, adding computational resources associated with a single instance of the program portion to the target quantity of computational resources associated with the program portion for each data item to be processed by the program portion.
15. The method of claim 1, wherein each request, of each set of requests of the plurality of sets of requests, requests computational resources required by a single instance of a particular program portion of the plurality of program portions.
16. The method of claim 1, wherein a first program portion of the plurality of program

portions is associated with a plurality of requests from the plurality of sets of requests.

17. The method of claim 1, further including receiving, in response to the initial set of requests, an indication that one or more requests of the initial set of requests was not fulfilled and aborting execution of the program based on the response.

18. Software stored in a non-transitory form on a computer-readable medium, for requesting, from a computational resource manager, computational resources for executing a program in a computing system, the software including instructions for causing a computing system to:

store information specifying target quantities of computational resources associated with respective ones of a plurality of program portions of the program, where each program portion uses computational resources granted by the computational resource manager enabling that program portion to be executed in the computing system; and

compute requests for submission to the computational resource manager as a plurality of sets of requests, where each set of requests, except for an initial set of requests, is processed by the computational resource manager only if the previously submitted one or more sets of the plurality of sets of requests were granted by the computational resource manager, the computing including:

forming the initial set of requests, and

forming one or more additional sets of requests for processing by the computational resource manager after the initial set of requests,

wherein at least one set of requests of the additional sets of requests includes a partial request for at least one program portion that is in a least-fulfilled set of one or more program portions,

wherein each program portion is associated with a respective fulfillment ratio of its quantity of previously requested computational resources divided by its associated target quantity of computational resources,

wherein a program portion is a member of the least-fulfilled set if its associated fulfillment ratio, over all of the plurality of program portions, is the minimum of the associated fulfillment ratios, and

wherein the partial request requests a quantity of computational resources that is small enough so that, after the partial request is granted, each program portion in the least-fulfilled set will have a fulfillment ratio that is greater than its fulfillment

ratio had been before the partial request had been granted, and is still smaller than one.

19. A computing system for requesting, from a computational resource manager, computational resources for executing a program in the computing system, the computing system including:

a data storage system storing information specifying target quantities of computational resources associated with respective ones of a plurality of program portions of the program, where each program portion uses computational resources granted by the computational resource manager enabling that program portion to be executed in the computing system; and

at least one processor configured to compute requests for submission to the computational resource manager as a plurality of sets of requests, where each set of requests, except for an initial set of requests, is processed by the computational resource manager only if the previously submitted one or more sets of the plurality of sets of requests were granted by the computational resource manager, the computing including:

forming the initial set of requests, and

forming one or more additional sets of requests for processing by the computational resource manager after the initial set of requests,

wherein at least one set of requests of the additional sets of requests includes a partial request for at least one program portion that is in a least-fulfilled set of one or more program portions,

wherein each program portion is associated with a respective fulfillment ratio of its quantity of previously requested computational resources divided by its associated target quantity of computational resources,

wherein a program portion is a member of the least-fulfilled set if its associated fulfillment ratio, over all of the plurality of program portions, is the minimum of the associated fulfillment ratios, and

wherein the partial request requests a quantity of computational resources that is small enough so that, after the partial request is granted, each program portion in the least-fulfilled set will have a fulfillment ratio that is greater than its fulfillment ratio had been before the partial request had been granted, and is still smaller than one.

20. A computing system for requesting, from a computational resource manager, computational resources for executing a program in the computing system, the computing system including:

means for storing information specifying target quantities of computational resources associated with respective ones of a plurality of program portions of the program, where each program portion uses computational resources granted by the computational resource manager enabling that program portion to be executed in the computing system; and

means for computing requests for submission to the computational resource manager as a plurality of sets of requests, where each set of requests, except for an initial set of requests, is processed by the computational resource manager only if the previously submitted one or more sets of the plurality of sets of requests were granted by the computational resource manager, the computing including:

forming the initial set of requests, and

forming one or more additional sets of requests for processing by the computational resource manager after the initial set of requests,

wherein at least one set of requests of the additional sets of requests includes a partial request for at least one program portion that is in a least-fulfilled set of one or more program portions,

wherein each program portion is associated with a respective fulfillment ratio of its quantity of previously requested computational resources divided by its associated target quantity of computational resources,

wherein a program portion is a member of the least-fulfilled set if its associated fulfillment ratio, over all of the plurality of program portions, is the minimum of the associated fulfillment ratios, and

wherein the partial request requests a quantity of computational resources that is small enough so that, after the partial request is granted, each program portion in the least-fulfilled set will have a fulfillment ratio that is greater than its fulfillment ratio had been before the partial request had been granted, and is still smaller than one.