Rapid Distribution of Tasks on a Commodity Grid

Ladislau Bölöni[1], Damla Turgut[1], Taskin Kocak[1], Yongchang Ji[2], and Dan C. Marinescu[2]

[1] Department of Electrical and Computer Engineering
[2] School of Computer Science,
University of Central Florida,
Orlando, FL 32816
{lboloni, turgut, tkocak}@cpe.ucf.edu,{yji,dcm}@cs.ucf.edu

Abstract. The global internet is rich in commodity resources but scarce in specialized resources. We argue that a grid framework can achieve better performance if it separates the management of commodity tasks from the tasks requiring specialized resources. We show that the performance of task execution on a commodity grid is the delay of entering into execution. This effectively transforms the resource allocation problem into a routing problem.

We present an approach in which commodity tasks are distributed to the computation service providers by the use of a forwarding mesh based on randomized Hamilton cycles. We provide stochastically weighted algorithms for forwarding. Mathematical analysis and extensive simulations demonstrate that the approach is scalable and provides efficient task allocation on networks loaded up to 95% of their capacity.

1 Introduction

The computational grid (and the internet at large) is rich in commodity resources but scarce in specialized resources. There is a large number of PC class hardware (Windows and Apple desktops, Unix and Linux workstations) with typically very low resource utilization. On the other hand, there is a scarcity of specialized resources, such as supercomputers, vector processors, specialized input and output devices and so on. Typically, the need for specialized resources is dictated by the nature of the application and, less often, by the chosen implementation.

If we look at the state of the art for distributed high performance computing, we see two different approaches:

- The computational grid community develops software which manages scarce specialized resources. Although the vision of grid computing was refined several times ([4] → [6] → [5] → [2]) the main deployment of grid applications are for projects with expensive specialized hardware. Examples of testbeds are the grid projects of the National Partnership for Advanced Computational Infrastructure (NPACI) and National Computational Science Alliance

P.M.A. Sloot et al. (Eds.): EGC 2005, LNCS 3470, pp. 721–730, 2005.

(NCSA) in the US or the European DataGrid project. The grid computing projects developed at IBM, Sun and Hewlett Packard are also largely fall in this category.
– A number of distributed computing initiatives are exploiting the abundance of commodity resources for solving highly parallelizable applications. Examples are SETI@Home [16], Folding@Home [13], the cryptographic challenges sponsored by RSA laboratories [15] or the Mersenne prime search [14]. The Berkeley Open Infrastructure for Network Computing (BOINC, [1, 12]) proposes to provide a framework more general than the SETI@Home project, which can be shared by a number of projects following this pattern of interaction. These projects, which rely on donated processor time are sometimes referred as "public computing".

Both approaches target grand challenge applications. The applications targeted by the grid computing community however, are more general than the typical public computing approaches. On the other hand, SETI@Home and the related applications are highly successful in harnessing large amount of cheap computing resources.

We note that many high performance computing workflows contain both specialized and commodity tasks. For the specialized tasks, the best thing the workflow engine can do is to queue them at the appropriate specialized providers, for instance through a system such as Condor [11]. For commodity subtasks however, this approach is not appropriate. There are a very large number of community service providers (on the order of millions), which makes it difficult to deploy any kind of centralized distribution system.

We note that if a task is executed on a commodity hardware, the main determining factor of the termination time is the time at which the task is taken into execution. Furthermore, given the abundance of the commodity resources, it is likely that if a task needs to be queued at a certain host, it is almost sure that somewhere on the internet there is a task which can take it into execution immediately. Under this assumption, the task allocation problem is reduced to a specialized routing problem. A similar idea is proposed in [7, 3]. The Wire Speed Grid Project at the University of Chalmers [17], proposes an architecture in which the task allocation is performed in a hardware accelerated manner on the network routers. As our tasks have a relatively long execution time, an application layer implementation would provide the same benefits.

2 Commodity Components in Grand Challenge Applications

Grand challenge applications range from the application of relatively simple algorithms on massive amounts of data (such as the SETI@Home project), to exhaustive search of a complex combinatorial problems with small amounts of input and output data (e.g. cryptographic analysis). Many of the high performance applications however, are what we call *grid workflows*. Problems with

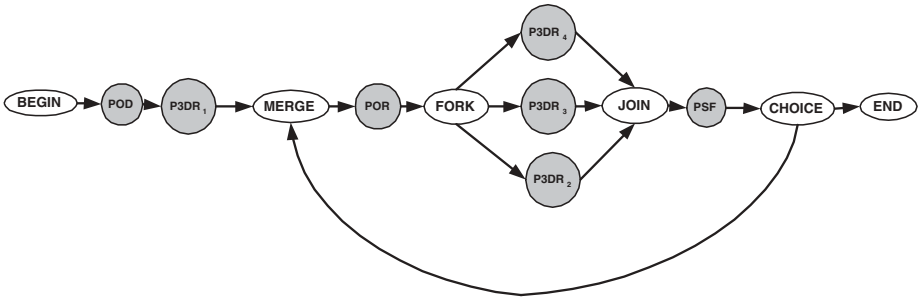

Fig. 1. Workflow for 3D virus structure reconstruction based on 2D electronmicroscope data

significant scientific and commercial interest such as predicting trajectories of hurricanes (WRF, ROMS), virus structure reconstruction, protein folding, DNA sequencing for individuals or designing custom drugs fall in this category.

Grid workflows involve a sequence of steps, such as data collection, filtering, computation, modelling and visualization. They frequently require interaction with the user in form of computation steering. The process can involve testing alternative hypothesis, thus the execution sequence can vary between individual runs. These problems are usually described as a workflow model of directed acyclic graphs, although cycles are sometimes necessary. The nodes of the graph are subtasks with different resource requirements.

Case Study: Structural Virology Application. In the following, we describe a typical grid application from the field of structural virology with which the authors have extensive experience. The 3D atomic structure determination of macromolecules based upon electron microscopy [9, 10, 8] consists of the following steps:

1. Extract individual particle projections from micrographs and identify the center of each projection.
2. Determine the orientation of each projection.
3. Carry out the 3D reconstruction of the electron density of the macromolecule.
4. Dock an atomic model into the 3D density map.

Steps 2 and 3 are executed iteratively until the 3D electron density map cannot be further improved at a given resolution; then the resolution is increased gradually. The number of iterations for these steps is in the range of hundreds and one cycle of iteration for a medium size virus may take several days. Typically it takes months to obtain a high resolution electron density map. Then Step 4 of the process can be pursued. Once we have a detailed electron density map of the virus structure, we can proceed to atomic level modelling, namely placing of groups of atoms, secondary, tertiary, or quaternary structures on the electron density maps.

The grid workflow for this procedure is described in Figure 1. The experimental data is collected using an electron microscope, and the initial input data

is 2D virus projections extracted from the micrographs. The goal of the computation is to construct a 3D model of the virus at specified resolution or the finest resolution possible given the physical limitations of the experimental instrumentation. First, we determine the initial orientation of individual views using an "ab initio" orientation determination program called `POD`. Then, we construct an initial 3D density model using our parallel 3D reconstruction program called `P3DR`. Next, we execute an iterative computation consisting of multi-resolution orientation refinement followed by 3D reconstruction. The program for orientation refinement is called `POR`.

In order to determine the resolution, the input data is split in a number of streams. For each stream, we construct a model of the electron density maps and determine the resolution by correlating the models with a program called `PSF`. The iterative process stops whenever no further improvement of the electron density map is noticeable or the goal which we specified is reached.

Although the grid workflow contains multiple data dependencies, it has components which can be executed on commodity hardware. The data acquisition step requires a computer connected to an electron microscope, and significant human work. The `POD` and `PSF` steps are an parallel programs, utilizing the message passing interface (MPI) and they require machines with very fast networking capabilities. The cheapest hardware which would still do the work is a Beowulf cluster of 32 or more computers and Gigabit Ethernet interconnects. The `P3DR` steps however are parallelizable as they will be correlated only in the next step of the workflow. Although computationally intensive, they can be run on commodity hardware. We need to note however, that components of the workflow requiring specialized hardware depend on results coming from commodity tasks.

3 System Architecture

The participants in the distributed task allocation algorithm are:

Application Client (AC). A host which desires to run a grid application, some part of which is expressible as task solvable by a commodity algorithm. The application client is usually controlled by a human user.

Commodity Resource Providers (CRP). Computers which can run one or more of the algorithms in the commodity algorithm server.

Distribution Nodes (DN). Computers which are able to forward tasks according to the distribution policy. All the CRPs are also distribution nodes, but a grid deployment might introduce distribution nodes to help the distribution of the packets. The application client needs to be in contact with at least one distribution node, which represents the entry point into the network.

Commodity Algorithm Server (CAS). A file service system which provides the standard implementation algorithms. This is normally a simple FTP or HTTP based service with a specific naming convention.

Commodity Solution Checker (CSC). A trusted web service which given a canonic description of a task and a proposed solution checks if the proposed

solution is an acceptable solution of the task. The CSC enables us to use CRPs with lower levels of trust. For some algorithms this check implies independent execution of the algorithm and the comparisor of the results. For many algorithms, the result can be verified without repeated execution.

The general process of the algorithm is as follows:

(1) The AC formulates a commodity problem as a task packet and sends it to one of its entry points.
(2) When a packet reaches a distribution node which is also a CRP, it is either bidded for its execution, or distributed/forwarded according to a *distribution policy*. The bidding is sent to the AC and a preliminary allocation is done as a soft state. The reply deadline is specified in the bid, and it is a relatively short period of time (at most several minutes).
(3) The AC sends a task award packet, containing the descriptions of the access methods of the application input. This might be contained in the confirmation packet itself or it can be a remote reference, accesible by protocols such as GridFTP. The bid might contain some setup information, such as whether the bidder needs to download the required algorithm fro the CAS or it has it in its local cache.
(4) The CRP starts the task execution process and sends a TASK_STARTED confirmation packet.
(5) [Optional] Algorithm download. If the provider does not have the required algorithm installed, it can download it from a trusted algorithm provider.
(6) [Optional] Data preparation. The input data of the process is loaded by the application using the GridFTP protocol. If the application input is very small, it can be sent in the task award packet.
(7) The CRP executes the algorithm on the specified data. It uploads the results to the locations indicated in the task specification packet. In case of success it reports to the application with a TASK_TERMINATED Packet. The CRP then becomes available for processing other tasks.

4 The n-Cycle Task Distribution Algorithm

The goal of the task distribution algorithm is to deliver tasks to commodity resource providers. With the number of CRPs involved (on the order of millions), scalability is of utmost importance. Having millions of hosts changing their availability on a minute-per-minute basis centralized algorithms based on global information are not appropriate.

The n-Cycle algorithm we propose uses only limited local information, it is virtually indefinitely scalable and performs efficient task distribution for grids loaded as high as 95% of their nominal capacity. The algorithm can be divided in two parts: the creation and maintenance of the forwarding mesh and the forwarding algorithm.

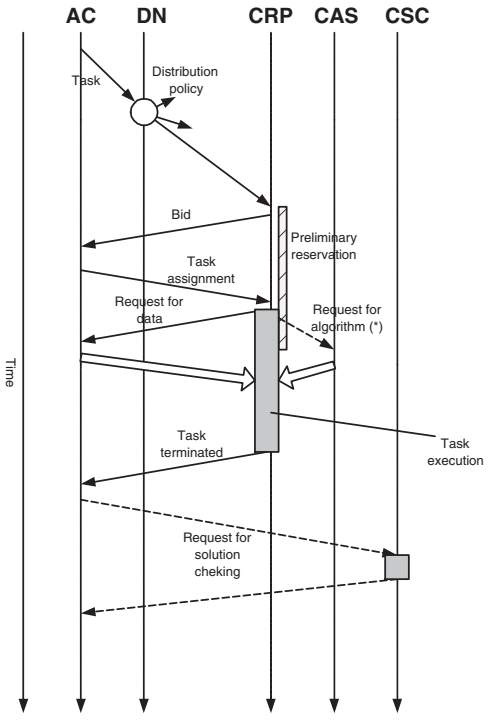

Fig. 2. The flow of the task allocation process

4.1 Creation and Maintenance of the Forwarding Mesh

The n-Cycle algorithm creates a forwarding mesh comprised of directional links. For any link $A \rightarrow B$, we will have task forwarded from A to B and status information propagated from B to A. The links of the forwarding mesh form n separate Hamiltonian cycles connecting all the elements in the grid node. The cycles are formed randomly, we are not interested in optimizing the length of the cycle. The randomness of the cycles is an important part of the algorithm. Figure 3 shows a 3-Cycle forwarding mesh on a grid of 5 nodes. For any n-Cycle mesh, every individual node will have n nodes "upstream" and n nodes "downstream" from it. The node forwards tasks to the upstream nodes and receives status updates from them. Similarly, the node receives tasks from the downstream nodes and forwards status updates to them.

4.2 Distributing Tasks on the Forwarding Mesh

One of the remarkable properties of the n-Cycle forwarding mesh is that a significant majority of the nodes can be reached by only $log_n(|W|)$ hops.

We can design a *random wandering* task allocation algorithm, with the following rule: if current host is free, take the incoming task into execution. If not,

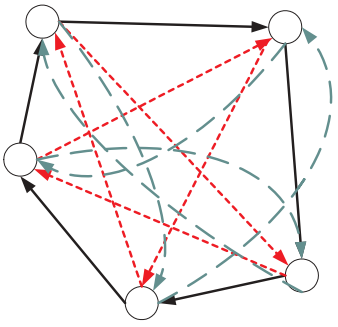

Fig. 3. A 3-Cycle forwarding mesh on a network of 5 nodes

then forward randomly to one of the uplink nodes. As we showed before, we are interested in bringing the task into execution as quickly as possible, which means that we need to minimize the number of hops.

For a random wandering algorithm, the number of hops depends on the average load of the network p. In a first approximation, for any number of hops h, the probability that a node will be allocated in less than h hops is $(1-n)^h$. Although this approach leads to satisfactory average values as long as the load is not getting close to 100%, the maximum values can be (potentially) indefinitely long. The advantage of a random wandering algorithm is that it operates without any information about the state of the network.

In the following we introduce a *weighted stochastic algorithm* which uses information collected from upstream nodes in the forwarding decision. In our simulation studies, we show that this algorithm leads to significantly better performance with an acceptable cost. Every node maintains its weight w which intuitely represents the desireability of the node as a forwarding target for a task. The weight w is composed in equal parts from (a) the ability of the node to receive a task for execution (b) the weights of the nodes downstream from the node. The weight w is propagated to the upstream nodes. A change in the weight is propagated only if it exceeds a threshold δ, preventing floods of updates.

At any given node, a task is either taken into execution (if the node is free), or forwarded to one of the upstream nodes with a probability proportional with their weight (as seen by the current node). The complete algorithm is presented in Algorithm 1.

5 Simulation

We have used the YAES [18] simulation framework to simulate the behavior of the algorithm. Table 1 illustrates the input and output parameters of the simulation as specified in the YAES configuration files.

Algorithm 1. Weighted stochastic task forwarding

When task T received by node N
  If STATUS == free
    take T into execution
    STATUS == busy
  Else
    forward to upstream node i with probability $\frac{w_i}{\sum_{k=1,n} w_k}$
  calculate new weight $w_{new} = f(STATUS, w_i)$
  If $|w - w_{new} < \delta|$
    send the new weight to all upstream nodes
    $w = w_{new}$
When weight w received from i-th downstream node
  $w_i = w$
  calculate new weight $w_{new} = f(STATUS, w_i)$
  If $|w - w_{new} < \delta|$
    send the new weight to all upstream nodes
    $w = w_{new}$

Table 1. YAES simulation parameters

Input parameters	
Number of grid nodes	100,000
Forwarding mesh	5-Cycle
Task arrival	Poisson-distributed arrival, mean 10...200 tasks/sec
Task servicing	Normally distributed, mean 60 sec/task
Simulation time	5000 seconds
Output parameters (Measurements)	
Hops per task	Number of hops a task is forwarded until it finds a host for execution (avg, max)
Average load	Ratio of busy vs. total nodes
Discarded tasks	Number of tasks which were discarded

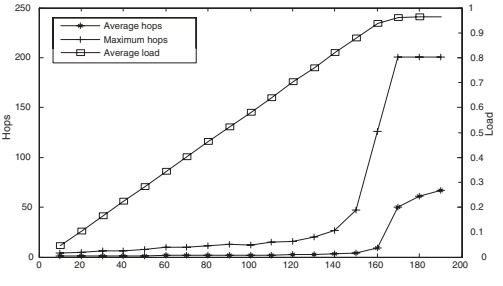

Fig. 4. Average number of hops, maximum number of hops and network load using weighted stochastic forwarding on the n-Cycle forwarding mesh

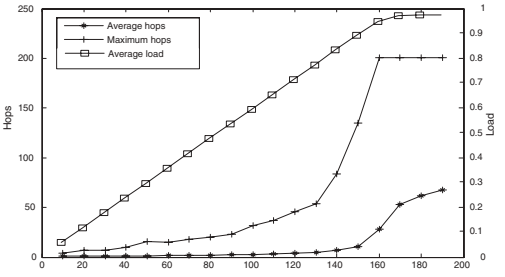

Fig. 5. Average number of hops, maximum number of hops and network load using random forwarding on the n-Cycle forwarding mesh

Figure 4 presents the average and maximum number of hops it takes for a task to be allocated and the total network load in function of the average number of arriving tasks. We note that both the average and maximum number of hops is staying virtually constant at a very low number, up to loads approaching 95%. At that moment the number of hops increases dramatically as the algorithm struggles to find free nodes in an overwhelmingly busy network.

The relatively constant number of nodes for moderate loads is explained by the single insertion point. The nodes closer to the insertion point will be filled in relatively quickly, so the majority of tasks need to "hop over" the busy nodes in this area. A good approximation of the size of this constant value is $log_N(n)$ which in our case is $log_5(10000)$, approximately 5.7. If we choose a random insertion point, we will obtain a diagram with a similar shape, but with an average number of hops for lightly loaded networks much smaller (about 1-2 hops).

In a different simulation run, Figure 5 presents the random walking algorithm. For small loads, this algorithm also shows very good results (due to the randomizing nature of the N-Cycle mesh). However, for greater loads, the maximum number of hops start to increase. For instance, at load of 90% the maximum will be as high as 100 hops vs. about 20 hops for the stochastically weighted algorithm.

6 Conclusions and Future Work

In this paper, we introduced an algorithm for allocating commodity tasks on a computational grid. Our analysis and simulation studies show that (a) the algorithm is scalable (b) it proved to be very efficient in allocating tasks to free computational service providers.

Our future work includes more extensive mathematical analysis of the algorithms. It is of special interest on modeling the influence of the estimation of the w_i values of the upstream nodes, as higher accuracies for these values require higher message traffic on the mesh. We also plan to extend the algorithm to heterogeneous networks.

References

1. D. P. Anderson. Public computing: Reconnecting people to science. In *Proceedings of the Conference on Shared Knowledge and the Web*, Nov 2003.
2. M. Baker. Ian Foster on Recent Changes in the Grid Community. URL `http://dsonline.computer.org/0402/d/o2004a.htm`.
3. B.Liljeqvist and L.Bengtsson. Grid computing distribution using network processors. In *Proc. of the 14th IASTED Parallel and Distributed Computing Conference*, Nov 2002.
4. I. Foster and C. Kesselman, editors. *The Computational Grid: Blueprint to a New Computer Infrastructure*. Morgan-Kauffman, 1998.
5. I. Foster, C. Kesselman, J. Nick, and S. Tuecke. The Physiology of the Grid: An open grid services architecture for distributed systems integration. URL `http://www.globus.org/research/papers/ogsa.pdf`.
6. I. Foster, C. Kesselman, and S. Tuecke. The anatomy of the grid: Enabling scalable virtual organizations. *International Journal of Supercomputer Applications*, 15(3), 2001.
7. A. Iamnitchi and I. Foster. On fully decentralized resource discovery in grif environments. In *Proceedings of the International Workshop on Grid Computing, Denver, CO, November 2001*, 2001.
8. Y. Ji, D. C. Marinescu, W. Zhang, and T. S. Baker. Orientation refinement of virus structures with unknown symmetry. In *Proceedings of the 17th Ann. Int'l Parallel and Distrib. Processing Symposium Nice, France*. IEEE Press, 2003.
9. D. C. Marinescu and Y. Ji. A computational framework for the 3d structure determination of viruses with unknown symmetry. *Journal of Parallel and Distributed Computing*, 63(7-8):738–758, 2003.
10. D. C. Marinescu and Y. Ji. A computational framework for the `3d` structure determination of viruses with unknown symmetry. *Journal of Parallel and Distributed Computing*, 63:738–758, 2003.
11. D. Thain, T. Tannenbaum, and M. Livny. Condor and the grid. In F. Berman, G. Fox, and T. Hey, editors, *Grid Computing: Making the Global Infrastructure a Reality*. John Wiley & Sons Inc., December 2002.
12. Berkeley Open Infrastructure for Network Computing. URL `http://boinc.berkeley.edu/`.
13. Folding@Home project. URL `http://www.stanford.edu/group/pandegroup/folding/`.
14. Mersenne Prime search. URL `http://www.mersenne.org/prime.htm`.
15. RSA Challenge. URL `http://www.rsasecurity.com/rsalabs/challenges/`.
16. SETI@Home project. URL `http://setiathome.ssl.berkeley.edu/`.
17. The Wire Speed Grid project. URL `http://www.ce.chalmers.se/staff/labe/Wire Speed Grid Project.htm`.
18. YAES: Yet Another Extensible Simulator. URL `http://netmoc.cpe.ucf.edu/Yaes/Yaes.html`.