

(19) World Intellectual Property Organization
International Bureau(43) International Publication Date
11 May 2006 (11.05.2006)

PCT

(10) International Publication Number
WO 2006/050349 A2

(51) International Patent Classification: Not classified

(21) International Application Number:
PCT/US2005/039440

(22) International Filing Date: 28 October 2005 (28.10.2005)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
0423990.1 29 October 2004 (29.10.2004) GB(71) Applicant (for all designated States except US):
HEWLETT-PACKARD DEVELOPMENT COMPANY, L.P. [US/US]; 20555 S.H. 249, Houston, TX 77070 (US).

(72) Inventors; and

(75) Inventors/Applicants (for US only): **DA SILVA, Fabricio, Alves, Barbosa** [BR/BR]; c/o Hewlett-Packard Computadores, LTDA, Av Ipiranga 6681, Predio 91 A, Port Alegre, CEP-90619-900 Rio Grande do Sul (BR). **DE CARVALHO, Silvia, Regina** [BR/BR]; c/o Hewlett-Packard Computadores, LTDA, Av Ipiranga 6681, Predio 91 A, Port Alegre, CEP-90619-900 Rio Grande do Sul (BR).(74) Agent: **BERNER, Kenneth, M.**; Lowe Hauptman Gilman & Berner, LLP, 1700 Diagonal Rd., Suite 300, Alexandria, VA 22314 (US).

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

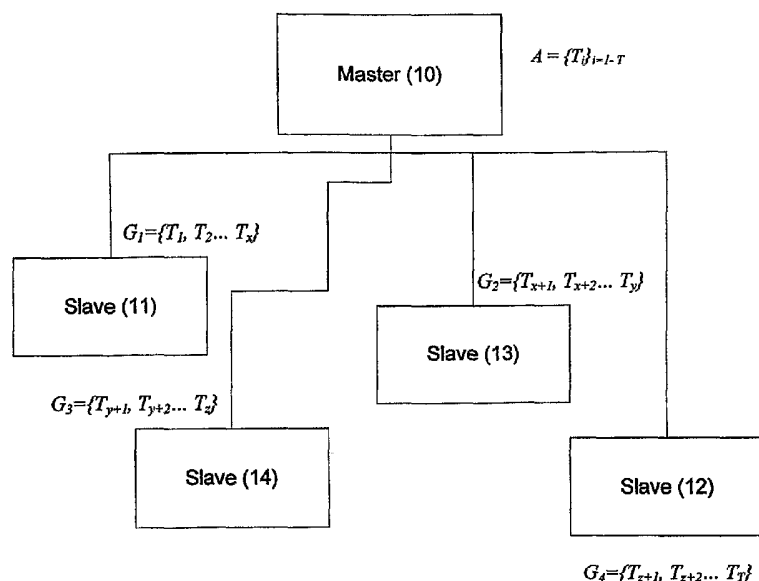
(84) Designated States (unless otherwise indicated, for every kind of regional protection available): ARIPO (BW, GH, GM, KE, LS, MW, MZ, NA, SD, SL, SZ, TZ, UG, ZM, ZW), Eurasian (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European (AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HU, IE, IS, IT, LT,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

Published:

— without international search report and to be republished upon receipt of that report

[Continued on next page]

(54) Title: METHODS AND APPARATUS FOR RUNNING APPLICATIONS ON COMPUTER GRIDS



(57) Abstract: A method of running grid applications on a grid, the grid comprising a plurality of computational units and the application comprising a plurality of tasks, the method including the steps of: estimating the task execution times on all computational units comprising the grid; grouping the tasks and assigning said groups to corresponding computational units; and, as the computational units complete execution of tasks, replicating tasks onto idle computational units in such a way that the remaining amount of computation is balanced between the computational units.



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

Methods and Apparatus for Running Applications on Computer Grids

Field of the invention

The invention relates to methods and apparatus for executing applications on computer grids. More particularly, although not exclusively, the invention relates to methods and apparatus for scheduling and running the components of applications, also known as tasks of grid-based applications, on computational units constituting a computational grid or cluster. Even more particularly, although not exclusively, the invention relates to scheduling and running tasks on heterogeneous distributed computational grids where the processing power of the node resources of the grid varies dynamically. The invention may be particular suitable for scheduling sequential independent tasks, otherwise known as Bag-of-Tasks or Parameter Sweep applications, on computational grids.

Background to the Invention

A computational grid, or more simply 'grid', can be thought of as a collection of physically distributed, heterogeneous computational units, or nodes. The physical distribution of the grid nodes may range from immediate proximity to wide geographical distribution. Grid nodes may be either heterogeneous or homogeneous with homogeneous grids differing primarily in that the nodes constituting such a grid provide essentially a uniform operating environment and computing capacity. Given the operational characteristics of grids as often being formed across administrative domains and over a wide range of hardware, homogeneous grids are considered a specific case of the general heterogeneous grid concept. The present invention contemplates both types.

In the described example, the present invention contemplates distributed networks of heterogeneous nodes which are desired to be treated as a unified computing resource.

Computational grids are usually built on top of specially designed middleware platforms known as grid platforms. Grid platforms enable the sharing, selection and aggregation of the variety resources constituting the grid. These resources which constitute the nodes of the grid can include supercomputers, servers, workstations, storage systems, desktop systems and specialized devices that may be owned and operated by different organizations.

The described embodiment of the present invention is concerned with grid applications known as Bag-of-Tasks (BoT) applications. These types of applications can be decomposed

into groups of tasks. Tasks for this type of grid application are characterized as being independent in that no communication is required between them while they are running and that there are no dependencies between tasks. That is, each task constituting an element of the grid application as a whole can be executed independently with its result contributing to the overall result of the grid-based computation. Examples of BoT applications include Monte Carlo simulations, massive searches, key breaking, image manipulation and data mining.

In this specification and the exemplary embodiments described therein, we will refer to a BoT application A as being composed of T tasks $\{T_i\}_{i=1..T}$

The amount of computation involved with each task T_i is generally predefined and may vary among the tasks A . Note that the input for each task A is one or more (input) files and the output one or more (output) files.

The present exemplary embodiment relates to clusters organized as a master-slave platform. According to this model, a master node is responsible for scheduling computation among the slave nodes and collecting the results. Other grid/cluster models are possible within the scope of the present invention, and may be capable of incorporating the execution/scheduling technique described herein with appropriate modification. For example, a further embodiment is described where the slave components or nodes are themselves clusters.

Grid platforms typically use a non-dedicated network infrastructure such as the internet for inter-node communication. In such a network environment, machine heterogeneity, long and/or variable network delays and variable processor loads and capacities are common. Since tasks belonging to a BoT application do not need to communicate with each other and can be executed independently, it is considered that BoT applications are particularly suitable for execution on such a grid infrastructure.

While heterogeneous grids have been found to be suitable for executing such applications, a significant problem is scalability and dynamic variation in node processing power. The applicants copending application No. GB0423988.5 the disclosure of which is incorporated in its entirety, is concerned with scaling and the present invention is concerned mainly with scheduling functionality to take into account dynamic load variation in the grid nodes.

In both homogeneous and heterogeneous grids, it is possible that the behavior of the grid nodes may vary over time. This can be due to extraneous loads put on the node. For example, a local user may begin using a node machine in an interactive manner while that node is carrying out an allocated sequence of task calculations for a grid application run by another user. This would have the effect of increasing the anticipated computation completion time

for that node. This may also reduce the efficacy of the task allocation technique described in patent application No. GB0423988.5 when the present embodiment of the invention is used in combination with that technique. Initial task grouping is generally based on static information whereby the maximum number of tasks to be assigned to a particular processor in the grid or

5 cluster is calculated according to the ceiling function $\left\lceil \frac{T}{P} \right\rceil$ where T is the total number of tasks to be allocated and P is the total number of processors. In the case of single-node processors, if the relative speed of the nodes changes, task distribution and allocation according to this method will be less accurate. It would therefore be desirable to develop scheduling and execution techniques which take into account dynamic variation in grid node

10 processing power.

Disclosure of the invention

In its broadest aspect, the invention provides for a method of running grid applications on a grid, the grid comprising a plurality of computational units and the application comprising a plurality of tasks, the method including the steps of: estimating the task execution times on all

15 computational units comprising the grid; grouping the tasks and assigning said groups to corresponding computational units; and, as the computational units complete execution of tasks, replicating tasks onto idle computational units in such a way that the remaining amount of computation is balanced between the computational units.

In a further aspect, the invention provides for a method of running an application on a

20 computational grid comprising a plurality of computational units, the application comprised of a plurality of tasks, the method including the steps of:

- A) grouping the tasks according to the total number of computational units and total number of tasks based on an initial determination or assumption in respect of the relative processing power of the computational units constituting the computational grid;
- 25 B) scheduling groups of tasks on computational units of the computational grid using a task queue;
- C) while there remain uncompleted tasks perform step D)
- D) when a computational unit P_i completes the execution of at least one task, perform the following steps (a) to (d):

- (a) compute the mean execution time for the completed task on computational unit P_i ;
- (b) update the task queue;
- (c) abort any still running replicas of the completed tasks;
- 5 (d) if computational unit P_i is idle perform the following steps
 - (i) if there are unfinished tasks on slower computational units then replicate the unfinished tasks on computational unit P_i ;
- E) end

10 Preferably, tasks are replicated at step (i) so that the amount of outstanding computation is balanced among the computational units.

The initial grouping of tasks may be based on a static determination of the relative processing power of the computational units.

The task queue preferably corresponds to a size ordered list of the tasks constituting the grid application.

15 Step (i) may comprise replicating a one or more tasks or an entire group onto an idle computational unit.

In an alternative embodiment, in step D) if the computational unit has not completed execution in a specified time, it is considered that that computational unit has failed or is offline and the method proceeds to step (d)(i) whereby any incomplete tasks allocated to that
20 failed or offline computational unit are replicated onto an idle computational unit.

Computational units may correspond to processors, nodes, clusters or other type of computing resources which can be considered as an grid resource aggregated or otherwise.

Preferably the task queue is ordered taking into account input files which are shared between tasks or have a degree of association.

25 The tasks are preferably grouped in step A) according to a method of scheduling the running of an application on a plurality of computational units, said application comprising a plurality of tasks, each task having at least one input file associated therewith, said method including the steps of:

aggregating said plurality of tasks into one or more groups of tasks; and

allocating each group of tasks to a computational unit, wherein the plurality of tasks are aggregated so that tasks which share one or more input file are included in the same group.

Alternatively, the tasks are preferably grouped in step A) according to a method of scheduling
5 tasks among a plurality of computing units, the method including the following steps:

I) define the number of tasks to be assigned in groups to the computing units, where P is the number of computing units;

II) compute the size of each task;

III) rank the task files in a list L in order of increasing size,

10 IV) for each group, beginning with the group with the largest number of tasks perform the following steps (a) to (e):

(i) assign the smallest unassigned task file to the group;

(j) set the task file list position index equal to 1;

15 (k) while the group is not completely populated by task files perform the following steps:

(i) if the position index plus P is less than or equal than the size of the list L, and the task file affinity between the task file at the position index and the task file at the position index +1 is less than a specified value, k then increment the position index by P;

20 otherwise increment position index by 1;

(ii) assign to the group, the task file located at position in list L

(l) Remove assigned task files from List L

(m) Increment $P = P - 1$

25 In a further aspect, the invention provides for a network adapted to operate in accordance with the method as hereinbefore defined.

In a further aspect, the invention provides for a system adapted to operate in accordance with the method as hereinbefore defined.

In yet a further aspect, the invention provides for a computer adapted to perform the method as hereinbefore defined.

- 5 In another aspect, the invention provides for a computer program adapted to perform the steps of the method as hereinbefore defined.

In another aspect, the invention provides for a data carrier adapted to store a computer program as hereinbefore defined.

- 10 The invention also provides for a master computer configured to carry out the method as hereinbefore defined.

In a further aspect, the invention provides for a computational grid adapted to operate in accordance with the method as hereinbefore defined.

Brief Description of the Drawings

- 15 The invention will now be described by way of example only, with reference to the drawings, in which:

Figure 1: illustrates an embodiment of the invention having a master-slave node configuration; and

Figure 2: illustrates a flow diagram showing the replication of tasks in accordance with an embodiment of the invention.

- 20 For the purposes of explanation, a simple execution model will be described initially in relation to a prior art technique for scheduling an application on a homogeneous grid. This will then be compared with an embodiment of the invention. The specific embodiment described herein relates to fine-grain Bag-of-Tasks applications on dedicated master-slave platform as shown in Figure 1. However, this is not to be construed as limiting and the
- 25 invention may be applied to other computing contexts with suitable modification. A further class of applications that may benefit from the invention are those composed of tasks with dependencies where sets of dependent tasks can be grouped and the groups are independent among themselves. The method may be modified slightly to group the tasks according to such dependencies.

Referring to Figure 1 the application A is composed of T homogeneous tasks. That is: $A = \{T_i\}_{i=1..T}$. The master node (10), is responsible for the organizing, scheduling, transmitting and receiving the tasks corresponding to the grid application. Referring to figure 1, each task goes through three phases during execution:

5 Initialization phase

This is the process whereby the files constituting the grid application and its data are sent from the master node (10) to the slave nodes (11-14) and the task is started. The duration of this phase is equal to t_{init} .

10 The set of files sent may include a parameter file corresponding to a specified task and an executable file which is charged with performing the computational task on the slave processor. The time in this phase includes the overhead incurred by the master node (10) to initiate a data transfer to a slave (11), for example, to initiate a TCP connection. For example, consider a task i that needs to send two files to a slave nodes before execution. The time t_{init} can then be computed-as follows:

$$15 \quad t_{init} = Lat_i + \frac{\sum_{j=1}^2 File_j}{B}$$

Where Lat_i is the overhead incurred by the master node to initiate data transfer to the slave node (11-14). $\sum File_j$ is the total size in bytes of the input files that have to be transferred to slave node s and B is the data transfer rate. For simplicity, in this example it is assumed that each task has only one separate parameter file of the same size associated with it.

20 Computation Phase

In this phase, the task processes the parameter file at the slave node (11-14) and produces an output file. The duration of this phase is equal to t_{comp} . Any additional overhead related to the reception of input files by a slave node is also included in this phase.

Completion Phase

25 During this phase, the output file is sent back to the master node (1) and the task T is terminated. The duration of this phase is t_{end} . This phase may require some processing at the master and this is mainly related to writing files at the file repository (not shown in (10)). This writing step may be deferred until disk resources are available. It is therefore considered

negligible. Thus the initialization phase of one slave can occur concurrently to the completion phase of another slave node.

The total execution time of a task is therefore:

$$t_{total} = t_{init} + t_{comp} + t_{end}$$

5 The exemplary embodiment described herein corresponds to a dedicated node cluster composed of $P+1$ homogeneous processors where $T \gg P$. The additional processor is the master node (10). Communication between the master (10) and slave (11-14) is by way of a shared link and, in this embodiment the master (10) can only send files through the network to a single slave at a given time. The communication link is full duplex. This embodiment of the
10 invention corresponds to a one-port model whereby there are at most two communication processes involving a given master, one sent and one received. The one-port embodiment discussed herein is particularly suited to LAN network connections.

A slave node (11-14) is considered to be idle when it is not involved with the execution of any of the three phases of a task. Figure 1 shows the execution of a set of tasks in a system
15 composed of four slave nodes. The initial grouping of tasks is based on static information available at the time of initialization of the application execution and reflects a snapshot of the processing power of the slave nodes.

The effective number of processors P_{eff} is defined as the maximum number of slave processors needed to run an application with no idle periods on any slave processor. Taking
20 into account the task and platform models of the particular embodiment described herein, a processor may have idle periods if:

$$t_{comp} + t_{end} < (P-1)t_{init}$$

P_{eff} is then given by the following equation:
25

$$P_{eff} = \left\lceil \frac{t_{comp} + t_{end}}{t_{init}} + 1 \right\rceil$$

The maximum number of tasks is to be executed on a given processor is:

$$M = \left\lceil \frac{T}{P} \right\rceil$$

30

Where T is the total number of tasks. For a platform with P_{eff} processors, the total execution time, or *makespan*, will be:

$$t_{makespan} = M(t_{init} + t_{comp} + t_{end}) + (P-1)t_{init}$$

The second term in the right side of this equation gives the time which is needed to start the first $(P-1)$ tasks in the other $P-1$ processors. If the platform has more processors than P_{eff} , then the overall makespan is dominated by communication times between the master and the slaves. Then:

$$t_{makespan} = MPt_{init} + t_{comp} + t_{end}$$

As there are idle periods on every processor, the following equation holds:

$$(P-1)t_{init} > (t_{comp} + t_{end})$$

This equation applies primarily to two cases:

- a. For very large platforms (P Large); and
- b. For applications with a large $\frac{t_{init}}{t_{comp}}$ ratio, such as fine-grain applications.

As described in the applicants copending patent application XXXX, t_{comp} may be increased by grouping sets of tasks sharing common input files into a larger task. By doing so, it is possible to increase the effective number of processors therefore increasing the number of slave processors that can be used effectively. The time corresponding to t_{init} should ideally not increase in the same proportion to t_{comp} . Thus, according to one form of task grouping method tasks which share one or more input files are preferably selected and scheduled so as to run on a common slave node or processor.

This may be achieved by introducing the concept of the file affinity which indicates the reduction in the amount of data that needs to be transferred to a remote node when all tasks of a group are sent to that node.

In this discussion it is assumed that the number of groups is equal to the number of nodes available. This is not however a limitation in this example and modifications to the scheduling method are viable to take into account different processor/group. For example, for some specific sets of applications/platforms, the optimal execution in terms of the makespan will use a number of groups smaller than the total number of processors. Given a set G of tasks composed of K tasks, $G=\{T_1, T_2, \dots, T_K\}$, and the set F of the Y input files needed by one or

more tasks belonging to group G , $F = \{f_1, f_2, f_3 \dots f_Y\}$, the file affinity I_{aff} is defined in one embodiment as follows:

$$I_{aff}(G) = \frac{\sum_{i=1}^Y (N_i - 1) |f_i|}{\sum_{i=1}^Y N_i |f_i|}$$

5

$|f_i|$ is the size in bytes of file f_i and N_i is the number of tasks in group G which have file f_i as an input file. $0 \leq I_{aff} < 1$. An input file affinity of zero indicates that there is no sharing of files among tasks of a group. An input file affinity close to one means that all tasks of a group have a high degree of sharing of input files.

- 10 A potential benefit of initially clustering tasks into groups for execution are that the grid scales that is, increasing the number of nodes results in a decrease in the total processing time for the grid application.

The equation above for file affinity is dominated by the combinatorial function whereby all possible pairs of tasks are considered. For large numbers of tasks, this can lead to very large numbers of combinations of task pairs. For example, there are $N(25, 5)$ ways of clustering 25
 15 tasks into 5 groups. This equates roughly to 10^{15} possible combinations. It may be therefore impractical to search exhaustively in solution space for an optimal task grouping. For this reason, a simplified heuristic may be used for determining the optimal task grouping which is based on the general file affinity equation described above.

- 20 Consider a group of tasks, each of which requires a different input file. Because there is no input file sharing, there is no file affinity between them. It is desirable to start processing them on slave nodes as soon as possible to minimize t_{init} . Therefore, the tasks are transferred to slave nodes in size order from smallest to largest with no account taken of sharing amongst input files (as this is zero).

- 25 If an application where all tasks share the same input file is considered, that input file only needs to be transferred once. This is taken into account by including the effect of file affinity. If the file affinity of two consecutive tasks (in size order) is very high, it is advantageous to assign those two tasks to the same processor instead of transferring the same set of input files

twice over the network. In the ideal situation described here, this set of files is transferred only once to each processor or node of the network.

This simplified heuristic reduces the size of the possible solution space and provides a viable method of calculating the file affinities for tasks to within a workable level of accuracy.

- 5 Taking into account file affinity, the simplified embodiment includes the following steps: Initially, for each computing unit or processor, the number of tasks to be aggregated into a group is defined for that computing unit. This is done so that the time needed for processing each group is substantially the same for each computing unit.

- 10 Then the total size of each task is calculated. Here, the size of each task corresponds to the sum of the input file sizes for the task concerned. For each group defined in the aggregation step, the tasks allocated to the group as a function of both the number of tasks determined previously and task affinity. The initial allocation step may be as follows. The reference to 'position' relates to the position of the task input file in the size-ordered list. The smallest size task, task(position) is assigned to a first group. Then the file affinity of the pair task(position) and task(position+1) in the size-ordered list is determined. If the file affinity κ is greater than a specified value, task(position+1) is assigned to the first group. If the file affinity is less than a specified value, task(position+1) is assigned to a subsequent group. This process is repeated, filling sequentially the groups in order until the group allocations determined in the initial step are populated with the size-ordered, associated tasks. This can be expressed in
- 20 pseudocode as follows:

- define the number of tasks to be assigned in groups to the computing units,
- P = the number of computing units;
 - o compute the size of each task;
 - 25 o rank the task files in a list L in order of increasing size,
 - o for each group, beginning with the group with the largest number of tasks:
 - assign the smallest unassigned task file to
 - 30 the group;
 - task file list position = 1;

- until the group is completely populated by task files do:
 - o if(position + P ≤ size of list L) and (task file affinity(task file[position], task file[position+1]) < a specified value, k) then position = position + P;
 - else position = position + 1;
 - o assign to the group, the task file at position in list L
- end do
- Remove assigned task files from List L
- P = P - 1

15 An example application of this simplified heuristic is as follows. Consider a set of tasks composed of ten task files which are to be distributed on three homogeneous slave processors. The set of input files needed by each task is described as $\{f_1, \dots, f_{10}\}$ where f_i is a real value that corresponds to the byte sum of the input files needed by task t_i . As the tasks are heterogeneous, they will share no input files and the file affinity between any pair of tasks

20 will be zero.

The 10 heterogeneous input file tasks are {20K, 35K, 44K, 80K, 102K, 110K, 200K, 300K, 400K, 450K}. Three groups of tasks are generated, one with 4 tasks and the others with 3 tasks. The simplified heuristic in the case of zero file affinity operates as follows. Each task is considered in size order. Thus, 20K is allocated to the first position of group 1. Then the 35K

25 input task is allocated to the next group following the principle that each group should minimize initial transmission or initialization time. Task 44K is allocated to the third group. Task 80K is then allocated to position two of the first group, 102K to the second position of group two and so on. This produces the group of files as follows: {20K, 80K, 200K, 400K}, {35K, 102K, 300K} and {44K, 110K, 400K}. At a first approximation this keeps the amount

30 of transmitted data similar for each group and allows the task transmission/calculation to be pipelined in a reasonably efficient manner. In a preferred embodiment, the transfer of the files

occurs in a pipelined manner, i.e.; where computation is overlapped with communication. Figure 5 illustrates the pipelined transfer of input files from a master to three slave processors. As can be seen in this example, the transfers to and from the master/nodes are staggered with the computation on the slaves being overlapped with the communication phase on one or more of the other processor nodes. This reduces t_{init} when executing a group of tasks on a slave processor.

Another example is that of 10 homogeneous tasks with ten completely homogeneous sets of input files {30K, 30K, 30K, 30K, 30K, 30K, 30K, 30K, 30K, 30K}. Again, three groups of tasks are generated, one with four tasks and the others with three. As the tasks are completely homogeneous, each pair will have a file affinity of 1. Thus, following the simplified embodiment of the heuristic, the three groups of input files will be {30K}, {30K}, and {30K}.

These two extreme examples serve to illustrate how the initial static task grouping may be performed.

The size of each task may be calculated on the basis of the byte sum of all of the input files needed to execute each task on a computing unit or grid node. The file affinity may usefully be defined as k for which an affinity of 0.5 is considered acceptable as a benchmark for grouping tasks into a specified group. Essentially, this equates to setting the minimum degree of 'association' which is necessary to consider two tasks as related or sharing input files. This ensures that the file affinity is maximized within a group. Thus sending similar sets of files to multiple processors is avoided. As noted above, if the next set of files is different enough (i.e., has a file affinity with a previously allocated task less than the minimum), that task will be located at the next processor position. Firstly, this is done so that tasks with the smallest byte sum are sent initially. Secondly, this is done to guarantee that the groups are as uniform as possible in respect of the number of bytes that need to be transmitting from the master node. Thus, at initialization of the procedure, the number of tasks is allocated to each processor based on the processing power of the processor concerned and the file affinity, and the tasks are dispatched or transferred to the processor in a pipelined way. Here, pipelined means overlapping computational and communication steps.

However, this treatment assumes an initial grouping based on static information relating to the relative processing power of the grid nodes. As noted above, the number of tasks to be assigned to each group is determined such that the time needed for processing each group is substantially the same for each computing unit. This will depend on each processors relative speed based on the average speed of the processors in the cluster. For example, if the relative

speed of a particular node processor is 1.0 compared to the average speed of the cluster nodes, the maximum number of tasks which should be assigned to that processor will be $\left\lceil \frac{T}{P} \right\rceil$.

This initial task allocation is static and based on an assumption that the relative speed of the node processors remains constant throughout the execution of the grid application.

- 5 In the case of a non-dedicated cluster environment or computational grid, this may not be true. For example, other users or unrelated processes may impose loads on one or more node processors on the grid or cluster. This will have the effect of varying the relative speeds of the node processors and thus reduce the efficacy of the initial task allocation.

10 According to one embodiment of the invention, dynamic characteristics of grid node power may be taken into account according to the following process, expressed as pseudocode and with reference to the flowchart of Figure 2:

```
group - tasks according on the basis of the relative
processing power of the nodes constituting the
computational grid and the total number of tasks to be
15 executed on the grid (20);

- schedule groups of tasks on nodes of the computational
grid using a task queue (21);

- on processor  $P_i$  completing the execution of a task (22),
do

20   o compute the mean execution time on processor  $P_i$ 
      (23);

      o update task queue (24);

      o abort any still running replicas of the completed
        tasks (25);

25   o if processor  $P_i$  is idle (26)

      ▪ if there are unfinished tasks on slower
        processors (29);
```

- replicate the unfinished tasks on processor P_i (28);

- end do (22) .

According to this embodiment, the tasks are replicated between grid nodes taking into account the dynamic rate of task completion. This implicitly takes into account dynamic variation in the processing power of the grid nodes. This method also takes into account the potential failure of a slave or grid node by dynamically asserting the failure of a machine by considering a dynamically adjustable timeout at step (22). That is, if the master node does not receive any results for a period of time equal or larger than the timeout, the machine is considered offline and its allocated tasks are replicated on an available grid node. Further, if a node fails, a whole group may be replicated. Otherwise tasks for which processing has not begun on the slowest node are replicated. This has the effect of balancing the outstanding computation among the nodes.

In a further embodiment, it is possible to consider initially grouping tasks on a grid composed of a set of clusters as opposed to a grid composed of a set of processors. In this embodiment, the number of tasks assigned to each cluster will be calculated based on the requirement that the time needed by the cluster for processing each group is the same for each cluster. As before, this will depend on the processing speed of the cluster aggregate and will depend on the internal structure of the particular cluster such as the number of processors, load from other users etc. Once the number of tasks to be assigned to each cluster is determined statically, the allocation method proceeds substantially as described above. Following this initial grouping based on static information, dynamic replication can be used to take into account cluster-level variability.

The invention is further intended to cover the task scheduling/grouping technique in its most general sense as specified in the claims regardless of the possible size of the solution space for the affinity determination. It is also noted that the described embodiment of the invention may be applied to the distribution of tasks among nodes in a grid system where the computational characteristics of such nodes may take a variety of forms. That is, node processing may take the form of numerical calculation, storage or any other form of processing which might be envisaged as part of distributed application execution. Further, embodiments of the present invention may be included in a broader scheduling system in the context of allocating information to generalized computing resources.

Although the invention has been described by way of example and with reference to particular simplified or reduced-scope embodiments it is to be understood that modification and/or improvements may be made without departing from the scope of the appended claims.

Where in the foregoing description reference has been made to integers or elements having
5 known equivalents, then such equivalents are herein incorporated as if individually set forth.

Claims

1. A method of running grid applications on a grid, the grid comprising a plurality of computational units and the application comprising a plurality of tasks, the method including the steps of: estimating the task execution times on all computational units comprising the grid; grouping the tasks and assigning said groups to corresponding computational units; and, as the computational units complete execution of tasks, replicating tasks onto idle computational units in such a way that the remaining amount of computation is balanced between the computational units.
2. A method as claimed in claim 1 wherein the tasks are placed in a task queue once they have been allocated to a computational unit.
3. A method of running an application on a computational grid comprising a plurality of computational units, the application comprised of a plurality of tasks, the method including the steps of:
 - A) grouping the tasks according to the total number of computational units and total number of tasks based on an initial determination or assumption in respect of the relative processing power of the computational units constituting the computational grid;
 - B) scheduling groups of tasks on computational units of the computational grid using a task queue;
 - C) while there remain uncompleted tasks perform step D)
 - D) when a computational unit P_i completes the execution of at least one task, perform the following steps (a) to (d):
 - (a) compute the mean execution time for the completed task on computational unit P_i ;
 - (b) update the task queue;
 - (c) abort any still running replicas of the completed tasks;
 - (d) if computational unit P_i is idle perform the following steps

(i) if there are unfinished tasks on slower computational units then replicate the unfinished tasks on computational unit P_i ;

E) end

4. A method as claimed in any of claims 1 to 3 wherein tasks are replicated so that the amount of outstanding computation is balanced among the computational units.
5. A method as claimed in any preceding claim wherein the initial grouping of tasks is based on a static determination of the relative processing power of the computational units.
6. A method as claimed in any of claims 2 to 4 wherein the task queue corresponds to a size ordered list of the tasks constituting the grid application.
7. A method as claimed in any preceding claim wherein one or more tasks or an entire group are replicated onto an idle computational unit.
8. A method as claimed in claim 3 wherein in step D) if the computational unit has not completed execution in a specified time, it is considered that that computational unit has failed or is offline and the method proceeds to step (d)(i) whereby any incomplete tasks allocated to that failed or offline computational unit are replicated onto an idle computational unit.
9. A method as claimed in claim 1 or 2 wherein if a computational unit has not completed execution in a specified time, it is considered that that computational unit has failed or is offline and any incomplete tasks allocated to that failed or offline computational unit are replicated onto an idle computational unit.
10. A method as claimed in any preceding claim wherein the computational units correspond to processors, nodes, clusters or other type of computing resources which can be considered as an grid resource, aggregated or otherwise.
11. A method as claimed in any one of claims 2 to 10 wherein the task queue is ordered taking into account input files which are shared between tasks or have a degree of association.
12. A method as claimed in any preceding claim wherein the tasks are grouped in according to a method of scheduling the running of an application on a plurality of computational units, said application comprising a plurality of tasks, each task having at least one input file associated therewith, said method including the steps of:

aggregating said plurality of tasks into one or more groups of tasks; and

allocating each group of tasks to a computational unit, wherein the plurality of tasks are aggregated so that tasks which share one or more input file are included in the same group.

13. A method as claimed in any of claims 1 to 12 wherein the tasks are grouped according to a method of scheduling tasks among a plurality of computing units, the method including the following steps:

I) define the number of tasks to be assigned in groups to the computing units, where P is the number of computing units;

II) compute the size of each task;

III) rank the task files in a list L in order of increasing size,

IV) for each group, beginning with the group with the largest number of tasks perform the following steps (a) to (e):

(i) assign the smallest unassigned task file to the group;--

(j) set the task file list position index equal to 1;

(k) while the group is not completely populated by task files perform the following steps:

(i) if the position index plus P is less than or equal than the size of the list L, and the task file affinity between the task file at the position index and the task file at the position index +1 is less than a specified value, k then increment the position index by P;

otherwise increment position index by 1;

(ii) assign to the group, the task file located at position in list L

(l) Remove assigned task files from List L

(m) Increment $P = P - 1$

14. A network adapted to operate in accordance with the method as claimed in any of claims 1 to 13.

15. A system adapted to operate in accordance with the method as claimed in any of claims 1 to 13.

16. A computer adapted to perform the method as claimed in any of claims 1 to 13.
17. A computer program adapted to perform the steps of the method as claimed in any of claims 1 to 13.
18. A data carrier adapted to store a computer program as claimed in claim 17.
19. A master computational unit configured to carry out the method as claimed in any of claims 1 to 13.
20. A computational grid adapted to operate in accordance with the method as claimed in any of claims 1 to 13.
21. A scheduling system for an aggregate of computational resources adapted to operate in accordance with the method as claimed in any of claims 1 to 13.

1/2

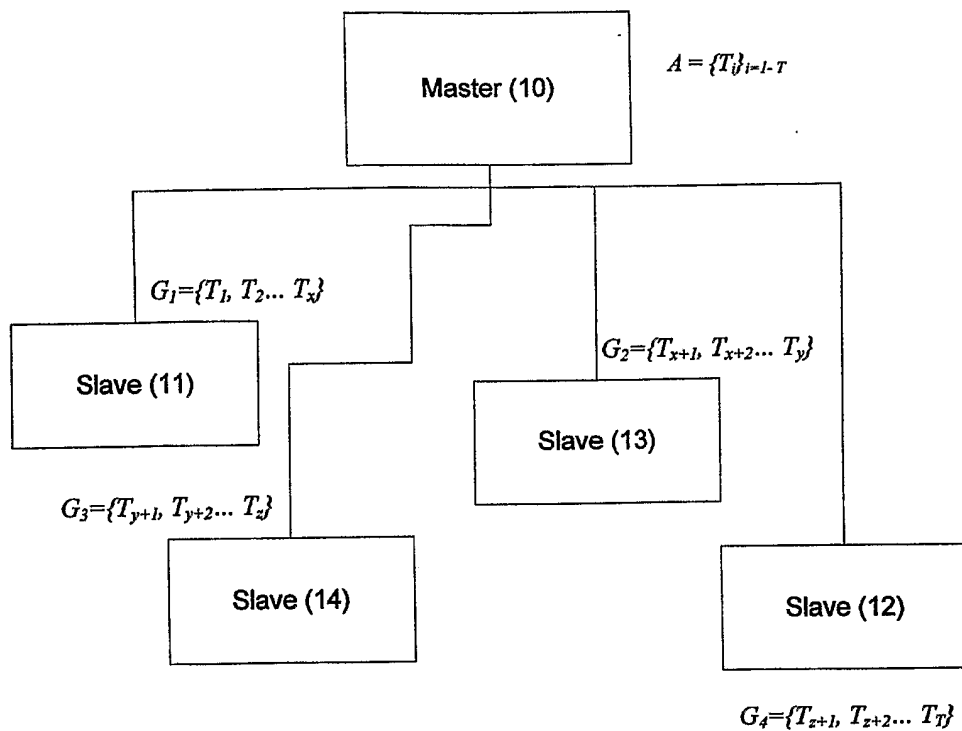


Figure 1

2/2

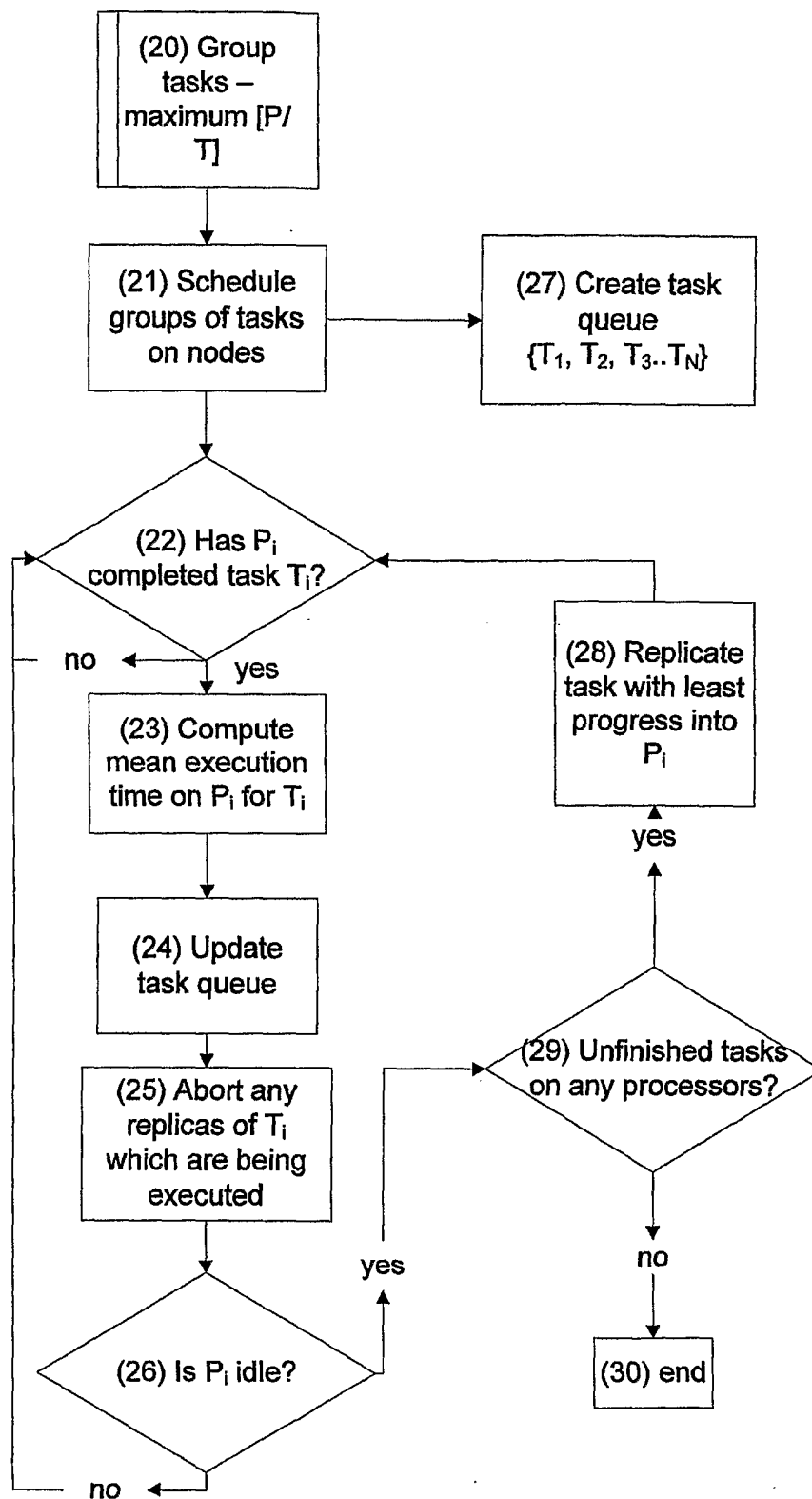


Figure 2